

# TARTALOMJEGYZÉK

Előszó a magyar kiadáshoz .....	21
Előszó .....	23
<b>I. rész. Mesterséges intelligencia .....</b>	<b>29</b>
<b>1. Bevezetés .....</b>	<b>31</b>
1.1. Mi az MI? .....	31
Emberi módon cselekedni: Turing-teszt megközelítés .....	32
Emberi módon gondolkodni: a kognitív modellezés .....	33
Racionálisan gondolkodni: a gondolkodás törvénye .....	34
Racionálisan cselekedni: a racionális ágens .....	35
1.2. A mesterséges intelligencia alapjai .....	36
Filozófia (i. e. 428-tól napjainkig) .....	36
Matematika (kb. 800-tól napjainkig) .....	38
Gazdaságtan (1776-tól napjainkig) .....	40
Neurális tudományok (1861-tól napjainkig) .....	41
Pszichológia (1879-tól napjainkig) .....	44
Számítógépes tudományok (1940-tól napjainkig) .....	45
Irányításmélet és kibernetika (1948-tól napjainkig) .....	46
Nyelvészeti (1957-tól napjainkig) .....	47
1.3. A mesterséges intelligencia története .....	48
A mesterséges intelligencia érlelődése (1943–1955) .....	48
A mesterséges intelligencia megszületése (1956) .....	49
Korai lelkesedés, nagy elvárások (1952–1969) .....	50
Egy adag realitás (1966–1973) .....	53
Tudásalapú rendszerek: a hatalom kulcsa? (1969–1979) .....	55
Az MI iparrá válik (1980-tól napjainkig) .....	57
A neurális hálók visszatérése (1986-tól napjainkig) .....	57
Az MI tudománnyá válik (1987-tól napjainkig) .....	58
Az intelligens ágensek kialakulása (1995-től napjainkig) .....	59
1.4. A mesterséges intelligencia jelenlegi helyzete .....	60
1.5. Összefoglalás .....	62
Irodalmi és történeti megjegyzések .....	62
Feladatok .....	63

<b>2.</b>	<b>Intelligens ágensek .....</b>	<b>66</b>
2.1.	Bevezetés .....	66
2.2.	jó viselkedés: a racionalitás koncepciója .....	68
	Teljesítménymértékek .....	69
	Racionalitás .....	70
	Mindentudás, tanulás és autonómia .....	71
2.3.	A Környezetek természete .....	72
	A környezet meghatározása .....	72
	A környezetek tulajdonságai .....	75
2.4.	Az intelligens ágensek struktúrája .....	79
	Ágensprogramok .....	79
	Egyeszerű reflexszerű ágensek .....	81
	Modellalapú reflexszerű ágensek .....	83
	Célorientált ágensek .....	84
	Hasznosságorientált ágensek .....	86
	Tanuló ágensek .....	86
2.5.	Összefoglalás .....	89
	Irodalmi és történeti megjegyzések .....	90
	Feladatok .....	92
<b>II. rész. Problémamegoldás .....</b>		<b>95</b>
<b>3.</b>	<b>Problémamegoldás kereséssel .....</b>	<b>97</b>
3.1.	Problémamegoldó ágensek .....	97
	Jól definiált problémák és megoldások .....	100
	A problémák megfogalmazása .....	101
3.2.	Példaproblémák .....	102
	Játékproblémák .....	103
	Valósvilág-beli problémák .....	106
3.3.	Megoldások keresése .....	108
	A problémamegoldó hatékonyság mérése .....	111
3.4.	Nem informált keresés .....	112
	Szélességi keresés .....	112
	Egyenletes költségű keresés .....	114
	Mélységi keresés .....	115
	Mélységekorlátozott keresés .....	117
	Iteratívan mélyülő mélységi keresés .....	117
	Kétirányú keresés .....	119
	A keresési stratégiák összehasonlítása .....	121
3.5.	Az ismételt állapotok elkerülése .....	121
3.6.	Keresés részleges információ mellett .....	123
	Szenzor nélküli problémák .....	124
	Eshetőségi problémák .....	126
3.7.	Összefoglalás .....	127
	Irodalmi és történeti megjegyzések .....	128
	Feladatok .....	130

<b>4.</b>	<b>Informált keresési és felfedező módszerek . . . . .</b>	<b>136</b>
4.1.	Informált (heurisztikus) keresési stratégiák . . . . .	136
	A mohó legjobbat-először keresés . . . . .	137
	A* keresés: a teljes becsült útköltség minimalizálása . . . . .	139
	Memóriakorlátozott heurisztikus keresés . . . . .	144
	Tanulunk, hogy jobban keressünk! . . . . .	147
4.2.	Heurisztikus függvények . . . . .	148
	A heurisztikus függvény pontosságának hatása a megoldás hatékonyságára . . . . .	149
	Elfogadható heurisztikus függvények kitalálása . . . . .	150
	A heurisztikus függvény tanulása tapasztalatból . . . . .	153
4.3.	Lokális kereső algoritmusok és optimalizációs problémák . . . . .	153
	Hegymászó keresés . . . . .	155
	Szimulált lehűtés . . . . .	158
	Lokális nyáláb keresés . . . . .	159
	Genetikus algoritmusok . . . . .	160
4.4.	Lokális keresés folytonos terekben . . . . .	163
4.5.	Online kereső ágensek és ismeretlen környezetek . . . . .	166
	Online keresési problémák . . . . .	167
	Online kereső ágensek . . . . .	169
	Online lokális keresés . . . . .	170
	Tanulás online keresés során . . . . .	172
4.6.	Összefoglalás . . . . .	173
	Irodalmi és történeti megjegyzések . . . . .	174
	Feladatok . . . . .	179
<b>5.</b>	<b>Kényszerkielégítési problémák . . . . .</b>	<b>183</b>
5.1.	Kényszerkielégítési problémák . . . . .	183
5.2.	A visszalépéses keresés alkalmazása kényszerkielégítési problémákra . . . . .	187
	Változó- és értékrendezés . . . . .	190
	Az információ terjesztése a kényszereken keresztül . . . . .	191
	Intelligens visszalépés: visszanézni . . . . .	195
5.3.	Lokális keresés kényszerkielégítési problémáknál . . . . .	197
5.4.	A problémák struktúrája . . . . .	199
5.5.	Összefoglalás . . . . .	202
	Irodalmi és történeti megjegyzések . . . . .	203
	Feladatok . . . . .	206
<b>6.</b>	<b>Keresés ellenséges környezetben . . . . .</b>	<b>209</b>
6.1.	Kétszemélyes játékok . . . . .	209
6.2.	Optimális döntések kétszemélyes játékokban . . . . .	211
	Optimális stratégiák . . . . .	212
	A minimax algoritmus . . . . .	213
	Optimális döntések többszemélyes játékokban . . . . .	214
6.3.	Alfa-béta nyerés . . . . .	216
6.4.	Nem tökéletes, valós idejű döntések . . . . .	219
	Kiértékelő függvények . . . . .	220

A keresés levágása .....	222
6.5. Véletlen elemet is tartalmazó játékok .....	224
Az állás kiértékelése véletlen csomópontokat tartalmazó játékok esetén ..	226
A várhatóminimax komplexitása .....	226
Kártyajátékok .....	228
6.6. A jelenleg legfejlettebb játékprogramok .....	229
6.7. Értékelés .....	233
6.8. Összefoglalás .....	235
Irodalmi és történeti megjegyzések .....	236
Feladatok .....	240
<b>III. rész. Tudás és következtetés .....</b>	<b>245</b>
<b>7. Logikai ágensek .....</b>	<b>247</b>
7.1. A tudásbázisú ágens .....	248
7.2. A wumpus világ .....	250
7.3. A logika .....	253
7.4. Az ítéletkalkulus: egy nagyon egyszerű logika .....	258
Szintaxis .....	258
Szemantika .....	260
Egy egyszerű tudásbázis .....	262
Következtetés .....	262
Ekvivalencia, érvényesség és kielégíthetőség .....	264
7.5. Az ítéletkalkulus következtetési mintái .....	265
Rezolúció .....	267
Előre- és hátrafelé láncolás .....	272
7.6. Hatékony ítéletkalkulus következtetés .....	276
Egy teljes visszalépéses algoritmus .....	276
Lokális keresés algoritmus .....	278
Nehéz kielégíthetőségi problémák .....	279
7.7. Ítéletlogikát alkalmazó ágensek .....	281
Cspdák és wumpusok megtalálása logikai következtetés felhasználásával ..	281
A hely és az irány nyomkövetése .....	282
Az áramkörön alapuló ágens .....	283
Összehasonlítás .....	287
7.8. Összefoglalás .....	289
Irodalmi és történeti megjegyzések .....	290
Feladatok .....	293
<b>8. Elsőrendű logika .....</b>	<b>297</b>
8.1. Még egyszer a reprezentációról .....	297
8.2. Az elsőrendű logika szintaxisa és szemantikája .....	302
Az elsőrendű logika modelljei .....	302
Szimbólumok és interpretációk .....	303
Termek .....	305
Atomi mondatok .....	306

Összetett mondatok .....	306
Kvantorok .....	307
Univerzális kvantor ( $\forall$ ) .....	307
Egzisztenciális kvantorok ( $\exists$ ) .....	308
Egymásba ágyazott kvantorok .....	309
Az $\forall$ és az $\exists$ kapcsolata .....	310
Egyenlőség .....	311
8.3. Az elsőrendű logika használata .....	311
Kijelentések és lekérdezések az elsőrendű logikában .....	312
A rokonsági tárgyterület .....	312
Számok, halmazok és listák .....	314
A wumpus világ .....	316
8.4. Tudástervezés az elsőrendű logikában .....	320
A tudástervezés folyamata .....	320
Az elektronikus áramkörök tárgyterülete .....	322
A feladat meghatározása .....	322
A releváns tudás összegyűjtése .....	322
A szótár meghatározása .....	323
A tárgyterülettel kapcsolatos általános tudás kódolása .....	324
A problémáspecifikus példányok kódolása .....	325
Lekérdezések megfogalmazása a következtetési eljárás felé .....	325
Hibák kiszűrése a tudásbázisból .....	326
8.5. Összefoglalás .....	326
Irodalmi és történeti megjegyzések .....	327
Feladatok .....	328
 9. Következtetés elsőrendű logikában .....	333
9.1. Ítéletlogikai következtetés kontra elsőrendű logikai következtetés .....	333
Kvantorokra vonatkozó következtetési szabályok .....	334
Redukálás ítéletlogikára .....	335
9.2. Egyesítés és kiemelés .....	336
Egy elsőrendű következtetési szabály .....	337
Egyesítés .....	338
Tárolás és visszakeresés .....	340
9.3. Előrefelé láncolás .....	342
Elsőrendű határozott klózok .....	342
Egy egyszerű előrefelé láncolási algoritmus .....	344
Hatókony előrefelé láncolás .....	346
9.4. Hátrafelé láncolás .....	350
Egy hátrafelé láncolási algoritmus .....	350
Logikai programozás .....	352
A logikai programok hatékony megvalósítása .....	354
Redundáns következtetés és végtelen hurkok .....	356
Korlátozott logikai programozás .....	358
9.5. Rezolúció .....	359
Az elsőrendű logika konjunktív normál formája .....	360

A rezolúciós következtetési szabály .....	362
Példabizonyítások .....	362
A rezolúció teljessége .....	365
Az egyenlőség kezelése .....	368
Rezolúciós stratégiák .....	369
Tételbizonyítók .....	371
<b>9.6. Összefoglalás.</b> .....	<b>375</b>
Irodalmi és történeti megjegyzések.....	376
Feladatok.....	382
<b>10. Tudásbázis reprezentáció.</b> .....	<b>387</b>
<b>10.1. Ontológiaszervezés .</b> .....	<b>387</b>
<b>10.2. Kategóriák és objektumok</b> .....	<b>390</b>
Fizikai összetétel.....	391
Mérések.....	393
Szubsztanciák és objektumok .....	395
<b>10.3. Cselekvések, szituációk és események</b> .....	<b>396</b>
A szituációkalkulus ontológiája .....	396
Cselekvések leírása a szituációkalkulusban .....	398
A reprezentációs probléma megoldása .....	400
A következtetési keretprobléma megoldása .....	402
Idő- és eseménykalkulus .....	403
Általánosított események .....	404
Folyamatok .....	406
Intervallumok .....	407
Folyó események és objektumok .....	409
<b>10.4. Mentális események és mentális objektumok.</b> .....	<b>410</b>
A hiedelmek formális elmélete .....	410
Tudás és hiedelem.....	412
Tudás, idő és cselekvés .....	413
<b>10.5. Az internetes bevásárlás világa.</b> .....	<b>414</b>
Ajánlatok összehasonlítása .....	418
<b>10.6. Következtető rendszerek kategóriák számára.</b> .....	<b>419</b>
Szemantikus hálók .....	419
Leíró logikák.....	423
<b>10.7. Következtetés alapértelmezett információval.</b> .....	<b>424</b>
Nyitott és zárt világok .....	425
Negálás mint kudarc és stabil modell szemantika .....	427
Körülírás és alapeseti logika.....	428
<b>10.8. Igazság-karbantartó rendszerek.</b> .....	<b>431</b>
<b>10.9. Összefoglalás.</b> .....	<b>433</b>
Irodalmi és történeti megjegyzések.....	434
Feladatok.....	441

<b>IV. rész. Tervkészítés .....</b>	<b>449</b>
<b>11. Tervkészítés .....</b>	<b>451</b>
<b>11.1. A tervkészítési probléma.</b>	<b>451</b>
A tervkészítési problémák nyelve .....	453
Kifejezőképesség és kiterjesztések .....	455
Példa: Légi teherszállítás .....	456
Példa: A pótkérék probléma .....	457
Példa: A kockavilág .....	458
<b>11.2. Tervkészítés állapottér-kereséssel.</b>	<b>459</b>
Előrefelé keresés az állapottérben .....	459
Hátrafelé keresés az állapottérben .....	461
Állapottér-keresési heurisztikák .....	462
Heurisztikák a részben rendezett tervkészítésre .....	471
<b>11.3. Részben rendezett tervkészítés .....</b>	<b>464</b>
Példa a részben rendezett tervkészítésre .....	468
Részben rendezett tervkészítés kötetlen változókkal .....	470
<b>11.4. Tervkészítési gráfok .....</b>	<b>472</b>
Tervkészítési gráfok heurisztikus becslésekre .....	475
A Graphplan algoritmus .....	476
A Graphplan algoritmus leállása .....	478
<b>11.5. Tervkészítés ítéletlogikával.</b>	<b>479</b>
Tervkészítési problémák ítéletlogikai leírása .....	480
Az ítéletlogikai leírás bonyolultsága .....	483
<b>11.6. A tervkészítési módszerek elemzése.</b>	<b>485</b>
<b>11.7. Összefoglalás.</b>	<b>486</b>
Irodalmi és történeti megjegyzések .....	487
Feladatok .....	490
<b>12. Tervkészítés és cselekvés a való világban .....</b>	<b>495</b>
<b>12.1. Idő, ütemezés és erőforrások.</b>	<b>495</b>
Ütemezés erőforráskorlátokkal .....	498
<b>12.2. Hierarchikus feladatháló tervkészítés .....</b>	<b>500</b>
A cselekvésdekompozíciók reprezentációja .....	501
A tervkészítő módosítása a dekompozíciók kezeléséhez .....	504
Elemzés .....	506
<b>12.3. Tervkészítés és cselekvés nemdeterministikus problémakörökben .....</b>	<b>509</b>
<b>12.4. Feltételes tervkészítés .....</b>	<b>511</b>
Feltételes tervkészítés teljesen megfigyelhető környezetekben .....	512
Feltételes tervkészítés részlegesen megfigyelhető környezetekben .....	516
<b>12.5. Végrehajtás monitorozása és újratervezése.</b>	<b>520</b>
<b>12.6. Folytonos tervkészítés .....</b>	<b>524</b>
<b>12.7. Multiágens tervkészítés .....</b>	<b>529</b>
Kooperáció: közös célok és tervezek .....	529
Többtestű tervezés .....	530
Koordináló mechanizmusok .....	532

Versengés .....	533
12.8. Összefoglalás.....	534
Irodalmi és történeti megjegyzések.....	535
Feladatok.....	539
<b>V. rész. Bizonytalan tudás és következtetés.....</b>	<b>543</b>
<b>13. Bizonytalanság.....</b>	<b>545</b>
13.1. Cselekvés bizonytalan tudás esetén .....	545
Bizonytalan tudás kezelése .....	546
Bizonytalanság és racionális döntések .....	548
Egy döntéselméleti ágens tervezése .....	549
13.2. Valószínűségi alapfogalmak .....	550
Állítások .....	550
Elemi események .....	551
A priori valószínűség .....	552
Feltételes valószínűség .....	554
13.3. Valószínűségi axiómák .....	555
A valószínűségi axiómák használata.....	556
Ami a valószínűségi axiómákat indokolja .....	557
13.4. Teljes együttes valószínűség-eloszláson alapuló következtetés .....	559
13.5. Függetlenség .....	562
13.6. A Bayes-tétel és használata.....	563
Bayes tételek alkalmazása: egyszerű eset.....	564
A Bayes-tétel alkalmazása: több együttes tény figyelembevétele .....	565
13.7. A wumpus világ újralátogatása.....	567
13.8. Összefoglalás.....	571
Irodalmi és történeti megjegyzések.....	572
Feladatok .....	574
<b>14. Valószínűségi következtetés .....</b>	<b>578</b>
14.1. A tudás reprezentálása bizonytalanság esetén .....	578
14.2. A Bayes-hálók szemantikája.....	581
Az együttes valószínűség-eloszlás filogény leírása .....	581
Feltételes függetlenségi relációk Bayes-hálókban .....	585
14.3. Feltételes eloszlások hatékony reprezentációja.....	587
Bayes-hálók folytonos változókkal .....	588
14.4. Egzakt következtetés Bayes-hálókban .....	591
Következtetés felsorolással .....	592
A változó eliminációs algoritmus .....	594
Az egzakt következtetés komplexitása .....	596
Csoporthozó algoritmusok .....	597
14.5. Közelítő következtetés Bayes-hálókban .....	598
Közvetlen mintavételezési módszerek .....	598
Következtetés Markov-lánc szimulációval .....	604
14.6. Elsőrendű reprezentációk valószínűségi kiterjesztése .....	607

14.7.	Egyéb módszerek a bizonytalan környezetben történő következtetéshez . . . . .	611
	Bizonytalansági következtetés szabályalapú eljárásokkal . . . . .	612
	Az ismerethiány reprezentálása: a Dempster–Shafer-elmélet . . . . .	614
	A meghatározatlanság reprezentálása: fuzzy halmazok és logikák . . . . .	615
14.8.	Összefoglalás . . . . .	617
	Irodalmi és történeti megjegyzések . . . . .	618
	Feladatok . . . . .	623
15.	<b>Időbeli valószínűségi következtetés . . . . .</b>	<b>627</b>
15.1.	Idő és bizonytalanság . . . . .	627
	Állapotok és megfigyelések . . . . .	628
	Stacionárius folyamatok és a Markov-feltétel . . . . .	629
15.2.	Következtetés időbeli modellekben . . . . .	632
	Szűrés és előrejelzés . . . . .	633
	Simítás . . . . .	635
	A legalovszínűbb sorozat megtalálása . . . . .	638
15.3.	Rejtett Markov-modellek . . . . .	639
	Egyszerűsített mátrix algoritmusok . . . . .	640
15.4.	Kalman-szűrők . . . . .	642
	Gauss-eloszlások frissítése . . . . .	644
	Egy egyszerű egydimenziós példa . . . . .	644
	Az általános eset . . . . .	647
	A Kalman-szűrés alkalmazhatósága . . . . .	648
15.5.	Dinamikus Bayes-hálók . . . . .	650
	DBH-k létrehozása . . . . .	651
	Egzakt következetés DBH-kban . . . . .	655
	Közeliítő következetés DBH-kban . . . . .	656
15.6.	Beszédfelismerés . . . . .	659
	Beszédhangok . . . . .	661
	Szavak . . . . .	664
	Mondatok . . . . .	666
	Egy beszédfelismerő építése . . . . .	668
15.7.	Összefoglalás . . . . .	670
	Irodalmi és történeti megjegyzések . . . . .	670
	Feladatok . . . . .	673
16.	<b>Egyszerű döntések meghozatala . . . . .</b>	<b>676</b>
16.1.	Meggyőződések és kívánságok összekapcsolása bizonytalanság esetén . . . . .	676
16.2.	A hasznosságelmélet alapjai . . . . .	678
	Megkötések a racionális preferenciákra . . . . .	678
	És aztán jött a hasznosság . . . . .	680
16.3.	Hasznosságfüggvények . . . . .	681
	A pénz hasznossága . . . . .	682
	Hasznosságskálák és a hasznosság megbecsélése . . . . .	684
16.4.	Többváltozós hasznosságfüggvények . . . . .	686
	Dominancia . . . . .	687

<b>A preferenciák rendszere és a többattribútumos hasznosság .....</b>	<b>689</b>
<b>16.5. Döntési hálók .....</b>	<b>691</b>
Döntési problémák reprezentálása döntési hálókkal.....	691
Döntési hálók kiértékelése .....	692
<b>16.6. Az információ értéke.....</b>	<b>693</b>
Egy egyszerű példa .....	694
Egy általános képlet .....	695
Az információ értékének tulajdonságai.....	696
Egy információgyűjtő ágens megvalósítása .....	697
<b>16.7. Döntéselméleti szakértő rendszerek .....</b>	<b>698</b>
<b>16.8. Összefoglalás.....</b>	<b>701</b>
Irodalmi és történeti megjegyzések.....	701
Feladatok .....	703
<b>17. Komplex döntések meghozatala .....</b>	<b>707</b>
<b>17.1. Szekvenciális döntési problémák .....</b>	<b>707</b>
Egy példa .....	707
Optimalitás szekvenciális döntési problémákban .....	710
<b>17.2. Értékiteráció .....</b>	<b>713</b>
Az állapotok hasznossága .....	713
Az értékiteráció algoritmus .....	714
Az értékiteráció konvergenciája .....	716
<b>17.3. Eljárásmódszer.....</b>	<b>718</b>
<b>17.4. Részlegesen megfigyelhető Markov döntési folyamatok .....</b>	<b>720</b>
<b>17.5. Döntéselméleti ágensek .....</b>	<b>724</b>
<b>17.6. Többágenses döntések: a játékelmélet .....</b>	<b>726</b>
<b>17.7. Működési mód tervezés .....</b>	<b>736</b>
<b>17.8. Összefoglalás.....</b>	<b>739</b>
Irodalmi és történeti megjegyzések.....	740
Feladatok .....	743
<b>VI. rész. Tanulás.....</b>	<b>747</b>
<b>18. Megfigyelések alapján történő tanulás .....</b>	<b>749</b>
<b>18.1. Tanulási formák .....</b>	<b>749</b>
<b>18.2. Induktív tanulás .....</b>	<b>751</b>
<b>18.3. Döntési fák megalkotása tanulással .....</b>	<b>754</b>
Cselekvő komponensként használt döntési fák.....	754
A döntési fák kifejezőképessége .....	755
A döntési fák példák alapján történő felépítése .....	756
Attribútumteszt-választás .....	760
A tanuló algoritmus teljesítményének becslése .....	761
Zaj és túllillesszkedés .....	763
A döntési fák alkalmazhatóságának kiterjesztése .....	765
<b>18.4. Hipotézishalmaz együttes tanulása .....</b>	<b>766</b>
<b>18.5. Miért működik a tanulás: a tanulás számítási elmélete .....</b>	<b>770</b>

Hány példára van szükség?	771
Döntési listák tanulása	773
Elemzés	775
<b>18.6. Összefoglalás</b>	<b>776</b>
Irodalmi és történeti megjegyzések	776
Feladatok	779
<b>19. A tudás szerepe a tanulásban</b>	<b>782</b>
19.1. A tanulás logikai megfogalmazása	782
Példák és hipotézisek	782
A pillanatnyilag legjobb hipotézis keresése	784
Legkisebb megkötés elvű keresés	787
<b>19.2. A tudás szerepe a tanulásban</b>	<b>791</b>
Néhány egyszerű példa	792
Néhány általános séma	793
<b>19.3. Magyarázatalapú tanulás</b>	<b>794</b>
Általános szabályok kinyerése példákból	795
A hatékonyság javítása	797
<b>19.4. Tanulás releváns információ alapján</b>	<b>799</b>
A hipotézistér meghatározása	800
Tanulás releváns információ felhasználásával	801
<b>19.5. Induktív logikai programozás</b>	<b>803</b>
Egy példa	803
Felülről lefelé tanulási módszerek	806
Induktív tanulás inverz rezolúcióval	809
Felfedezés induktív logikai programozással	811
<b>19.6. Összefoglalás</b>	<b>812</b>
Irodalmi és történeti megjegyzések	813
Feladatok	816
<b>20. Statisztikai tanulási módszerek</b>	<b>818</b>
20.1. Statisztikai tanulás	818
<b>20.2. Teljes adattal történő tanulás</b>	<b>822</b>
Maximum-likelihood paramétertanulás: diszkrét modellek	822
Naiv Bayes-modellek	825
Maximum-likelihood paramétertanulás: folytonos eset	826
Bayes-paramétertanulás	827
Bayes-hálóstruktúrák tanulása	830
<b>20.3. Rejtett változókkal történő tanulás: az EM algoritmus</b>	<b>831</b>
Nem ellenőrzött osztályozás: Gauss-eloszlások keverékének tanulása	832
Rejtett változókkal felépített Bayes-hálók tanulása	835
Rejtett Markov-modellek tanulása	838
Az EM algoritmus általános alakja	839
Bayes-hálóstruktúra tanulása rejtett változók esetén	839
<b>20.4. Példányalapú tanulás</b>	<b>841</b>
Legközelebbi-szomszéd modellek	841

<b>Kernelmódszerek . . . . .</b>	<b>843</b>
<b>20.5. Neurális hálók . . . . .</b>	<b>844</b>
A neurális háló egységei . . . . .	845
Hálóstruktúrák . . . . .	846
Egyrétegű előrecsatolt neurális hálók (perceptronok) . . . . .	848
Többrétegű előrecsatolt neurális hálók . . . . .	852
Neurális hálóstruktúrák tanulása . . . . .	856
<b>20.6. Kernelgépek . . . . .</b>	<b>857</b>
<b>20.7. Esettanulmány: kézzel írott számjegyek felismerése . . . . .</b>	<b>860</b>
<b>20.8. Összefoglalás . . . . .</b>	<b>863</b>
Irodalmi és történeti megjegyzések . . . . .	864
Feladatok . . . . .	868
<b>21. Megerősítéses tanulás . . . . .</b>	<b>873</b>
<b>21.1. Bevezetés . . . . .</b>	<b>873</b>
<b>21.2. Passzív megerősítéses tanulás . . . . .</b>	<b>875</b>
Közvetlen hasznosságbecslés . . . . .	876
Adaptív dinamikus programozás . . . . .	877
Az időbeli különbség tanulása . . . . .	879
<b>21.3. Aktív megerősítéses tanulás . . . . .</b>	<b>881</b>
Felfedezés . . . . .	882
Egy cselekvésérték-függvény tanulása . . . . .	885
<b>21.4. A megerősítéses tanulás általánosítóképessége . . . . .</b>	<b>887</b>
Alkalmazások a játékok területén . . . . .	890
Robotirányítási alkalmazások . . . . .	891
<b>21.5. Stratégiakeresés . . . . .</b>	<b>892</b>
<b>21.6. Összefoglalás . . . . .</b>	<b>895</b>
Irodalmi és történeti megjegyzések . . . . .	896
Feladatok . . . . .	899
<b>VII. rész. Kommunikáció, észlelés és cselekvés . . . . .</b>	<b>903</b>
<b>22. Kommunikáció . . . . .</b>	<b>905</b>
<b>22.1. A kommunikáció mint cselekvés . . . . .</b>	<b>905</b>
A nyelv alapjai . . . . .	906
A kommunikációt alkotó lépések . . . . .	909
<b>22.2. Formális nyelvtan az angol nyelv egy töredékére . . . . .</b>	<b>911</b>
Az $\mathcal{E}_0$ nyelvtana . . . . .	912
<b>22.3. Szintaktikai analízis (elemzés) . . . . .</b>	<b>913</b>
Hatékony elemzés . . . . .	915
<b>22.4. Kiterjesztett nyelvtanok . . . . .</b>	<b>921</b>
Igék alkategóriákba osztása . . . . .	923
Kiterjesztett nyelvtanok generálóképessége . . . . .	925
<b>22.5. Szemantikai értelmezés . . . . .</b>	<b>926</b>
Az angol nyelv egy részletének szemantikája . . . . .	927
Idő és igeidő . . . . .	928

Kvantifikálás .....	929
Pragmatikus értelmezés .....	932
Nyelv generálása DCG-kkel .....	933
<b>22.6.</b> Többérműség és feloldása .....	934
A többérműség feloldása .....	936
<b>22.7.</b> Szövegértés .....	937
Utalásfeloldás .....	937
Egy koherens szöveg struktúrája .....	939
<b>22.8.</b> A nyelvtan indukciós tanulása .....	941
<b>22.9.</b> Összefoglalás .....	942
Irodalmi és történeti megjegyzések .....	943
Feladatok .....	947
<b>23.</b> <b>Valószínűségi nyelvfeldolgozás .....</b>	<b>951</b>
<b>23.1.</b> Valószínűségi nyelvi modellek .....	951
Valószínűségi környezetfüggetlen nyelvtanok .....	954
PCFG-valószínűségek tanulása .....	956
PCFG-szabálystruktúrák tanulása .....	957
<b>23.2.</b> Információkeresés .....	957
Az IR-rendszerek értékelése .....	961
Az IR-rendszerek továbbfejlesztése .....	962
Az eredményhalmaz prezentálása .....	963
Az IR-rendszerek megvalósítása .....	964
<b>23.3.</b> Információkinyerés .....	966
<b>23.4.</b> Gépi fordítás .....	969
Gépi fordító rendszerek .....	971
Statisztikai gépi fordítás .....	972
Valószínűségek tanulása gépi fordításhoz .....	976
<b>23.5.</b> Összefoglalás .....	977
Irodalmi és történeti megjegyzések .....	978
Feladatok .....	981
<b>24.</b> <b>Az észlelés .....</b>	<b>984</b>
<b>24.1.</b> Bevezetés .....	984
<b>24.2.</b> Képalkotás .....	986
Lencsék nélküli képek: a sötétkamra .....	986
Lencserendszerek .....	987
A fény: a képalkotás fotometriája .....	988
Színek: a képalkotás spektrális fotometriája .....	990
<b>24.3.</b> Előzetes képfeldolgozási műveletek .....	990
Éldetektálás .....	991
A kép szegmentálása .....	994
<b>24.4.</b> 3D információ kinyerése .....	995
A mozgás .....	996
Kétkamerás (binokuláris) térbeli látás .....	999
Textúragradiensek .....	1001

Árnyalás .....	1002
Kontúrok .....	1003
<b>24.5. Objektumok felismerése .....</b>	<b>1007</b>
Fényességalapú felismerés .....	1009
Jellemzőalapú felismerés .....	1010
Az elhelyezkedés becslése .....	1013
<b>24.6. Navigálás és manipulálás a látás segítségével .....</b>	<b>1015</b>
<b>24.7. Összefoglalás .....</b>	<b>1016</b>
Irodalmi és történeti megjegyzések .....	1017
Feladatok .....	1020
<b>25. Robotika .....</b>	<b>1023</b>
<b>25.1. Bevezetés .....</b>	<b>1023</b>
<b>25.2. Robothardver .....</b>	<b>1025</b>
Érzékelők .....	1025
Beavatkozó szervek .....	1026
<b>25.3. Érzékelés a robotikában .....</b>	<b>1029</b>
Helymeghatározás .....	1030
Térképezés .....	1035
További érzékelési típusok .....	1038
<b>25.4. Mozgástervezés .....</b>	<b>1039</b>
Konfigurációs tér .....	1039
Celladekompozíciós módszerek .....	1042
Szkeletonizációs módszerek .....	1044
<b>25.5. Bizonytalan mozgások tervezése .....</b>	<b>1046</b>
Robusztus módszerek .....	1047
<b>25.6. Mozgás .....</b>	<b>1049</b>
Dinamika és vezérlés .....	1050
Potenciáltér-vezérlés .....	1052
Reaktív irányítás .....	1053
<b>25.7. Szoftverarchitektúrák a robotikában .....</b>	<b>1055</b>
Alárendelt architektúra .....	1055
Háromréteges architektúra .....	1057
Robotprogramozási nyelvek .....	1057
<b>25.8. Alkalmazási területek .....</b>	<b>1059</b>
<b>25.9. Összefoglalás .....</b>	<b>1061</b>
Irodalmi és történeti megjegyzések .....	1062
Feladatok .....	1066
<b>VIII. rész. Konklúziók .....</b>	<b>1071</b>
<b>26. Filozófiai alapok .....</b>	<b>1073</b>
<b>26.1. Gyenge MI: tudnak-e a gépek intelligensen cselekedni? .....</b>	<b>1073</b>
A képesség hiányából vett érv .....	1075
A matematikai ellenvetés .....	1076
A meghatározatlanságból vett érv .....	1077

26.2. Erős MI: tudnak-e ténylegesen gondolkodni a gépek? . . . . .	1079
A test–elme probléma . . . . .	1081
Az „agy a tartályban” kísérlet . . . . .	1082
Az agyprotézis-kísérlet . . . . .	1083
A kínai szoba . . . . .	1085
26.3. A mesterséges intelligencia fejlesztésének etikai kérdései és kockázatai . . . . .	1087
26.4. Összefoglalás . . . . .	1092
Irodalmi és történeti megjegyzések . . . . .	1092
Feladatok . . . . .	1095
<b>27. MI: jelen és jövő . . . . .</b>	<b>1096</b>
27.1. Ágensösszetevők . . . . .	1096
27.2. Ágensarchitektúrák . . . . .	1099
27.3. Egyáltalán a jó irányba haladunk? . . . . .	1100
27.4. Mi van, ha az MI sikerrel jár? . . . . .	1103
<b>A) Matematikai alapok . . . . .</b>	<b>1105</b>
A1. Bonyolultságanalízis és az O() jelölés . . . . .	1105
Aszimptotikus analízis . . . . .	1105
NP és az inherensen nehéz problémák . . . . .	1106
A2. Vektorok, mátrixok és lineáris algebra . . . . .	1108
A3. Valószínűségi eloszlások . . . . .	1109
Irodalmi és történeti megjegyzések . . . . .	1111
<b>B) Megjegyzések a nyelvekről és az algoritmusokról . . . . .</b>	<b>1112</b>
B1. Nyelvek definiálása Backus–Naur-formában (BNF) . . . . .	1112
B2. Az algoritmusok leírása pszeudokódval . . . . .	1113
B3. Online segédnyújtás . . . . .	1114
Irodalomjegyzék . . . . .	1115
Magyar nyelvű szakirodalom . . . . .	1169
Névmutató . . . . .	1173
Tárgymutató . . . . .	1187

# ELŐSZÓ A MAGYAR KIADÁSHOZ

Ön a „Mesterséges intelligencia: modern megközelítésben” könyv második kiadásának fordítását tárja a kezében. Az első kiadás óta sok év telt el, és közben a mesterséges intelligencia területe folyamatosan fejlődött tovább. Számos problémakör és módszer átértékelődött, új távlatok nyíltak, és különösen a nagy komplexitású problémák megoldásánál egyre nagyobb hangsúlyt kaptak az olyan megoldási megközelítések, amelyek különböző ismeretek együttes felhasználásán alapulnak. A szerzők úgy gondolták, hogy e kihívásoknak csak úgy tudnak megfelelni, ha a könyvnek egy új, jelentősen átdolgozott változatát készítik el.

Az első kiadás szakmai színvonalára jellemző, hogy az átdolgozás és a bővítés nem érintette a könyv lényegi szakmai vonulatát. A könyvben ezúttal több helyet kaptak az utóbbi években intenzívebben művelt területek. A könyv több témaörrel bővült, illetve több témaör – amely az első kiadásban csak érintőlegesen szerepelt – az új kiadásban nagyobb teret kapott. Így önálló fejezet foglakozik a kényszerkielégítési problémákkal, teljesen új arculatot kaptak a tervkészítés fejezetei is, de szinte minden, az első kiadásban is érintett terület bővebben, a legújabb eredményeket is bemutatva jelent meg. Hogy a szerzők célkitűzése minél hitelesebb legyen, átdolgozták az összes fejezet irodalmi összefoglalóját és gyakorló feladatait. Az irodalmi összefoglalókban igyekeztek a fejezetek mondanivalóját a legfrissebb kutatási eredményekhez kapcsolva bemutatni.

A második kiadás magyar fordítását megelőzte a könyv olasz nyelvre való lefordítása. A könyv így már öt (világ)nyelven terjeszti az informatika egyik legizgalmasabb területének vívmányait. Folyamatosan bővült az olvasók köre. Új egyetemek új kurzusokban fogadták el a könyvet mint meghatározó, alapvető irodalmat. Már Magyarországon is az informatikus egyetemi hallgatók több „generációja” használta tankönyvként. Reméljük, hogy az új kiadás magyar változata az ő tanulmányi munkájukat is jobban fogja segíteni.

A második kiadás egyáltalán nem jelentett könnyebb fordítási munkát. A szerzők az anyag bővítését végigkondoltan tették meg. A könnyebb olvashatóság és a logikusabb gondolatvezetés érdekében átdolgozták azon fejezetek anyagát is, amelyek szakmai szempontból az első kiadással majdnem teljesen azonosak. Az új kutatási eredményekkel együtt így ismét olyan új fogalmak jelentek meg, amelyeknek elfogadott magyar szóhasználata még nem létezett. Nyelvújító jelleggel megkíséreltük ilyenkor a lényeget legjobban megragadó anyanyelvi megfelelőt megtalálni. A második kiadás fordításában erőteljesen támaszkodtunk az első fordítás tapasztalataira, így ezúttal is köszönnett

tartozunk mindeneknek, akik akkor az anyanyelvi szókészlet kialakításában segítségre voltak. A második kiadáshoz is mellékelünk egy magyar nyelvű irodalomjegyzéket, hogy az angol nyelvben kevésbé jártas olvasóknak is segítséget adjunk egyes részterületek bővebb tanulmányozásához. Az irodalomjegyzék kialakításához Sántáné Tóth Edit segítségét kértük, akinek széles körű ismeretei és személyes vázára elengedhetetlen volt, hogy a magyar „MI társadalmat” feltérképezze, és akinek a sok hónapos intenzív munkájáért ezúttal is köszönetet mondunk.

Budapest, 2005. május

*A fordítók és a lektor*

# ELŐSZÓ

A mesterséges intelligencia (MI) nagyon nagy terület, és ez a könyv is igen terjedelmes. A könyvben megkíséreltük e területet teljes szélességében feltární: átöleve a logikát, a valószínűség-számítást, a folytonos matematikát, az érzékelést, a következtetést, a tanulást, a cselekvést, valamint minden, a mikroelektronikától a világúrt kutató robotokig. A könyv azért is terjedelmes, mert bizonyos mértékig el is mélyedünk az eredmények tárgyalásában, miközben az egyes fejezetek központi részében csak a leglényegesebb gondolatokat igyekszünk bemutatni. További eredményekre az egyes fejezetek végén található irodalmi feljegyzésekben mutatunk rá.

A könyv alcíme „modern megközelítésben”. E talán önmagában üres szófordulat szándékolt jelentése az, hogy az ismert eredményeket egy közös keretbe szintetizálva igyekeztük bemutatni, ahelyett hogy az MI egyes részterületeit külön-külön a saját történelmi háttérükben tárgyaljuk. Elnézést is kérünk mindenektől, akik a saját szakterületüket így a szokásosnál kevésbé ismerik fel.

A könyv fő egységesítő motívuma az **intelligens ágens** koncepciója. Felfogásunkban az MI egyszerűen a környezetüket észlelő és cselekvő ágensek tanulmányozása. minden ilyen ágens egy érzékeléseket cselekvésekre leképező függvényt valósít meg. A könyvben e függvénykapcsolat többféle reprezentációjával foglalkozunk, olyan reprezentációkkal, mint például a produkciós rendszerek, a reaktív ágensek, a valós idejű feltételes tervkészítő rendszerek, a neurális hálók és a döntéselméleti rendszerek. A továbbiakban bemutatjuk a tanulás szerepét, mint a rendszertervezőnek azt a módszerét, amivel ismeretlen környezeteket is képes figyelembe venni, és megmutatjuk, hogy a tanulás milyen korlátozásokat jelent az ágens tervezésében, kiemelve az explicit tudásreprezentáció és a következtetés szerepét. A robotikát és a gépi látást nem függetlenül definiált problémaként tárgyaljuk, hanem mint a cél eléréséhez szükséges szolgáltatásokat. Külön hangsúlyozzuk a feladatkörnyezet fontosságát az ágens megfelelő megtervezése szempontjából.

A fő célunk azokat a *gondolatokat* átadni, amelyek az MI-kutatás elmúlt 50 évben és a kapcsolódó területeken az elmúlt kétezer évben születtek. E gondolatok bemutatásánál, miközben a precizitás megtartására törekedtünk, igyekeztünk kerülni a túlzott formalizmust. Ahol alkalmasnak találtuk, pszeudokód szintű algoritmusokat is közöltünk, hogy a gondolatok konkrét alakot öltsenek. Az általunk használt pszeudokódöt röviden a B) függelékben írjuk le. A különböző programozási nyelveken implementált algoritmusokat az olvasó a <http://aima.cs.berkeley.edu> weboldalon találja meg.

A könyvet alapvetően egy alapképzésbeli tárgyhoz vagy tárgysorozathoz szántuk. Ugyanakkor használható a graduális képzés tárgyaiban is (esetleg az irodalmi megjegyzésekben javasolt források némelyikének hozzáadásával). A széles témafeladás és a nagyszámú részletezett algoritmus miatt a könyv alapreferenciaként szolgálhat az MI-szakos végzős hallgatók és a szűkebb kutatási területükön kitörni kívánó szakemberek számára. Egyetlen feltétel a számítógépes tudományok alapvető fogalom-szintű ismerete (algoritmusok, adatstruktúrák, komplexitás) egy első-másodéves hallgató szintjén. Az analízis alsóbb éves szintű ismerete jól jön a neurális hálók és a statisztikai tanulás részleteinek megértéséhez. A szükséges matematikai háttér egyes részeit az A) függelékben adjuk meg.

## A KÖNYV ÁTTEKINTÉSE

A könyv nyolc részből áll. Az I. rész, a **Mesterséges intelligencia** az MI-kutatások olyan megközelítését adja, amely az intelligens ágens elkezelés köré szerveződik. Egy olyan rendszer köré, amely képes eldönteni, hogy mit cselekedjen és amely cselekedni is képes. A II. rész, a **Problémamegoldás** olyan döntési módszerekkel foglalkozik, amikor néhány lépésre előre kell gondolkodni. Ilyen pl. a sakk vagy a nyílt terepen történő navigálás. A III. rész, a **Tudás és következtetés** a vilagról alkotott tudás reprezentálásával foglalkozik, valamint azzal, hogy hogyan működik egy ilyen reprezentáció, milyen reprezentációkat ismerünk, hogyan befolyásolhatjuk ezeket a cselekvésünkkel, és hogy ezzel a tudással hogyan lehet logikusan következtetni. Ennek folytatásaként a IV. rész, a **Tervkészítés** e következtetési módszerek használatát mutatja a cselekvés eldöntésében, pontosabban a *tervek* készítésében. Az V. rész, a **Bizonytalan tudás és következtetés** hasonlít a III. és IV. részekre, de a *bizonytalanságok* mellett következtetésre és döntéshozatalra koncentrál. Ilyen problémákra pl. az orvosi diagnosztizáló és kezelő rendszerekben kell számítani.

A II.-tól az V. részig az anyag az intelligens ágenseknek azzal a komponensével foglalkozik, amely a döntéshozatalért felelős. A VI. rész, a **Tanulás** a döntéshozó komponens által igényelt tudás megszerzési módszereit írja le. A VII. rész, a **Kommunikáció, észlelés és cselekvés** olyan módszereket tár fel, amelyekkel az intelligens ágens a környezetét érzékelheti – látás, tapintás, hallás vagy akár nyelvmegértés révén – abból a célból, hogy megtudja, mi történik körülötte. Továbbá olyan módszerekkel foglalkozik, melyek képesek az ágens tervezet igazi cselekvéssé alakítani, ahol a cselekvés lehet egy robotmozgás, de akár nyelvi közlés is. Végül a VIII. rész, a **Konklúziók** az MI múltját és jövőjét, valamint a filozófiai és etikai következményeit elemzi.

## VÁLTOZÁSOK AZ ELSŐ KIADÁSHOZ KÉPEST

Az első kiadás óta az MI sok változáson ment át, és ez a könyv is sokat változott. Lényegében át is írtunk minden fejezetet abból a célból, hogy a könyv tükrözze a kutatás legújabb eredményeit, a korábbi munkáknak az új eredményekhez jobban illeszkedő értelmezését adhassa, valamint hogy a gondolatok folyamának oktató jellege még jobban érvényesülni tudjon. Az MI követőit bátorítani szeretnénk, hogy a mai módszerek

sokkal praktikusabbak, mint az 1995-ösek. A tervkészítést példának véve az első kiadás algoritmusai néhány tíz lépésből álló tervek generálására voltak képesek, e kiadás algoritmusai több tízezer lépésgyű tervekre skálázhatók fel. Hasonló nagyságrendű javulás tapasztalható a valószínűségi következetés, a nyelvfeldolgozás és más területek esetében is. A könyvben az alábbiak a legfontosabb változások:

- Az I. részben méltatjuk a szabályozáselmélet, a játékelmélet, a gazdaságtan és a neurális tudományok történelmi szerepét. Ez segíti, hogy e gondolatokat a következő fejezetekben integráltabb módon tudjuk tárgyalni.
- A II. részben az online keresési algoritmusokkal foglalkozunk, és egy új, a korlátosztáskielégítésről szóló fejezetet is beiktattunk. Ez utóbbi természetes átmenetet képez a logikával foglalkozó anyagrészek felé.
- A III. részben az ítéletkalkulus, amit az első kiadásban az elsőrendű logika felé vezető lépcsőnek tekintettünk, itt mint egy önálló hasznos reprezentációs nyelv kerül bemutatásra, gyors következetési eljárásokkal és áramköralapú ágensfejlesztéssel. Az elsőrendű logikával foglalkozó fejezetet átszerveztük, hogy az anyag áttekinthetőbb legyen és alkalmazási példaként az internetes vásárlás tárgyterülettel bővítettük.
- A IV. részben foglalkozunk olyan újabb módszerekkel, mint a GRAPHPLAN és a kielégítésalapú tervkészítés. Bővült az ütemezés, a feltételes tervkészítés, a hierarchikus tervkészítés és a többágenses tervkészítés bemutatása.
- Az V. részben a valószínűségi hálók anyagát új algoritmusokkal bővítettük, mint például a változóeliminálás és Markov lánc Monte Carlo algoritmusok. Egy, a bizonytalan temporális következetésekkel foglalkozó új fejezetet írtunk, ahol rejttett Markov-modellekről, Kalman-szűrőkről és dinamikus valószínűségi hálókról van szó. Mélyebb a Markov döntési folyamat tárgyalása, új alfejezetek foglalkoznak a játékelméettel és a mechanizmustervezéssel.
- A VI. részben kapcsoljuk össze a statisztikai, a szimbolikus és a neurális tanulást továbbá a boosting algoritmusokkal, az EM algoritmussal, a példányalapú tanulással és a kernelmódszerekkel (support vector machines, szupport vektor gépek) foglalkozó részekkel bővítjük anyagot.
- A VII. részben a nyelvfeldolgozás tárgyalását a párbeszédmegértéssel és a nyelvtan-indukálással bővítjük. Új rész foglalkozik a nyelv valószínűségi modellezésével, az információ-visszakeresési és fordítási alkalmazásokkal. A robotika tárgyalásánál kiemeljük a bizonytalan szenzoradatok egyesítését. A látással foglalkozó anyagrész objektumfelismeréssel bővült.
- A VIII. részben egy új alfejezetben az MI etikai következményeivel foglalkozunk.

## A KÖNYV JAVASOLT HASZNÁLATA

A könyvnek 27 fejezete van, mindegyik nagyjából egyhetes előadásnyi anyagot jelent. Az egész könyv feldolgozása két félévet tenne így ki. A tanfolyamot azonban a szükebb hallgatói és előadói érdeklődéshez is szabhatjuk. Mivel a könyv széles területet fed le, jó háttéranyag lehet egy rövid bevezető alapkonzuszhöz, de felhasználható egyes tématörököt magasabb szinten bemutató specializált graduális kurzusokhoz is. A könyv első kiadását felhasználó 600 egyetemi kurzus tematikából egy ízelítő az

*aima.cs.berkeley.edu* weblapon található. Itt néhány hasznos tanács is található arra nézve, hogy konkrét speciális igényekhez hogyan állíthatunk össze egy megfelelő téma-szekvenciát.

 A könyv 385 gyakorlatot tartalmaz. Azokat, amelyekhez nagyobb lélegzetű programozási munka is szükséges, egy klaviatúra ikon jelzi. Az ilyen gyakorlatokat legjobban úgy lehet megoldani, hogy igénybe vessük az *aima.cs.berkeley.edu* programkódját szolgáltatásait. Méreténél fogva néhány gyakorlat féléves házi feladatként is kiadható. A gyakorlatok egy része irodalmazást is igényel. Ezeket egy nyitott könyv ikon jelzi.

 A könyvben a fontos részeket egy mutató kéz ikon jelzi. Hogy a könyvben az egyes téma-kat könnyű legyen megtalálni, kiterjedt tárgymutatót is készítettünk.

## A WEBOLDAL HASZNÁLATÁRÓL

Az *aima.cs.berkeley.edu* weboldalon megtalálhatók:

- a könyvbeli algoritmusok implementációja több különböző programozási nyelven;
- a könyvet használó több mint 600 oktatási intézmény listája, számos webes linkkel az online oktatási anyagokhoz;
- annotált linkek listája több mint 800 olyan weboldalhoz, ahol hasznos MI-tartalom található;
- a segédanyagok és linkek fejezetenkénti listája;
- útmutató, hogy hogyan csatlakozhatunk a könyvvel foglalkozó vitafórumhoz;
- útmutató, hogy a kérdéseinkkel és az észrevételeinkkel hogyan léphetünk kapcsolatba a szerzőkkel;
- útmutató, hogy a nagy valószínűséggel előforduló hibákat hogyan jelezük;
- a könyvbeli ábrák, fóliák és más oktatáshoz hasznos anyagok.

## KÖSZÖNETNYILVÁNÍTÁS

A 24. fejezet (Az észlelés) nagy részét Jitendra Malik, a 25. fejezet (Robotika) többségét az első kiadásban John Canny, a jelen kiadásban pedig Sebastian Thrun írta. Az Irodalmi és történeti megjegyzéseket az első kiadáshoz Doug Edwards kutatta fel. Tim Huang, Mark Paskin és Cynthia Bruyns segítettek az ábrák és az algoritmusok megfelelő formájú kialakításában. A Prentice Hall kiadónál Alan Apt, Sondra Chavez, Toni Holm, Jake Warde, Irwin Zucker és Camille Trentacoste minden tőlük telhetőt megették, hogy rákénszerítsenek bennünket a határidők betartására, és sok hasznos javaslattal segítették, hogy a könyv a megfelelő tartalommal és megfelelő köntösben jelenjen meg.

Stuart meg szeretné köszönni szüleinek a folyamatos támogatást és bátorítást, feleségenek, Loy Shefflottnak a végletes türelmét és a határtalan bölcsességét. Reméli, hogy Gordon és Lucy rövidesen már olvashatják is ezt. Igen segítőképesnek bizonyult az RKHC (Russell Kivételes Hallgatói Csoportja) is.

Peter szeretné megköszönni szüleinek (Torsten és Gerda) az indítatást. Feleségeinek (Kris), gyerekeinek és barátainak köszöni a bátorítást és azt, hogy elviselték őt a könyv megírásának hosszú ideje és a könyv átírásának még hosszabb ideje alatt.

Adósak vagyunk a Berkeley, a Stanford, az MIT és a NASA könyvtárosainak és a CiteSeer, továbbá a Google fejlesztőinek, hogy a kutatás módját forradalmasították.

Nem tudunk külön-külön köszönetet mondani mindenkinék, aki a könyvet használta, és hasznos megjegyzésekkel állt elő, azonban az alább felsoroltak különösen hasznos megjegyzéseit meg szeretnénk köszönni: Eyal Amir, Krzysztof Apt, Ellery Aziel, Jeff Van Baalen, Brian Baker, Don Barker, Tony Barrett, James Newton Bass, Don Beal, Howard Beck, Wolfgang Bibel, John Binder, Larry Bookman, David R. Boxall, Gerhard Brewka, Selmer Bringsjord, Carla Brodley, Chris Brown, Wilhelm Burger, Lauren Burka, Joao Cachopo, Murray Campbell, Norman Carver, Emmanuel Castro, Anil Chakravarthy, Dan Chisarick, Roberto Cipolla, David Cohen, James Coleman, Julie Ann Comparini, Gary Cottrell, Ernest Davis, Rina Dechter, Tom Dietterich, Chuck Dyer, Barbara Engelhardt, Doug Edwards, Kutluhan Erol, Oren Etzioni, Hana Filip, Douglas Fisher, Jeffrey Forbes, Ken Ford, John Fosler, Alex Franz, Bob Futrelle, Marek Galecki, Stefan Gerberding, Stuart Gill, Sabine Glesner, Seth Golub, Gosta Grahne, Russ Greiner, Eric Grimson, Barbara Grosz, Larry Hall, Steve Hanks, Othar Hansson, Ernst Heinz, Jim Hendler, Christoph Herrmann, Vasant Honavar, Tim Huang, Seth Hutchinson, Joost Jacob, Magnus Johansson, Dan Jurafsky, Leslie Kaelbling, Keiji Kanazawa, Surekha Kasibhatla, Simon Kasif, Henry Kautz, Gernot Kerschbaumer, Richard Kirby, Kevin Knight, Sven Koenig, Daphne Koller, Rich Korf, James Kurien, John Lafferty, Gus Larsson, John Lazzaro, Jon LeBlanc, Jason Leatherman, Frank Lee, Edward Lim, Pierre Louveaux, Don Loveland, Sridhar Mahadevan, Jim Martin, Andy Mayer, David McGrane, Jay Mendelsohn, Brian Milch, Steve Minton, Vibhu Mittal, Leora Morgenstern, Stephen Muggleton, Kevin Murphy, Ron Musick, Sung Myaeng, Lee Naish, Pandu Nayak, Bernhard Nebel, Stuart Nelson, XuangLong Nguyen, Illah Nourbakhsh, Steve Omohundro, David Page, David Palmer, David Parkes, Ron Parr, Mark Paskin, Tony Passera, Michael Pazzani, Wim Pijs, Ira Pohl, Martha Pollack, David Poole, Bruce Porter, Malcolm Pradhan, Bill Pringle, Lorraine Prior, Greg Provan, William Rapaport, Philip Resnik, Francesca Rossi, Jonathan Schaeffer, Richard Scherl, Lars Schuster, Soheil Shams, Stuart Shapiro, Jude Shavlik, Satinder Singh, Daniel Sleator, David Smith, Brian So, Robert Sproull, Lynn Stein, Larry Stephens, Andrea Stolcke, Paul Stradling, Devika Subramanian, Rich Sutton, Jonathan Tash, Austin Tate, Michael Thielscher, William Thompson, Sebastian Thrun, Eric Tiedemann, Mark Torrance, Randall Upham, Paul Utgoff, Peter van Beek, Hal Varian, Sunil Vemuri, Jim Waldo, Bonnie Webber, Dan Weld, Michael Wellman, Michael Dean White, Kamin Whitehouse, Brian Williams, David Wolfe, Bill Woods, Alden Wright, Richard Yen, Weixiong Zhang, Shlomo Zilberstein és a Prentice Hall névtelen bírálói.

## A KÖNYV BORÍTÓJÁRÓL

A borítón látható képet a szerzők szerkesztették és Lisa Marie Sardegna és Maryann Simmons kivitelezte SGI Inventor™ és az Adobe Photoshop™ segítségével. A címlap az MI történetéből az alábbiakat eleveníti fel:

1. Arisztotelész tervkészítő algoritmusa a *De Motu Animalium* c. műből (kb. i.e. 400).
2. Ramon Lull fogalomgenerátora az *Ars Magna* c. műből (1300 körül).

3. Charles Babbage Differenciál Gépe, az első univerzális számítógép prototípusa (1848).
4. Az elsőrendű logika Gottlob Frege-féle formalizmusa (1789).
5. A logikai következtetés Lewis Carroll-féle diagramjai.
6. A valószínűségi hálózat Sewall Wright-féle ábrázolása (1921).
7. Alan Turing (1912–1954).
8. Shakey a Robot (1969–1973).
9. Egy korszerű diagnosztikai szakértői rendszer (1993).

## A SZERZŐKRŐL

**Stuart Russell** 1962-ben Portsmouth-ban, Angliában született. 1982-ben Oxfordban fizikából szerzett kiüntetéses oklevelet, majd 1986-ban a Stanfordon doktorált számítógépes tudományokból. Ettől kezdve a Kaliforniai Berkeley Egyetemen (University of California at Berkeley) dolgozik, ahol jelenleg a számítógépes tudományok professzora, az Intelligens Rendszerek Központ igazgatója és a Smith-Zadeh mérnöktudományi tanszék vezetője. 1990-ben megkapta a Nemzeti Kutatási Alap Fiatal Kutatók Elnöki Díját, 1995-ben pedig másodmagával a Számítógép és a Gondolat Díjat. 1996-ban a Miller Intézet ösztöndíjas professzora volt a Kaliforniai Egyetemen és 2000-ben megkapta a tekintélyes hároméves, rektori kutatói-professzori ösztöndíjat. 1998-ban a Stanfordon Forsythe-emlékelőadásokat tartott. Az Amerikai MI Társaság tagja, korábban pedig a társaság vezetőségi tagja volt. Több mint 100 cikket publikált az MI különböző területeiről. További könyvei között megtaláljuk a *The Use of Knowledge in Analogy and Induction* és (Eric Wefalddal együtt) a *Do the Right Thing: Studies in Limited Rationality* c. könyveket.

**Peter Norvig** a Search Quality részleg vezetője a Google-nál. Az Amerikai MI Társaság vezetőségi tagja. Korábban a NASA Ames Research Centerben a Számítástudományi Osztály igazgatója volt, ahol felügyelte a NASA MI- és robotikai kutatásait és fejlesztéseit. Ezt megelőzően a Junglee-nél dolgozott tudományos főmunkatársként, ahol hozzájárult az első internetes információkinyerő szolgáltatás kifejlesztéséhez, valamint a Sun Microsystems Laboratoriesnál vezető kutatóként, ahol az intelligens információ-visszaszerzésen dolgozott. Diplomáját alkalmazott matematikában a Brown Egyetemen, PhD-fokozatát, számítógépes tudományokban, a Kaliforniai Egyetemen Berkeleyben kapta. A Dél-kaliforniai Egyetemen professzorként, a Berkeleyben, Kaliforniában pedig a tanszéki kutatócsoport tagjaként dolgozott. Több mint 50 publikációja van a számítógépes tudományok területén. Ezek között találjuk a *Paradigms of AI Programming: Case Studies in Common Lisp*, *Verbmobil: A Translation System for Face-to-Face Dialog* és *Intelligent Help System for UNIX* c. könyveket.

# NÉVMUTATÓ

## A

Aarup, M. 535  
Abbott, L. F. 1111  
Abelson, R. P. 56, 945  
Abramson, B. 129  
Acero, A. 673  
Acharya, A. 175  
Ackley, D. H. 177  
Adams, J. 409  
Adelson-Velsky, G. M. 237  
Adorf, H.-M. 536  
Agmon, S. 866  
Agre, P. E. 538  
Aho, A. V. 945, 1111  
Ait-Kaci, H. 380  
Aizerman, M. 868  
Albus, J. S. 897  
Aldiss, B. 1091  
Aldous, D. 176  
al-Hvarizmi 39  
Allais, M. 685  
Allen, B. 488, 535  
Allen, J. F. 437, 490, 947  
Allen, N. 45, 49, 59, 290  
Allen, W. 666  
Almuallim, H. 815  
Alperin Resnick, L. 423, 439  
Alshawi, H. 944, 945  
Alterman, R. 536  
Amarel, S. 128, 131, 434  
Ambros-Ingerson, J. 536, 538  
Amit, D. 866  
Anbulagan 292  
Andersen, S. K. 618  
Anderson, A. R. 1093, 1094  
Anderson, J. A. 63, 866  
Anderson, J. R. 45, 349, 620, 814  
Andre, D. 178, 899, 1058

## B

Anshelevich, V. A. 239  
Appel, K. 203  
Appelt, D. 945, 980  
Apt, K. R. 204  
Apté, C. 979  
Arentoft, M. M. 535  
Ares, M. 866  
Arisztotelész. 27, 36, 37, 38, 90,  
376, 435, 438, 1092  
Arkin, R. 1065  
Arlazarov, V. L. 237  
Armstrong, D. M. 1093  
Arnould, A. 38, 676, 701  
Arora, S. 129  
Asada, M. 1064  
Ashby, W. R. 865  
Asimov, I. 1063, 1091  
Astrom, K. J. 741  
Atanasoff, J. 46  
Atkeson, C. G. 897  
Atkin, L. R. 129  
Attenborough, R. 704  
Audi, R. 1094  
Austin, G. A. 777  
Austin, J. L. 943  
Axelrod, R. 742

Babbage, Ch. 28, 46, 236, 1090  
Bacchus, F. 189, 204, 206, 573,  
620, 702  
Bachmann, P. G. H. 1111  
Backus, J. W. 944  
Bacon, F. 37  
Baeza-Yates, R. 979  
Baird, L. C. I. 740  
Bajcsy, R. 1019  
Baker, C. L. 63

Baker, J. 672, 978  
Balashek, S. 672  
Baldwin, J. M. 164  
Ball, M. 490  
Ballard, B. W. 239, 242  
Baluja, S. 177  
Bancilhon, F. 379  
Banerji, R. 791  
Bar-Hillel, Y. 945, 980  
Barrett, A. 536  
Barry, M. 622  
Bar-Shalom, Y. 671  
Bartak, R. 206  
Bartlett, F. 44  
Bartlett, P. 779, 867, 898  
Barto, A. G. 178, 741, 897, 899  
Barton, G. E. 949  
Barwise, J. 295  
Basye, K. 535, 1064  
Bateman, J. A. 945  
Bates, E. 947  
Baum, E. 161, 239, 864  
Baum, L. E. 671, 867  
Baxter, J. 898  
Bayardo, R. J. 292  
Bayerl, S. 381  
Bayes, T. 40, 572  
Beal, D. F. 238  
Bear, J. 980  
Beckert, B. 381  
Beeri, C. 205  
Bell, C. 508, 535  
Bell, J. L. 328  
Bell, T. C. 978  
Bellman, R. E. 32, 41, 128, 129,  
174, 714, 740  
Belongie, S. 867, 1020  
Bender, E. A. 130  
Bendix, P. B. 380

- Bennett, B. 374, 440  
 Bennett, F. H. 178  
 Bentham, J. 701  
 Berger, H. 43  
 Berlekamp, E. R. 130  
 Berleur, J. 1088  
 Berliner, H. J. 237, 238, 239  
 Bernardo, J. M. 828  
 Berners-Lee, T. 435  
 Bernoulli, D. 682, 701  
 Bernoulli, J. 40, 572, 705  
 Bernstein, A. 236  
 Bernstein, P. L. 574, 745  
 Berrou, C. 620  
 Berry, C. 46  
 Berry, D. A. 897  
 Bertele, V. 619  
 Bertoli, P. 537  
 Bertsekas, D. 1111, 740, 899  
 Bezzel, M. 129  
 Bibel, W. 378, 381  
 Bickford, M. 375  
 Biddulph, R. 672  
 Bidlack, C. 1064  
 Bigelow, J. 47  
 Biggs, N. L. 203  
 Bilmes, J. 673  
 Binder, J. 671  
 Binmore, K. 742  
 Birnbaum, L. 980  
 Biro, J. I. 1094  
 Birtwistle, G. 438  
 Bishop, C. M. 90, 177, 619  
 Bishop, R. H. 90  
 Bistarelli, S. 204  
 Bitman, A. R. 237  
 Bitner, J. R. 204  
 Blake, A. 659, 672  
 Blei, D. M. 978  
 Blinder, A. S. 745  
 Block, N. 1093, 1094  
 Blum, A. L. 489  
 Blumer, A. 778  
 Bobick, A. 671  
 Bohrow, D. 52, 814, 980  
 Boddy, M. 537, 1100  
 Boden, M. A. 1094  
 Boneh, D. 161  
 Bonet, B. 489, 537  
 Boole, G. 38, 39  
 Boolos, G. S. 381  
 Booth, A. D. 978  
 Booth, T. L. 980  
 Borel, E. 742  
 Borenstein, J. 1064, 1065  
 Borgida, A. 423, 439  
 Boser, B. 867  
 Bottou, L. 867, 1020  
 Boutilier, C. 436, 539, 741  
 Bower, G. H. 897  
 Box, G. E. P. 177  
 Boyan, J. A. 176, 897  
 Boyen, X. 672  
 Boyer, R. S. 374, 375, 381  
 Brachman, R. 423, 438, 439, 441  
 Bradtke, S. J. 897  
 Brafman, R. I. 539, 897  
 Brahmagupta 203  
 Braitenberg, V. 1065  
 Brandeis, L. 1089  
 Bransford, J. 950  
 Bratko, I. 175, 380, 809  
 Bratman, M. 91, 1094  
 Braverman, E. 868  
 Breese, J. S. 618, 620, 622, 703, 7100  
 Breiman, L. 777  
 Brelaz, D. 204  
 Brent, R. P. 176  
 Bresnan, J. 944  
 Brewka, G. 439  
 Briggs, R. 434  
 Brin, S. 979  
 Brioschi, F. 619  
 Broadbent, D. E. 45  
 Broca, P. 42  
 Brooks, M. J. 91, 538, 1055, 1064  
 Brooks, R. A. 1065  
 Brown, A. 488  
 Brown, J. S. 440, 816  
 Brown, M. 866  
 Brown, P. F. 974, 980  
 Brownston, L. 378  
 Brudno, A. L. 237  
 Brunnstein, K. 1088  
 Brunot, A. 867, 1020  
 Bryant, R. E. 518, 537  
 Bryson, A. E. 54  
 Buchanan, B. G. 55, 56, 91, 621, 790  
 Bulfin, R. 742  
 Bundy, A. 382, 814  
 Bunt, H. 437  
 Buntine, W. 815  
 Burgard, W. 1064  
 Buro, M. 232  
 Burstall, R. M. 381, 436  
 Burstein, J. 1075  
 Bylander, T. 464, 487  
 Byron, Lord 46
- C**
- Califf, M. E. 947  
 Calvanese, D. 439  
 Cambefort, Y. 91  
 Cameron-Jones, R. M. 808  
 Campbell, M. S. 230  
 Canny, J. 1064, 1065  
 Cantu-Paz, E. 177  
 Čapek, J. 1062  
 Čapek, K. 1062, 1090  
 Carlin, J. B. 868  
 Carbonell, J. G. 536, 778, 814  
 Carbonell, J. R. 814  
 Cardano, G. 40, 572  
 Carnap, R. 37, 556, 559, 573, 620  
 Carroll, L. 28  
 Carrasco, R. C. 947  
 Cassandra, A. R. 741  
 Ceri, S. 379  
 Chakrabarti, P. P. 175  
 Chambers, R. A. 891, 897  
 Chan, W. P. 1018  
 Chandra, A. K. 379  
 Chang, C.-L. 382  
 Chang, K. C. 619  
 Chapman, D. 488, 538  
 Charniak, E. 32, 56, 378, 978, 945, 946, 981  
 Chatfield, C. 671  
 Chatila, R. 1063  
 Cheeseman, P. 40, 59, 205, 621, 1063  
 Chekaluk, R. 535, 1064  
 Chen, R. 232, 672  
 Cheng, J. 619  
 Chervonenkis, A. J. 868  
 Chickering, D. M. 238  
 Chierchia, G. 63, 944  
 Chomsky, C. 978  
 Chomsky, N. 45, 47, 944, 946  
 Choset, H. 1064  
 Chung, K. L. 574, 1111  
 Church, A. 336, 377  
 Church, K. 945, 978  
 Churchland, P. M. 1093  
 Churchland, P. S. 1085, 1093, 1094  
 Cimatti, A. 490

- Clamp, S. E. 573  
 Clark, K. L. 425, 439  
 Clark, P. 815  
 Clark, S. 1063  
 Clarke, E. 381, 489  
 Clarke, A. C. 618, 1088  
 Clarke, M. R. B. 239  
 Clearwater, S. H. 742  
 Clinton, W. J. 1007  
 Clocksin, W. F. 380  
 Clowes, M. B. 1004  
 Cobham, A. 40  
 Cobley, P. 943  
 Cohen, W. W. 815  
 Cohen, B. 292  
 Cohen, C. 1064  
 Cohen, J. 379  
 Cohen, P. R. 58, 62, 539, 944  
 Cohn, A. G. 440  
 Collins, G. 537  
 Collins, A. M. 814  
 Collins, M. 866  
 Collins, M. J. 978  
 Colmerauer, A. 379, 944  
 Colombano, S. P. 177  
 Condon, J. H. 237  
 Congdon, C. B. 1064  
 Connell, J. 1065  
 Console, L. 91  
 Conway, J. H. 130  
 Cook, P. J. 1088, 1089  
 Cook, S. A. 40, 291, 292, 1111  
 Cooper, G. 619  
 Copeland, J. 437, 1094  
 Cormen, T. H. 1111  
 Cortes, C. 867, 1020  
 Cournot, A. 742  
 Covington, M. A. 947  
 Cowan, J. D. 53, 865  
 Cox, I. 671, 1063  
 Cox, R. T. 559, 573  
 Craig, J. 1065  
 Craik, K. J. 44, 45  
 Creimers, A. B. 1064  
 Crick, F. H. C. 163  
 Cristianini, N. 866, 868  
 Crocker, S. D. 237  
 Crockett, L. 436  
 Croft, W. B. 979  
 Cruse, D. A. 962  
 Culberson, J. 176  
 Cullingford, R. E. 56, 945  
 Currie, K. W. 535
- Cussens, J. 947  
 Cybenko, G.  
 Csorba, M. 1063
- D**
- D'Ambrosio, B. 618  
 da Vinci, L. 36  
 Daganzo, C. 619  
 Dagum, P. 619  
 Dahl, O.-J. 438  
 Dale, R. 945, 947  
 Damerau, F. 979  
 Daniels, C. J. 179  
 Dantzig, G. B. 177  
 Darabos, T. 1095  
 Darlington, J. 381  
 Darwiche, A. 619, 622  
 Darwin, Ch. 163, 1089  
 Dasgupta, P. 178  
 Daun, B. 536  
 Davidson, D. 437, 440  
 Davies, T. R. 799, 814  
 Davis, E. 435, 438, 440, 441,  
 Davis, G. 536  
 Davis, K. H. 672  
 Davis, M. 276, 291, 365, 377  
 Davis, R. 816  
 Dawid, S. 618  
 Dayan, P. 898  
 Deacon, T. W. 58  
 Deale, M. 536  
 Dean, M. E. 375  
 Dean, T. 535, 671, 741, 1064,  
 1065, 1099  
 Debevec, P. 1019  
 Debreu, G. 689  
 de Bruin, A. 238  
 Dechter, R. 174, 204, 205, 206, 619  
 DeCoste, D. 866, 867  
 de Dombal, F. T. 573  
 Deerwester, S. C. 978  
 de Finetti, B. 558, 573  
 de Freitas, J. F. G. 659  
 DeGroot, M. H. 574  
 DeJong, G. 814, 980  
 de Kleer, J. 205, 378, 440  
 Del Favero, B. A. 618  
 Della Pietra, S. A. 974, 980  
 Della Pietra, V. J. 974, 980  
 de Marcken, C. 129, 947  
 Dempster, A. P. 621, 671, 864  
 De Morgan, A. 203
- Denes, P. 672  
 Deng, X. 178  
 Denis, F. 947  
 Denker, J. 867, 1020  
 Dennett, D. C. 1078, 1080, 1094  
 Deo, N. 129  
 De Raedt, L. 815, 947  
 DeSarkar, S. C. 175, 178  
 de Saussure, F. 943  
 Descartes, R. 37, 1081, 1092  
 Descotte, Y. 535  
 Deville, Y. 204  
 Devol, G. 1063  
 Devroye, L. 864, 865  
 Dickinson, M. H. 1018  
 Dickmanns, E. D. 1020  
 Dietterich, T. 778, 814, 815,  
 899  
 DiGioia, A. M. 61  
 Dijkstra, E. W. 129, 174, 1093  
 Dill, D. L. 375  
 Dimopoulos, Y. 489  
 Diofantosz, 203  
 Diorio, C. 673  
 Dissanayake, G. 1063  
 Dix, J. 439  
 Dixon, J. K. 237  
 Dizdarevic, S. 177, 181  
 Do, M. B. 535  
 Domingos, P. 574  
 Donskoy, M. V. 237  
 Doran, J. 174, 175  
 Doran, C. 944  
 Dorf, R. C. 90  
 Doucet, A. 672  
 Dow, R. J. F. 867  
 Dowling, W. F. 292  
 Dowty, D. 945  
 Doyle, J. 90, 205, 439, 440, 702  
 Drabble, B. 535  
 Draper, D. 537  
 Drebbel, C. 47  
 Dreussi, J. 536  
 Dreyfus, H. L. 436, 1077, 1078,  
 1079, 1094, 1100  
 Dreyfus, S. E. 128, 129, 740,  
 1077, 1078, 1079  
 Drucker, H. 867, 1020  
 Druzdzel, M. J. 619  
 Du, D. 293  
 Dubois, D. 292, 621  
 Duda, R. O. 574, 621, 864, 868  
 Dudek, G. 1066

- Duffy, K. 866  
 Dumais, S. T. 978  
 Dunham, B. 54  
 Durfee, E. H. 539  
 Durrant-Whyte, H. 1063  
 Dürer, A. 1017  
 Dyer, M. 56, 945  
 Džeroski, S. 811, 815, 947
- E**
- Early, J. 945  
 Eastlake, D. E. 237  
 Ebeling, C. 237  
 Eckert, J. 46  
 Eco, U. 943  
 Edmonds, D. 48  
 Edmonds, J. 40  
 Edwards, D. J. 237  
 Edwards, P. 1094  
 Edwards, W. 702  
 Egedi, D. 944  
 Ehrenfeucht, A. 778  
 Einstein, A. 29, 40  
 Eiter, T. 439  
 Elfes, A. 1063  
 Elhadad, M. 945  
 Elkan, C. 616, 864  
 Elliott, G. L. 204  
 Elman, J. 947  
 Empson, W. 946  
 Enderton, H. B. 328, 377  
 Ephrati, E. 539  
 Erdmann, M. A. 129  
 Erman, L. D. 672  
 Ernst, G. 174  
 Ernst, H. A. 1064  
 Ernst, M. 489  
 Erol, K. 536  
 Essig, A. 573  
 Etchemendy, J. 295  
 Etzioni, O. 91, 537, 1090, 1102  
 Evans, T. G. 52  
 Everett, B. 1064
- F**
- Fagin, R. 205, 438  
 Fahlman, S. E. 53, 439, 440  
 Farrell, R. 378  
 Faugeras, O. 1018, 1019, 1020  
 Fearing, R. S. 1065  
 Featherstone, R. 1065
- Feigenbaum, E. A. 55, 56, 63, 434, 777  
 Feinstein, M. H. 63  
 Feldman, J. 63, 703  
 Feldman, R. 814  
 Fellbaum, C. 944  
 Feller, W. 1111  
 Felner, A. 176  
 Feng, C. 815  
 Feng, L. 1064  
 Fermat, P. 40, 572  
 Ferraris, P. 537  
 Fikes, R. E. 91, 380, 487, 536, 538, 814, 1063  
 Findlay, J. N. 435  
 Finney, D. J. 619  
 Firby, J. 1058  
 Firby, R. J. 538  
 Fischer, M. J. 436  
 Fisher, R. A. 573  
 Fix, E. 865  
 Fogel, D. B. 178  
 Fogel, L. J. 178  
 Forbes, J. 898  
 Forbus, K. D. 378, 440  
 Ford, K. M. 63, 1093  
 Forestier, J.-P. 899  
 Forgy, C. 378  
 Forrest, S. 177  
 Forsyth, D. 1019, 1020  
 Fortescue, M. D. 300  
 Fortmann, T. E. 671  
 Foster, D. W. 981  
 Fourier, J. 203  
 Fox, D. 1064  
 Fox, M. S. 489, 535  
 Frakes, W. 979  
 Francis, S. 979  
 Franco, J. 292  
 Frank, M. 206  
 Frank, R. H. 1088  
 Freeman, W. 620  
 Frege, G. 28, 39, 291, 327, 376, 377, 381  
 Freuder, E. C. 204, 205, 206  
 Freund, Y. 777  
 Friedberg, R. M. 54, 178  
 Friedman, G. J. 177  
 Friedman, J. 777, 868  
 Friedman, N. 864, 865  
 Fristedt, B. 897  
 Frost, D. 206  
 Fry, D. B. 672
- Fuchs, J. J. 535  
 Fudenberg, D. 742  
 Fukunaga, A. S. 535  
 Fung, R. 619  
 Furey, T. 866  
 Furnas, G. W. 978  
 Furst, M. 489
- G**
- Gaifman, H. 620  
 Gale, W. A. 978  
 Galilei, G. 31, 88, 812  
 Gallaire, H. 378  
 Gallier, J. H. 292, 328, 377  
 Gallo, G. 129  
 Gamba, A. 866  
 Gamberini, L. 866  
 Garcia, P. 947  
 Garding, J. 1019  
 Gardner, M. 291  
 Gary, M. R. 1111  
 Garfield, J. L. 63  
 Garrett, C. 161  
 Gaschnig, J. 174, 180, 204, 205, 621  
 Gasquet, A. 535  
 Gasser, R. 176, 231  
 Gat, E. 1065  
 Gauss, K. F. 129, 203, 248, 670  
 Gawande, A. 1090  
 Gawron, J. M. 673  
 Ge, N. 946  
 Gee, A. H. 659  
 Geffner, H. 489, 535, 537  
 Geiger, D. 618  
 Gelatt, C. D. 177, 205  
 Gelb, A. 671  
 Gelernter, H. 50, 380  
 Gelfond, M. 439  
 Gelman, A. 868  
 Geman, D. 619  
 Geman, S. 619  
 Genesereth, M. R. 90, 129, 359, 365, 380, 381, 384, 1069  
 Gentner, D. 814  
 Gentzen, G. 377  
 Georgeff, M. P. 538  
 Gerbault, F. 865  
 Gerevini, A. 489  
 Germann, U. 976  
 Gershwin, G. 664  
 Ghahramani, Z. 620, 671, 672, 898  
 Ghallab, M. 487, 489, 535

- Ghose, S. 175  
 Giacomo, G. D. 436  
 Gibson, J. J. 1018  
 Gilks, W. R. 620, 864  
 Gilmore, P. C. 377  
 Gini, M. 537  
 Ginsberg, M. L. 205, 206, 233, 380, 384, 538, 622  
 Gittins, J. C. 897  
 Giunchiglia, E. 537  
 Givan, R. 945  
 Glanc, A. 1062  
 Glavieux, A. 620  
 Glover, F. 176  
 Glymour, C. 1083  
 Goebel, J. 865  
 Goebel, R. 32, 60, 90  
 Gold, B. 673  
 Gold, E. M. 777, 946, 947  
 Goldberg, D. E. 177  
 Golden, K. 538  
 Goldman, N. 945, 946  
 Goldman, R. 537, 620, 945, 946  
 Goldszmidt, M. 864  
 Golgi, C. 42  
 Gomard, C. K. 813  
 Gomes, C. 176  
 Good, I. J. 236, 618, 778, 1091  
 Gooday, J. M. 440  
 Goodman, D. 60  
 Goodman, N. 437, 813  
 Goodnow, J. J. 777  
 Gordon, M. J. 328  
 Gordon, N. J. 672  
 Gorry, G. A. 573  
 Gottlob, G. 205, 379  
 Gotts, N. 440  
 Gödel, K. 39, 359, 377, 1076  
 Graham, S. L. 944  
 Grassmann, H. 327  
 Grayson, C. J. 682  
 Green, B. 945  
 Green, C. 51, 375, 378, 381, 436, 487  
 Greenbaum, S. 944  
 Greenblatt, R. D. 237  
 Greenstreet, M. R. 381  
 Greiner, R. 814  
 Grice, H. P. 943  
 Grosf, B. 814  
 Grosz, B. J. 737, 742, 946, 947  
 Grove, A. 573, 702  
 Grove, W. 1075  
 Grumberg, O. 489  
 Grundy, W. 866  
 Gu, J. 205  
 Guard, J. 374, 381  
 Guha, R. V. 435  
 Guibas, L. J. 1064  
 Gutfreund, H. 866  
 Guthrie, F. 203  
 Guy, R. K. 130  
 Guyon, I. 867, 1020  
 Guyon, I. M. 867  
 Gyorfi, L. 864
- H**
- Haas, A. 438  
 Hacking, I. 574  
 Hähnel, D. 1064  
 Haken, W. 203  
 Hald, A. 574  
 Hale, J. 946  
 Halpern, J. Y. 573, 620  
 Halpin, M. P. 206  
 Hamming, R. W. 574  
 Hammond, K. 536  
 Hamscher, W. 91  
 Handschin, J. E. 671  
 Hanks, S. 537  
 Hanna, F. K. 816  
 Hansard, W. 973  
 Hansen, E. 175, 537, 741  
 Hansen, P. 292  
 Hanski, I. 91  
 Hansson, O. 176, 180  
 Harada, D. 899  
 Haralick, R. M. 204  
 Hardin, G. 742  
 Harel, D. 436  
 Harman, G. 1094  
 Harrison, M. A. 944  
 Harsányi, J. 742  
 Harshman, R. A. 978  
 Hart, P. E. 174, 574, 621, 864, 868  
 Hart, T. P. 237  
 Hartley, R. 1019  
 Haslum, P. 535  
 Hastie, T. 865, 868  
 Haugeland, J. 32, 63, 1077, 1094  
 Haussler, D. 778, 815, 866  
 Havelund, K. 375  
 Hayes, P. J. 63, 435, 436, 437, 438, 440, 1093  
 Hayes-Roth, F. 672
- Hearst, M. A. 981  
 Heawood, P. 1077  
 Hebb, D. O. 48, 53, 896  
 Heckerman, D. 59, 61, 614, 618, 621, 622, 703  
 Hefaisztosz, 1063  
 Heim, I. 944  
 Heinz, E. A. 238  
 Held, M. 176  
 Helmholtz, H. 44, 1017, 1018  
 Hempel, C. 37  
 Henderson, T. C. 193, 204  
 Hendler, J. 536  
 Hendrix, G. G. 422  
 Henrion, M. 588, 619, 703  
 Henzinger, M. 965, 979  
 Henzinger, T. A. 90  
 Herbrand, J. 291, 336, 377  
 Herskovits, E. 864  
 Hewitt, C. 378, 379  
 Hierholzer, C. 178  
 Hilbert, D. 39  
 Hilgard, E. R. 897  
 Hingorani, S. L. 671  
 Hintikka, J. 437  
 Hinton, G. E. 57, 177  
 Hirsh, H. 814  
 Hirst, G. 945, 946  
 Ho, Y.-C. 54  
 Hoane, J. 230  
 Hobbes, T. 36  
 Hobbs, J. R. 441, 939, 940, 945, 946, 980  
 Hockey, B. A. 944  
 Hodges, J. L. 865  
 Hoff, M. E. 53, 889, 896, 897  
 Hoffman, J. 489  
 Hoffmann, J. 489  
 Hogan, N. 1065  
 Holland, J. H. 177  
 Holldobler, S. 436  
 Hollerbach, J. M. 1065  
 Holloway, J. 237  
 Holzmann, G. J. 375  
 Hon, H. 673  
 Honavar, V. 947  
 Hong, J. 777  
 Hood, A. 42  
 Hopcroft, J. 1064, 1111  
 Hopfield, J. J. 57, 867  
 Horn, A. 291  
 Horn, B. K. P. 1019, 1020

- Horning, J. J. 947  
 Horowitz, E. 129  
 Horowitz, M. 375  
 Horrocks, J. C. 573  
 Horswill, I. 1057, 1065  
 Horvitz, R. 618, 622, 703, 1100  
 Horvitz, E. J. 59  
 Hovel, D. 618, 622  
 Hovy, E. 945  
 Howard, R. A. 691, 702, 703, 740  
 Hsu, F.-H. 230, 238  
 Huang, T. 671  
 Huang, X. D. 673  
 Hubel, D. 1020  
 Huber, D. 1064  
 Huddleston, R. D. 944  
 Huffman, D. A. 53, 1004  
 Huffman, S. 1064  
 Hughes, B. D. 170  
 Huhn, M. N. 91  
 Hume, D. 37, 946  
 Hunsberger, L. 737, 742  
 Hunt, E. B. 777  
 Hunter, L. 865  
 Hurwicz, L. 742  
 Hutchins, W. J. 980  
 Huttenlocher, D. P. 1019  
 Huygens, C. 572, 741  
 Hwa, R. 978  
 Hwang, C. H. 435, 945  
 Hyun, S. 535, 1064
- I**
- Indyk, P. 865  
 Ingerman, P. Z. 944  
 Inoue, K. 810  
 Intille, S. 671  
 Inza, I. 177, 181  
 Isard, M. 659, 672  
 Israel, D. 980
- J**
- Iaakkola, T. 620, 672, 898  
 Jackel, L. 867, 1020  
 Jacquard, J. 46  
 Jaffar, J. 380  
 Jahr, M. 976  
 James, H. 44  
 James, W. 44  
 Jaskowski, S. 377  
 Jaumard, B.
- Jeavons, P. 206  
 Jefferson, G. 1079  
 Jeffrey, R. C. 381, 573, 702  
 Jeffreys, H. 978  
 Jelinek, F. 672, 673, 978  
 Jenkin, M. 1066  
 Jennings, H. S. 44  
 Jensen, F. 618  
 Jensen, F. V. 618, 622  
 Jespersen, O. 944  
 Jevons, W. S. 291, 815  
 Jimenez, P. 537  
 Joachims, T. 979  
 Johnson, C. R. 91  
 Johnson, D. S. 1111  
 Johnson, M. 946, 947, 950  
 Johnson, W. W. 128  
 Johnson-Laird, P. N. 63  
 Johnston, M. D. 176, 205, 536  
 Jones, M. 574  
 Jones, N. D. 813, 814  
 Jones, R. 378  
 Jonsson, A. 91, 535  
 Jordan, M. I. 620, 622, 671, 672, 741, 867, 894, 898  
 Joshi, A. K. 944, 946  
 Joskowicz, L. 440  
 Joslin, D. 489  
 Jouannaud, J.-P. 380  
 Joule, J. 812  
 Juang, B.-H. 673  
 Judd, J. S. 867  
 Juels, A. 177  
 Julesz, B. 1018  
 Jurafsky, D. 63, 673, 946, 947, 981
- K**
- Kadane, J. B. 742  
 Kaelbling, L. P. 741, 899, 1065  
 Kager, R. 946  
 Kahneman, D. 32, 585, 685  
 Kaindl, H. 175  
 Kalman, R. E. 643, 670  
 Kambhampati, S. 489, 535, 536  
 Kameya, Y. 621  
 Kaneyama, M. 980  
 Kan, A. 129, 499  
 Kanade, T. 999  
 Kanal, L. N. 175, 176, 622  
 Kanazawa, K. 671, 672, 741  
 Kanefsky, B. 40, 292
- Kanoui, H. 328, 379  
 Kant, E. 378  
 Kaplan, D. 438  
 Kaplan, R. 945, 980  
 Karmarkar, N. 177  
 Karmiloff-Smith, A. 947  
 Karp, R. M. 40, 129, 176, 1111  
 Karypis, G. 964  
 Kasami, T. 944  
 Kasper, R. T. 945  
 Kassirer, J. P. 573  
 Kaszparov, G. 60, 230  
 Kaufmann, M. 382  
 Kautz, D. 489  
 Kautz, H. 176, 489  
 Kavraki, L. 1065  
 Kay, A. R. 43  
 Kay, M. 673, 971, 980  
 Kaye, R. 292  
 Keane, M. A. 178  
 Kearns, M. 741, 778, 897  
 Kedar-Cabelli, S. 814  
 Keene, R. 60  
 Keeney, R. L. 685, 690, 702  
 Kehler, A. 946  
 Keil, F. C. 34, 63, 1094  
 Keim, G. A. 61, 206  
 Keller, R. 814  
 Kelly, J. 621, 865  
 Kemp, M. 1017  
 Kempe, A. B. 1077  
 Kempelen, F. 236  
 Kenley, C. R. 619  
 Kepler, J. 643, 1017  
 Kern, C. 381  
 Kernighan, B. W. 129  
 Keynes, J. M. 572  
 Khatib, O. 1065  
 Khorsand, A. 175  
 Kietz, J.-U. 815  
 Kim, J. H. 618  
 King, R. D. 812  
 Kirby, M. 1020  
 Kirchner, C. 380  
 Kirkpatrick, S. 177, 205, 292  
 Kirman, J. 741  
 Kirousis, L. M. 1019  
 Kitano, H. 1064  
 Kjaerulff, U. 671  
 Klein, D. 978  
 Kleinberg, J. M. 979  
 Knight, K. 32, 976, 980, 981, 982  
 Knoblock, C. A. 128, 536

- Knuth, D. E. 237, 380, 944, 1064,  
 1111  
 Koditschek, D. 1065  
 Koehler, J. 489  
 Koenderink, J. J. 1018, 1019  
 Koenig, S. 129, 178, 741  
 Kohn, W. 380  
 Koller, D. 573, 621, 671, 742, 979  
 Kolmogorov, A. N. 555, 572, 573,  
 670  
 Kolodner, J. 56, 814, 945  
 Kondrak, G. 204, 206  
 Konolige, K. 438, 439, 538, 1065  
 Koopmans, T. C. 740  
 Kopernikusz, 1089  
 Koren, Y. 1065  
 Korf, R. E. 129, 130, 175, 176, 178,  
 179, 238, 488  
 Kortenkamp, D. 1064, 1066  
 Koss, F. 1064  
 Kotok, A. 237  
 Koutsoupias, E. 176  
 Kowalski, R. 360, 379, 380, 436,  
 437, 439  
 Koza, J. R. 178  
 Kramnik, V. 231  
 Kratzer, A. 944  
 Kraus, S. 539  
 Kraus, W. F. 177  
 Krauss, P. 620  
 Kripke, S. A. 437  
 Krishnan, T. 864, 865  
 Krovetz, R. 979  
 Kruppa, E. 1018  
 Kucera, H. 979  
 Kuehner, D. 379  
 Kuhn, H. W. 742  
 Kuhns, J.-L. 979  
 Kuipers, C. 177, 181  
 Kuipers, B. J. 1063  
 Kukich, K. 979  
 Kulikowski, C. 778  
 Kumar, P. R. 175, 176  
 Kumar, V. 90, 175, 176, 206, 964  
 Kuniyoshi, Y. 1064  
 Kuper, G. M. 379  
 Kurzweil, R. 32, 1091  
 Kyburg, H. E. 573
- L**  
 Ladkin, P. 437  
 Ladner, R. E. 436
- Lafferty, J. 979  
 Laguna, M. 176  
 Laird, J. E. 59, 349, 378, 536, 814,  
 1099  
 Laird, N. 671, 864  
 Laird, P. 176, 205  
 Lakoff, G. 435, 946  
 Lakemeyer, G. 1064  
 Lamarck, J. 164  
 La Mettrie, J. O. 1089, 1093  
 La Mura, P. 702  
 Landauer, T. K. 978  
 Langley, P. 816  
 Langlotz, C. P. 59, 91  
 Langton, C. 177  
 Lansky, A. L. 538  
 Laplace, P. 40, 556, 572, 611  
 Lappin, S. 946  
 Lari, K. 978  
 Larkey, P. D. 742  
 Larrañaga, P. 177, 181  
 Larsen, B. 618  
 Larson, G. 792  
 Laruelle, H. 535  
 Lassez, J.-L. 380  
 Lassila, O. 435  
 Latombe, J.-C. 535, 1064, 1065  
 Laugherty, K. 682  
 Laurelle, H. 489  
 Lauritzen, S. 618, 619, 622, 703  
 Lavrač, N. 777, 811, 815  
 Lawler, E. L. 129, 175, 499  
 Lazanas, A. 1065  
 Leacock, C. 1075  
 Leaper, D. J. 573  
 Leass, H. J. 946  
 LeCun, Y. 867, 1020  
 Lederberg, J. 55  
 Lee, K.-F. 673  
 Lee, R. C.-T. 619  
 Lee, T. M. 43, 382  
 Leech, G. 944, 979  
 Lefkovitz, D. 239  
 Lehmann, D. 539  
 Leibniz, G. W. 37, 572, 741  
 Leimer, H. 618  
 Leiserson, C. E. 1111  
 Lemmer, J. F. 622  
 Lenat, D. B. 435, 445, 816  
 Lenstra, J. K. 129, 499  
 Lenzerini, M. 439  
 Leonard, H. S. 437  
 Leonard, J. J. 1063
- Leone, N. 205, 379, 439  
 Lesh, N. 537  
 Leśniewski, S. 437  
 Lespérance, Y. 436  
 Lesser, V. R. 539, 672  
 Lettvin, J. Y. 1015  
 Letz, R. 381  
 Levesque, H. J. 436, 439, 441,  
 539, 1058  
 Levitt, G. M. 236  
 Levitt, T. S. 1063  
 Levy, D. N. 239  
 Lewis, D. D. 980  
 Lewis, D. K. 91, 945, 1093  
 Lewis, R. A. 812  
 Lifschitz, 439, 487  
 Li, C. M. 292  
 Li, M. 778  
 Lieberman, L. 1019  
 Lifschitz, V. 439, 487  
 Lighthill, J. 54, 57  
 Lin, D. 866  
 Lin, F. 436  
 Lin, S. 129, 174  
 Linden, T. A. 537  
 Lindsay, R. K. 434, 980  
 Linné, C. 435  
 Lipkis, T. A. 439  
 Littman, M. L. 61, 177, 206, 537,  
 742  
 Liu, J. S. 672  
 Liu, W. 865  
 Lloyd, E. K. 203  
 Lloyd, J. W. 379  
 Locke, J. 37, 943  
 Locke, W. 980  
 Lodge, D. 1103  
 Logemann, G. 276  
 Lohn, J. D. 177  
 Long, D. 489  
 Longuet-Higgins, H. C. 1018  
 Lotem, A. 490  
 Lovejoy, W. S. 741  
 Lovelace, A. 46  
 Loveland, D. 276, 379, 381, 382  
 Lowe, D. G. 1019  
 Lowere, B. 176, 672  
 Lowrance, J. D. 621  
 Lowry, M. 375  
 Lowry, M. R. 381  
 Loyd, S. 128  
 Lozano-Perez, T. 1064, 1065  
 Löwenheim, L. 328

- Luby, M. 619, 672  
 Lucas, J. R. 1076, 1077  
 Lucas, P. 622, 698, 700  
 Luce, D. R. 742  
 Luger, G. F. 63  
 Lugosi, G. 864  
 Lull, R. 27, 36  
 Luong, Q.-T. 1019  
 Lusk, E. 382
- M**
- Machover, M. 328  
 MacKay, D. J. C. 867  
 Mackworth, A. K. 32, 60, 90, 204, 206  
 Madigan, C. F. 292  
 Mahanti, A. 179  
 Mahaviracarya, 572  
 Maier, D. 205, 379  
 Majercik, S. M. 537  
 Makov, U. E. 865  
 Mali, A. D. 536  
 Malik, J. 671, 867, 994, 1019, 1020  
 Malik, S. 292  
 Mann, W. C. 946  
 Manna, Z. 381  
 Manning, C. D. 978, 979, 981  
 Manolios, P. 382  
 Mansour, Y. 741  
 Marais, H. 965, 979  
 Marbach, P. 898  
 Marcu, D. 976  
 Marin, J. 777  
 Markov, A. A. 670  
 Maron, M. E. 574, 979  
 Marriott, K. 204, 206, 380  
 Marsland, A. T. 239  
 Martelli, A. 175, 381  
 Marthi, B. 672  
 Martin, J. H. 63, 673, 946, 947, 981  
 Martin, N. 378  
 Martin, P. 536, 945  
 Maslov, S. Y. 378  
 Mason, M. 129, 537, 1065, 1066  
 Mataric, M. J. 1065  
 Mateis, C. 439  
 Mates, B. 290  
 Matheson, J. E. 691, 702  
 Maturana, H. R. 1015  
 Mauchly, J. 46  
 Maxwell, J. 611, 945  
 Mayer, A. 176, 180
- Mayne, D. Q. 671  
 Mazumder, P. 129  
 McAllester, D. A. 58, 238, 374, 440, 488, 489, 945  
 McCarthy, J. 49, 51, 56, 90, 236, 377, 410, 436, 439, 1086  
 McCartney, R. D. 381  
 McCawley, J. D. 944  
 McClelland, J. L. 57  
 McConnell-Ginet, S. 63, 944  
 McCulloch, W. 47, 48, 53, 293, 844, 846, 865, 1015  
 McCune, 371, 374, 381  
 McDermott, D. 32, 378, 420, 438, 439, 489, 536, 538  
 McDermott, J. 57, 349, 378  
 McDonald, R. 299  
 McEliece, R. J. 620  
 McGill, M. J. 979  
 McGregor, J. J. 204  
 McGuinness, D. L. 423, 439  
 McKeown, K. 945  
 McLachlan, G. J. 864, 865  
 McMillan, K. L. 489  
 McNealy, S. 1089  
 Meehan, J. 378  
 Meehl, P. 1075  
 Megiddo, N. 742  
 Melčuk, I. A. 944  
 Mellish, C. S. 380  
 Mendel, G. 163  
 Mercer, J. 978  
 Merkhofer, M. M. 702  
 Merlin, P. M. 379  
 Metropolis, N. 177, 619  
 Mézard, M. 867  
 Michalski, R. S. 777, 778  
 Michaylov, S. 380  
 Michel, S. 981  
 Michie, 174, 175, 536, 1064  
 Michie, D. 63, 239, 891, 897  
 Middleton, B. 588  
 Milgrom, P. 742  
 Mill, J. S. 38, 784  
 Miller, A. C. 702  
 Miller, D. 538  
 Millstein, T. 489  
 Milner, A. J. 328  
 Minker, J. 378  
 Minsky, M. L. 48, 51, 54, 56, 63, 438, 865  
 Minton, S. 176, 205, 536, 814  
 Mitchell, D. 176, 292
- Mitchell, M. 177, 178  
 Mitchell, T. M. 91, 777, 778, 790, 791, 814, 1099  
 Mitkov, R. 946  
 Moffat, A. 979  
 Mohr, R. 193, 204  
 Mohri, M. 908  
 Moisl, H. 945, 947  
 Montague, P. R. 897  
 Montague, R. 437, 438, 945  
 Montanari, U. 175, 203, 204, 381  
 Montererlo, M. 1064  
 Mooney, R. 814, 947  
 Mooney, R. J. 941, 947  
 Moore, A. W. 176, 897  
 Moore, E. F. 129  
 Moore, J. D. 536  
 Moore, J. S. 374, 375, 381, 382, 438  
 Moore, R. C. 437  
 Moravec, H. P. 1063, 1083, 1091  
 More, T. 49  
 Morgan, J. 673, 944  
 Morgan, N. 673  
 Morgenstern, L. 438, 439  
 Morgenstern, O. 41, 236, 239, 702, 742  
 Moritz, M. 965, 979  
 Morris, P. 91, 535  
 Morrison, E. 236  
 Morrison, P. 236  
 Moses, Y. 438  
 Moskewicz, M. W. 292  
 Mosteller, F. 981  
 Mostow, J. 176, 180  
 Motzkin, T. S. 866  
 Moussouris, J. 237  
 Moutarlier, P. 1063  
 Mozetic, I. 777  
 Muggleton, S. H. 804, 810, 815, 947  
 Muller, U. 867, 1020  
 Mundy, J. 1018  
 Murakami, T. 232  
 Murga, R. 177, 181  
 Murphy, K. 620, 1064  
 Murphy, R. 1066  
 Muscettola, N. 91, 535, 536  
 Müller, M. 239  
 Myers, K. L. 538  
 Myerson, R. B. 742  
 Myraug, B. 438

- N**
- Nadal, J.-P. 867
  - Nagel, T. 1093
  - Nalwa, V. S. 44
  - Nardi, D. 439
  - Nash, J. 729, 742
  - Nau, D. S. 175, 233, 238, 490, 536
  - Nauck, 129
  - Naur, P. 944
  - Nayak, P. 91, 440, 535, 536
  - Nealy, R. 231
  - Nebel, B. 487, 489
  - Nelson, G. 381
  - Netto, E. 129
  - Neumann, J. 41, 47, 49, 236, 239, 702, 730, 742
  - Nevill-Manning, C. G. 941, 947
  - Nevins, A. J. 378
  - Newborn, M. M. 175
  - Newell, A. 34, 38, 45, 49, 50, 59, 91, 128, 174, 236, 237, 349, 378, 438, 487, 536, 814, 1099
  - Newman, P. 1063
  - Newton, I. 165, 176
  - Ng, A. Y. 741, 898
  - Nguyen, X. 489
  - Niblett, T. 815
  - Nicholson, A. 671, 741
  - Nielsen, P. E. 378
  - Niemelä, I. 439
  - Nilsson, D. 703
  - Nilsson, N. J. 32, 60, 63, 90, 91, 130, 174, 180, 236, 365, 381, 487, 620, 1063, 1069
  - Niranjan, M. 659
  - Nixon, R. 430
  - Noda, I. 1064
  - Norman, D. A. 980
  - North, T. 54
  - Norvig, P. 28, 60, 378, 586, 673, 946
  - Nourbakhsh, I. 129
  - Nowick, S. M. 375
  - Nowlan, S. J. 177
  - Nunberg, G. 946
  - Nussbaum, M. C. 38
  - Nygaard, K. 438
- O**
- Ockham, W. 776
  - Ogasawara, G. 671
  - Ogawa, S. 43
  - Oglesby, F. 374
  - Olatainty, B. 535
  - Olawsky, D. 537
  - Olesen, K. G. 618, 619
  - Oliver, R. M. 703
  - Olshen, R. A. 777
  - Olson, C. F. 1014, 1019
  - Olum, P. 1018
  - Omohundro, S. 978
  - Oncina, J. 947
  - Oppacher, F. 177
  - Oppen, D. C. 381
  - Ormoneit, D. 898
  - Ortony, A. 946
  - Osawa, E. 1064
  - Osborne, M. J. 742
  - Osborner, D. N. 778
  - Ostland, M. 671
  - Overbeek, R. 382
  - Overmars, M. 1065
  - Owens, A. J. 178
- P**
- Page, C. D. 815
  - Page, L. 979
  - Pak, I. 672
  - Palay, A. 238
  - Pallottino, S. 129
  - Palmer, D. A. 981
  - Palmer, S. 1020
  - Palmieri, G. 866
  - Pang, C.-Y. 129
  - Pánini, 48, 944
  - Papadimitriou, C. H. 176, 178, 740, 978, 1019, 1111
  - Papadopoulou, T. 1019
  - Papavassiliou, V. 898
  - Papert, S. 54
  - Pardalos, P. M. 293
  - Parekh, R. 947
  - Pareto, V. 729
  - Parisi, D. 947
  - Parisi, G. 620
  - Park, S. 375
  - Parker, D. B. 866
  - Parker, L. E. 1065
  - Part, R. 899
  - Parrod, Y. 535
  - Parzen, E. 865
  - Pascal, B. 36, 40, 572
  - Pasero, R. 328, 379
  - Pasula, H. 621, 671, 672
  - Paterson, M. S. 381
  - Patil, R. 439, 945
  - Patrick, B. G. 175
  - Patten, T. 945
  - Paul, R. P. 1065
  - Paull, M. 292
  - Pazzani, M. 574
  - Peano, G. 327
  - Pearl, J. 59, 91, 174, 176, 205, 239, 412, 576, 577, 580, 586, 618, 619, 620, 622, 706
  - Pearson, J. 206
  - Pecheur, C. 375
  - Pednault, E. P. D. 487, 538
  - Peirce, C. S. 204, 419, 438, 943, 945
  - Peled, D. 381
  - Pelikan, M. 177
  - Pell, B. 91, 535, 536
  - Pemberton, J. C. 179
  - Penberthy, J. S. 488
  - Peng, J. 897
  - Penix, J. 375
  - Penrose, R. 1076
  - Peot, M. 537, 619
  - Pereira, F. 352, 908, 944, 947
  - Pereira, L. M. 352
  - Peres, Y. 672
  - Perrault, C. R. 944
  - Person, C. 897
  - Peters, S. 945
  - Peterson, 620
  - Petrie, T. 671
  - Pfeffer, A. 621, 742
  - Pfeifer, G. 439
  - Philips, A. B. 176, 205
  - Pijls, W. 238
  - Pinker, S. 946
  - Pitts, W. 47, 48, 53, 293, 844, 846, 865
  - Pitts, W. H. 1015
  - Plaat, A. 238
  - Place, U. T. 1093
  - Plamondon, P. 981
  - Plankalkül, 46
  - Platón, 437, 1092
  - Plotkin, G. 380, 815
  - Plunkett, K. 947
  - Pnueli, A. 436
  - Podelski, A. 380

- Pohl, I. 129, 174  
 Polguere, A. 944  
 Pollack, M. E. 489, 944  
 Pomerleau, D. A. 1020  
 Ponte, J. M. 979  
 Poole, D. 32, 60, 90, 618, 621  
 Popper, K. R. 573  
 Porfiriusz, 438  
 Porter, M. F. 979  
 Posegga, J. 381  
 Post, E. L. 291  
 Prade, H. 621  
 Pradhan, M. 588  
 Pratt, V. R. 436  
 Prawitz, D. 377  
 Press, W. H. 177  
 Prete, F. 1018  
 Price, B. 436, 741  
 Prieditis, A. 176, 180  
 Prinz, D. G. 236  
 Prior, A. N. 435  
 Prosser, P. 204  
 Protagoras, 943  
 Proust, M. 974  
 Provan, G. M. 588  
 Pryor, L. 537  
 Puget, J.-F. 815  
 Pullum, G. K. 944, 947  
 Puterman, M. L. 90, 740  
 Putnam, H. 91, 365, 377, 573, 1093  
 Puzicha, J. 867, 1020  
 Pylyshyn, Z. 1094
- Q**
- Quillian, M. R. 438  
 Quine, W. V. 395, 435, 437  
 Quinlan, E. 980  
 Quinlan, J. R. 777, 806, 808, 815, 980  
 Quirk, R. 944
- R**
- Rabani, Y. 177  
 Rabideau, G. 535  
 Rabiner, L. R. 673  
 Rabinovich, Y. 177  
 Raibert, M. 1028  
 Raghavan, P. 978  
 Raiffa, H. 685, 690, 702, 742  
 Rajan, K. 91, 535  
 Ramekrishnan, R. 378
- Ramon y Cajal, S. 42  
 Ramsey, F. P. 41, 573, 702  
 Rao, A. 91  
 Rao, B. 671  
 Raphael, B. 174, 378, 814  
 Raphson, J. 165, 176  
 Raschke, U. 1064  
 Rassenti, S. 742  
 Ratner, D. 128  
 Rauch, H. E. 670, 671  
 Rayner, M. 799  
 Rayson, P. 949  
 Reboh, R. 380  
 Rechenberg, I. 177  
 Reddy, R. 672  
 Regin, J. 204  
 Reichenbach, H. 573  
 Reif, J. 1065  
 Reingold, E. M. 204  
 Reiss, M. 232  
 Reiter, E. 945  
 Reiter, R. 436, 439, 741  
 Reitman, W. 239  
 Remus, H. 239  
 Rényi, A. 572  
 Rescher, N. 435  
 Reynolds, C. W. 539  
 Ribeiro-Neto, B. 979  
 Rice, T. R. 702  
 Rich, E. 32  
 Richardson, M. 673  
 Richardson, S. 620  
 Rieger, C. 56, 945  
 Riesbeck, C. 56, 378, 945  
 Riley, M. 908  
 Ringle, M. 1094  
 Rintanen, J. 537  
 Ripley, B. D. 868  
 Rissland, E. L. 63  
 Ristad, E. S. 949  
 Ritchie, G. D. 816  
 Ritov, Y. 671  
 Rivest, R. 1111  
 Roberts, L. G. 1019  
 Roberts, M. 236  
 Robertson, N. 205  
 Robertson, S. E. 574, 979  
 Robinson, A. 377  
 Robinson, G. 380  
 Robinson, H. 45  
 Robinson, J. A. 51, 359, 378  
 Roche, E. 980  
 Rochester, N. 49, 50
- Rock, I. 1020  
 Roossin, P. 980  
 Rorty, R. 1093  
 Rosenblatt, F. 53, 1018  
 Rosenblitt, D. 488  
 Rosenbloom, P. S. 59, 349, 378, 536, 814, 1099  
 Rosenblueth, A. 47  
 Rosenbluth, A. 177, 619  
 Rosenbluth, M. 177, 619  
 Rosenfeld, E. 63  
 Rosenholtz, R. 1001, 1019  
 Rosenschein, J. S. 742  
 Rosenschein, S. J. 91, 1065  
 Rosenthal, D. M. 1094  
 Ross, S. M. 574, 1111  
 Rossi, F. 204  
 Roussel, P. 328, 379  
 Rouveiro, C. 815  
 Roveri, M. 490  
 Rowat, P. F. 1064  
 Roweis, S. T. 671  
 Rozonoer, L. 868  
 Rubin, D. 671, 672, 864, 868  
 Rubinstein, A. 742  
 Rumelhart, D. E. 57  
 Ruspini, E. H. 621  
 Russell, B. 37, 47, 377, 795  
 Russel, J. G. B. 702  
 Russel, S. J. 28, 49, 60, 175, 238, 359, 586, 621, 671, 672, 673, 702, 741, 814, 898, 899, 1058, 1064, 1100, 1102  
 Rustagi, J. S. 620  
 Ruzzo, W. L. 944  
 Ryder, J. L. 239
- S**
- Sabin, D. 204  
 Sacerdoti, E. D. 380, 488, 536  
 Sackinger, E. 867, 1020  
 Sacks, E. 440  
 Sadri, F. 437  
 Sag, J. 944, 946  
 Sagalowicz, D. 380  
 Sager, N. 944  
 Sagiv, Y. 379  
 Sahami, M. 622, 978, 980  
 Sahni, S. 129  
 Salisbury, J. 1065  
 Salmond, D. J. 672  
 Salomaa, A. 978

- Salton, G. 979  
 Samuel, A. L. 49, 50, 91, 231, 236, 890, 896  
 Samuelsson, C. 799  
 Sanna, R. 866  
 Saraswat, V. 204  
 Sastry, S. 90  
 Sato, T. 380, 621  
 Saul, L. K. 620, 672, 898  
 Saund, E. 978  
 Savage, L. J. 558, 573, 702  
 Sayre, K. 1073  
 Scarcello, F. 205, 379, 439  
 Schabes, Y. 944, 980  
 Schaeffer, J. 176, 231, 238, 239  
 Schank, R. C. 56, 945  
 Schapire, R. E. 777  
 Scharir, M. 1064  
 Scheines, R. 864  
 Scherl, R. 436, 1058  
 Schickard, W. 36  
 Schmolze, J. G. 439  
 Schneeberger, J. 436  
 Schnitzius, D. 536  
 Schoenberg, I. J. 866  
 Schofield, P. D. A. 128  
 Schoppers, M. J. 538  
 Schölkopf, B. 866, 867, 868  
 Schönning, T. 292  
 Schrag, R. C. 292  
 Schröder, E. 291  
 Schubert, L. K. 435, 489, 945  
 Schultz, W. 897  
 Schulz, D. 1064  
 Schumann, J. 381  
 Schütze, H. 947, 978, 979, 981  
 Schwartz, J. T. 1064  
 Schwartz, S. P. 435  
 Schwartz, W. B. 573  
 Scott, D. 620  
 Scriven, M. 1093  
 Searle, J. R. 43, 943, 1080, 1081, 1083, 1084, 1085, 1086, 1093  
 Sejnowsky, T. 897  
 Self, M. 621, 865  
 Selfridge, M. 980  
 Selfridge, O. G. 49  
 Selman, B. 176, 292, 439, 489  
 Sen, S. 898  
 Sergot, M. 437  
 Serina, I. 489  
 Sestoft, P. 813  
 Settle, L. 374
- Seymour, P. D. 205  
 Shachter, R. D. 586, 618, 619, 703, 741  
 Shafer, G. 621, 622  
 Shaham, R. W. 1094  
 Shahookar, K. 129  
 Shakespeare, W. 981  
 Shallal, L. 380  
 Shanahan, M. 436, 437  
 Shankar, N. 374  
 Shannon, C. E. 49, 210, 219, 236, 760, 777, 978  
 Shapiro, E. 815  
 Shapiro, S. C. 63  
 Shapley, S. 742  
 Sharir, M. 1064  
 Sharp, D. H. 865  
 Shavlik, J. 778  
 Shaw, J. C. 236, 291  
 Shawe-Taylor, J. 866, 868  
 Shazeer, N. M. 61, 206  
 Sheiber, S. M. 947  
 Shelley, M. 1090  
 Shenoy, P. P. 621  
 Shewchuk, J. 535, 1064  
 Shi, J. 994  
 Shieber, S. M. 947  
 Shimelevich, L. I. 671  
 Shin, M. C. 740  
 Shmoys, D. B. 129, 499, 536  
 Shoham, Y. 91, 380, 437, 702  
 Shortliffe, E. H. 56, 621  
 Shwe, M. 619  
 Sidner, C. L. 946  
 Siekmann, J. 382  
 Sietsma, J. 867  
 Siklossy, L. 536  
 Silverstein, C. 965, 979  
 Simard, P. 867, 1020  
 Simmons, R. 129, 945  
 Simon, H. A. 34, 38, 41, 45, 49, 50, 53, 62, 91, 128, 174, 230, 236, 237, 375, 381, 487, 1101  
 Simon, J. C. 292  
 Simons, P. 439  
 Sinclair, A. 177  
 Singh, M. P. 91  
 Singh, S. P. 897  
 Sirovitch, L. 1020  
 Skinner, B. F. 47, 90  
 Skolem, Th. 377  
 Slagle, J. R. 51, 237  
 Stake, D. J. 129
- Slater, E. 236  
 Sleator, D. 944  
 Slocum, J. 945  
 Sloman, A. 1094  
 Slovic, P. 32  
 Smallwood, R. D. 741  
 Smith, A. 40  
 Smith, A. F. M. 672, 828, 865  
 Smith, B. 91, 535  
 Smith, D. E. 359, 380, 384, 537, 703  
 Smith, D. R. 381  
 Smith, J. M. 177  
 Smith, J. Q. 703  
 Smith, R. C. 1063  
 Smith, R. G. 91  
 Smith, S. J. J. 233  
 Smith, V. 742  
 Smith, W. D. 239  
 Smola, A. J. 868  
 Smolensky, P. 58  
 Smyth, P. 671  
 Soderland, S. 488  
 Solomonoff, R. 49  
 Somers, H. 945, 947, 980  
 Sompolinsky, H. 866  
 Sondik, E. J. 741  
 Sosic, R. 205  
 Souchanski, M.  
 Sowa, J. 441  
 Sparck, J. K. 979  
 Spiegelhalter, D. J. 618, 620, 864  
 Spielberg, S. 1091  
 Spirtes, P. 864  
 Springsteen, B. 1088  
 Sproull, R. F. 703  
 Srinivas, B. 944  
 Srinivasan, A. 812, 815  
 Srivas, M. 375  
 Srivastava, B. 536  
 Stadler, J. 535  
 Stallman, R. M. 204  
 Stanfill, C. 865  
 Staniland, J. R. 573  
 States, D. J. 865  
 Steel, S. 536, 538  
 Stefk, M. 441, 621  
 Stein, L. A. 1103  
 Steinbach, M. 964  
 Steiner, W. 1064  
 Stern, H. S. 868  
 Stickel, M. J. E. 373, 381, 945, 980  
 Stiller, L. B. 231

- Stillings, N. A. 63  
 Stob, M. 778  
 Stockman, G. 238  
 Stokes, I. 535  
 Stolcke, A. 978  
 Stone, P. 539  
 Stone, P. J. 777  
 Stone, P. T. 777  
 Stork, D. G. 868  
 Story, W. E. 128  
 Strachey, Ch. 239  
 Strat, T. M. 621  
 Striebel, C. T. 670, 671  
 Strohm, G. 535  
 Stuckey, P. J. 204, 206, 380  
 Stutz, J. 621, 865  
 Subramanian, D. 440, 814, 1102  
 Sugihara, K. 1019  
 Sugnet, C. 866  
 Sussman, G. J. 204, 378, 488  
 Sutherland, G. L. 55  
 Sutherland, I. 203  
 Sutton, R. S. 740, 897, 898, 899  
 Svartvik, J. 944  
 Svestka, P. 1065  
 Svetnik, V. B. 671  
 Swade, D. D. 46  
 Swartz, R. 1075  
 Swerling, P. 670  
 Swift, T. 380  
 Syrjänen, T. 439  
 Szathmáry, E. 177
- T**
- Tadepalli, P. 815  
 Tait, P. G. 128  
 Tamaki, H. 380, 978  
 Tambe, M. 814  
 Tanca, L. 379  
 Tank, D. W. 43  
 Tarjan, R. E. 1111  
 Tarski, A. 39, 374, 945  
 Tash, J. K. 741  
 Tate, 488, 508, 535  
 Tate, A. 536  
 Tatman, J. A. 741  
 Taylor, C. 1019  
 Taylor, W. 40, 292  
 Teller, A. 177, 619  
 Teller, E. 177, 619  
 Temperley, D. 944  
 Tennenholz, M. 897
- Tesauro, G. 232, 239, 891, 898  
 Thagard, P. 63  
 Thaler, R. 702  
 Thielscher, M. 436  
 Thitimajshima, P. 620  
 Thomas, A. 864  
 Thomason, R. H. 945  
 Thompson, D. W. 1011  
 Thompson, H. 980  
 Thompson, K. 231, 237  
 Thompson, R. H. 946  
 Throop, T. A. 233  
 Thrun, S. 1058, 1063, 1064  
 Tibshirani, R. 865, 868  
 Tinsley, M. 231  
 Tirole, J. 742  
 Titterington, D. M. 865  
 Toffler, A. 1088  
 Tomasi, C. 999  
 Torrance, M. C. 206  
 Tomas, C. 537  
 Torres y Quevedo, L. 236  
 Touretzky, D. S. 439  
 Traverso, P. 490  
 Troyanskii, P. 980  
 Trucco, E. 1020  
 Tsang, E. 206  
 Tsitsiklis, J. N. 740, 898, 899, 1111  
 Turner, K. 742  
 Tung, F. 670, 671  
 Turcotte, M. 812  
 Turing, A. 28, 32, 39, 45, 49, 50,  
 63, 210, 236, 336, 377, 618, 865,  
 896, 972, 1074, 1076, 1093  
 Turtle, H. R. 979  
 Tversky, A. 32, 585, 685  
 Tyson, M. 980
- U**
- Ullman, J. D. 378, 379, 945, 1111  
 Ullman, S. 1018  
 Urquhart, A. 435  
 Uskov, A. V. 237  
 Utgoff, P. E. 791
- V**
- Vaessens, R. J. M. 536  
 Valiant, L. 778  
 van Beek, P. 204, 206  
 van Bethem, J. 435  
 van Doorn, A. J. 1018, 1019
- Van Eco, 943  
 Van Emden, M. H. 379, 439  
 Van Harmelen, F. 814  
 van Heijenoort, J. 381  
 Van Hentenryck, P. 204  
 van Nunen, J. A. E. E. 740  
 Van Roy, B. 898  
 Van Roy, P. L. 352, 356  
 van Run, P. 189, 206  
 Vapnik, V. N. 867, 868, 1020  
 Varaiya, P. 90, 899  
 Vardi, M. Y. 379, 438  
 Varian, H. R. 742  
 Vaucanson, J. 1063  
 Vazirani, U. 176  
 Vecchi, M. P. 177, 205  
 Veloso, M. 814  
 Vempala, S. 978  
 Vere, S. A. 535  
 Verma, T. 618  
 Verri, A. 1020  
 Vickrey, W. 738  
 Vigoda, E. 672  
 Vinge, V. 1091  
 Viola, P. 1020  
 Visser, W. 375  
 Vitanyi, P. M. B. 778  
 Volk, K. 865  
 von Helmholtz, H. 44  
 von Mises, R. 573  
 von Stengel, B. 742  
 Von Winterfeldt, D. 702  
 Voorhees, E. M. 980  
 Vossen, T. 490
- W**
- Wadsworth, C. P. 328  
 Waibel, A. 673  
 Waldinger, R. 380, 381, 488  
 Walker, H. 865  
 Wall, R. 945  
 Wallace, D. 981  
 Walras, L. 41  
 Walsh, M. J. 178  
 Walter, G. 865, 1063  
 Waltz, D. 53. 204, 865, 1006  
 Wang, E. 440  
 Wanner, E. 300  
 Warmuth, M. 128. 778  
 Warren, D. H. D. 352, 355, 379,  
 380, 488, 537, 944  
 Warren, D. S. 379

- Washington, G. 409  
 Wasow, T. 944  
 Watkins, C. J. 740, 897  
 Watson, J. 44  
 Watson, J. D. 163  
 Watt, J. 47  
 Wattenberg, M. 177  
 Weaver, W. 760, 978, 980  
 Weber, J. 63, 671, 946  
 Wefald, E. H. 175, 238, 1100  
 Wegbreit, B. 1064  
 Wegman, M. N. 381  
 Weidenbach, C. 381  
 Weiner, N. 980  
 Weinstein, S. 778  
 Weisler, S. 63  
 Weiss, G. 91, 539  
 Weiss, S. 778  
 Weiss, Y. 620  
 Weizenbaum, J. 1089, 1094  
 Weld, D. S. 91, 440, 488, 489,  
 490, 536, 537, 1090  
 Wellman, M. P. 41, 620, 622, 702,  
 741, 1065  
 Werbos, P. 741, 897  
 Wermuth, N. 619  
 Wertheimer, M. 1018  
 Wesley, L. P. 538  
 Wesley, M. A. 1064  
 Weymouth, T. 1064  
 Wheatstone, C. 1018  
 White, J. L. 375, 564  
 Whitehead, A. N. 49, 377, 795  
 Whiter, A. M. 535  
 Whitney, R. A. 945  
 Whorf, B. 300  
 Widrow, B. 53, 889, 896, 897  
 Wiener, N. 47, 210, 236, 670, 865  
 Wilber, B. M. 380  
 Wilcox, B. 239  
 Wilczek, F. 867  
 Wilensky, R. 56, 944, 945, 1086  
 Wilfong, G. T. 1063  
 Wilkins, D. E. 235, 535, 538  
 Wilks, Y. 945  
 Williams, B. 91, 440, 535, 536  
 Williams, R. J. 740, 897, 898  
 Williamson, M. 537  
 Wilson, A. 949  
 Wilson, R. A. 34, 63, 1094  
 Wilson, R. J. 203  
 Winker, S. 374, 381  
 Winograd, S. 53
- Winograd, T. 53, 56, 378, 980  
 Winston, P. H. 32, 53, 777, 787  
 Wirth, R. 815  
 Witten, I. H. 947, 979  
 Wittgenstein, L. 37291, 394, 435,  
 943  
 Wojciechowski, W. S. 375  
 Wojcik, A. S. 375  
 Wolf, A. 682  
 Wolpert, D. 742  
 Wong, A. 979  
 Wood, D. E. 175  
 Wood, M. K. 177  
 Woods, W. A. 438, 439, 945  
 Wooldridge, M. 91  
 Woolsey, K. 891  
 Wos, L. 374, 380, 381, 382  
 Wöhler, F. 1080  
 Wright, S. 28, 177, 618  
 Wrightson, G. 382  
 Wu, D. 946  
 Wundt, W. 44  
 Wygant, R. M. 378
- Y**
- Yakimovsky, Y. 703  
 Yamada, K. 976, 981  
 Yan, D. 535  
 Yang, C. S. 979  
 Yang, Q. 490, 536  
 Yannakakis, M. 178, 205  
 Yap, R. H. C. 380  
 Yedidja, J. 620  
 Yip, K. M.-K. 440  
 Yngve, V. 944  
 Yoshikawa, T. 1065  
 Young, R. M. 129, 536  
 Young, S. J. 978  
 Young, T. 990  
 Younger, D. H. 944  
 Yung, M. 180  
 Yvanovich, M. 536
- Z**
- Zadeh, L. A. 621  
 Zaidel, M. 944  
 Zapp, A. 1020  
 Zaritskii, V. S. 671  
 Zelle, J. 941, 947  
 Zermelo, E. 236, 742  
 Zhai, C. 979

# TÁRGY MUTATÓ

3-SAT 185, 347, 383, 1107  
8-as kirakójáték 103  
15-ös kirakójáték 128, 174  
 $\epsilon$ -elfogadható 175  
 $\epsilon$ -gömb 771  
 $X^2$  metszés 764  
#P-nehéz 597

## A, Á

a legjobb túlélése 672  
a priori tudás 782  
A\* dekódoló 668  
ABC komputer 46  
ABSOLVER 151  
ABSTRIPS 536  
absztraktió 101  
absztraktíós hierarchia 536  
AC-3 193, 204, 207  
AC-4 193, 204  
ACT 349  
ACT\* 814  
Ada programnyelv 46  
ADABoost 768, 769, 777  
adaline (Hebb tanulási módszerei) 5353  
adaptív dinamikus programozás 877, 895  
adat. hiányzó 765  
adatbányászat 59  
adatbázis  
deduktív 333  
adathalmaz  
indító 777  
adatkomplexitás 347, 379  
adattársítás problémája 671  
additív értékfüggvény 690  
ADL. lásd cselekvésleíró nyelv

Advice Taker 51, 56  
AFSM lásd kiterjesztett véges automata  
ágens 35, 66, 89  
áramkörön alapuló 283  
architektúra 79  
autonóm 72  
célorientált 84, 90  
egyszerű reflexszerű 81, 90  
értékfüggvény 684  
érzetkelőmentes tervkészítő 510  
felfelejtéses tervkészítő 511  
folytonos tervkészítő 511, 524, 525, 535, 538  
hasznossággalapú 874  
hasznosságorientált 86, 90  
intelligens 23, 62  
kockázatkereső 683  
kockázatkerülő 683  
kockázatsemleges 684  
kritikus 87  
modellalapú 84  
modellalapú reflexszerű 90  
móh 882  
problémageneráló 88  
problémamegoldó 97  
racionális 35, 66, 68, 70, 89  
reflexszerű 874  
sorrendezett  
hasznosságfüggvény 684  
szoftverágens 75  
tervezés 727  
tanuló 86  
tanuló elem 87  
végrehajtó elem 87  
újratervező 511, 520  
végrehajtás monitorozó 520  
ágensfüggvény 67, 89  
ágensprogram 67, 79, 90  
korutin 79  
agglomeratív klaszterezés 964  
aggregáció 498  
AIBO robot 1028, 1061  
akármikor algoritmus 1099  
AKO lásd aszimptotikus korlátozott optimalitás  
aktivációs függvény 845  
akusztikai modell 660, 663, 664, 669, 937  
alakillesztés 862  
alakzatkontextus 1011  
alap rezoluciós tételek 272, 365  
alapértelmezett  
elmélet kiterjesztése 430  
érték 422  
alapliterál 453  
alapterm 307, 334  
ALFA-BÉTA-KERESÉS 219, 222  
algoritmus 39  
belsı-kılsı 957  
csempézés 857  
egyesítési fa 597  
Markov lánc Monte Carlo 604  
minimax 213  
változó eliminációs 594  
ALICE 1074  
ALIGN 1014  
ALisp 1058  
alkategória 923  
alkategória-lista 923  
állandó időkülönbségű simítás 637  
állapot 103, 406  
aktuális 154  
belekéneszterítés 124  
dinamikus 1050  
generálás 109  
hiedelmi 125, 229, 516, 549, 721, 1029

- kezdeti 103  
kifejtése 109  
legvalószínűbb 1046  
megkötés 456  
tulajdonságai 220
- ÁLLAPOTÁTMENET** 514  
állapotátmenet-mátrix 626, 640, 650  
állapotátmenet-modell 629, 634, 640, 675, 741, 875, 884  
állapotátmenet-függvény 100, 103, 185, 211, 914  
**ÁLLAPOTÁTMENET-Fv** 100  
**ÁLLAPOT-FRISÍTÉS** 84  
állapotkorlátozás 483  
állapotmegfogalmazás  
teljes 155
- állapottér**  
átmérője 117  
biztonságosan feltárható 168  
felszín 154  
metaszintű 147  
objektumszintű 147  
útja 100
- ÁLLÍT** 431  
állítás 550  
Almanac Game 703  
ALPAC 980  
általános problémamegoldó 34, 50  
általános robot nyelv (GRL) 1057  
általánosítás 752  
általánosított Modus Ponens 337  
alulgenerál 913  
Alvey-jelentés 57  
ALVINN 60  
AM (Automated Mathematician)  
816  
amőba 231, 236, 240  
anafora 937  
analitikus általánosítás 814  
Analitikus Gép (Babbage) 46  
analízis 909  
analógiás következtetés 814  
**ANALOGY** 52, 64  
angol árverés 737  
antoníma 944  
anyag 395  
Apolló program 945  
architektúra  
alárendeli 1055  
háromréteges 1057  
hibrid 1055  
szoftver 1055
- ARGOK** 339  
**Arisztotelész**  
szillogisztika 34  
tervkészítő algoritmus 27  
ámyalás 996, 1002  
**ARPAbet** fonetikus ábécé 661, 663  
árverés  
angol 737  
első áras zárt ajánlatú 738  
Vickrey-féle 738  
zárt ajánlatú 738  
zárt ajánlatú második áras 738  
**Asimo humanoid robot** 1024  
**ASPEN** 535  
**Association for Computational  
Linguistics** 947  
aszimptotikus elemzés 1106  
aszimptotikus korlátozott  
optimalitás (AKO) 1102  
asszociáció jobbra 936  
asszociatív memória 867  
átírás 372  
átíró szabályok 908  
átlagjutalom 712  
ámenet-valószínűség 605  
**ATMS** lásd feltételezésalapú igazság-karban tartó rendszer  
atomi mondat 258, 306  
attribútum  
folytonos értékkészletű  
kimeneti 766  
folytonos és egészértékű  
bemeneti 765  
irreleváns 763  
legfontosabb 757  
nyelvtan 944  
sokértékű 765  
áttünés 1011  
**AURA** 374, 375, 381  
automata  
automatikus összeszerelés 107  
autonóm tengeralattjáró (AUV)  
1023  
AUV lásd autonóm tengeralattjáró  
axióma 313  
cselekvéskiszáró 453  
egyedi cselekvés 401  
egyedi elnevezések 401  
egyedi füzér 411  
előfeltétel 482  
hasznátható 372  
hatás 398  
keret 399
- kínai szoba 1085  
Kolmogorov-féle 555  
követő állapot 400  
lehetőségi 398  
Peano 314, 345  
**STRIPS** 487  
azonosságelmélet 1093  
azonossági bizonytalanság 609
- B**
- Babbage, Ch.**  
Analitikus Gépe 46  
Differenciál Gépe 28  
**backgammon** 210  
**Backus–Naur-forma (BNF)** 1112  
**Baldwin-hatás** 177  
bal-sarok elemzők 919  
bang–bang szabályozás 891  
**BASEBALL** kérdésválaszoló rendszer  
945  
bátoriségi faktor 850, 879  
**Baum–Welch-algoritmus** 864  
Bayes–háld 578, 617, 751, 975  
dinamikus 627, 650, 724, 1029  
hálóstruktúrák tanulása 830  
hibrid 588, 617  
következetítés 592  
paramétertanulás 827  
tanulás rejtett változókkal 835  
**Bayes-i** modellösszevonás 957  
**Bayes-modell**  
naiv 825, 863  
Bayes–Nash-egyenlőség 736  
Bayes–osztályozó 567  
Bayes–szabály 40, 564  
Bayes–tanulás 769, 819, 863  
Bayes–tételek 564  
bázisfüggvény 888  
bázisvonal 1000  
**BDD** lásd bináris döntési diagram  
Beard úr 683  
beavatkozó 73, 1023  
szerv 66  
**Bécsi Kör** 37  
becslés, konzisztens 600  
beépítés 910  
behatalás 174  
behaviorista 44  
**Belle**, program 237  
**Bellman-egyenlet** 714  
**Bellman-frissítés** 715

- belső állapot 83  
 bemeneti rezolúció 370  
**BENÍQ** 439  
 bennefoglalás 371  
 bennefoglalási háló 341  
 beszédfelismerés 659, 909  
 beszédhang 660  
   ~modell 664  
 beszédmegértés 659  
 beszédrész 954  
 beszélő 905  
 beszélőazonosítás 663  
 béta-eloszlás 828  
**BETOLT** 340, 383  
 betűrejtvény 186  
 bigram modell 660, 666  
 billentés 1027  
 bináris döntési diagram (BDD) 489, 518, 537  
 biológiai naturalizmus 1081  
 biometrikus azonosítás 1007  
 bizonyítás 266  
   nem konstruktív 364  
 bizonyítás-ellenőrző 374  
 bizonyosságfüggvény 614  
 bizonyosságháló 578  
 bizonyossági állapot 632  
 bizonyossági tényező 56, 614  
 bizonyosságterjesztés 620  
 bizonytalan azonosság 1097  
 bizonytalanság 545, 614  
 bizonytalanság melletti  
   optimizmus 172  
 biztosítási prémium 684  
**BKG** ostáblaprogram 239  
**BLACKBOX** 486, 489  
**BNF** *lásd* Backus-Naur-forma  
 boldog görbe 762  
 Boltzmann-gép 868  
 bolygójáró 1023  
 bonyolultságelmélet 1106  
**Boole**  
   CSP 185  
   kulesszó-modell 958  
   logika 39, 291  
   véletlen változók 551  
**bootstrap** 777  
**Boyer-Moore-tételbizonyító** 374  
 Bridge Baron 232, 233  
 bridzs kártyajáték 228, 232  
 British National Corpus 979  
 Brown-korpusz 979  
 Bugs 620
- BUILD** 440  
 büntetés 89
- C**
- cáfólásteljes rezolúció 359, 365  
 Canby-féle éldetektor 994  
**CARMEL** 1064  
 Cassandra 537  
 Caterpillar-angol 970  
**C-BURIDAN** 537  
 cél 84, 97, 312  
   célorientált következtetés 275  
   ~függvény 47, 154  
   hatásos 798  
   ~megfogalmazás 98  
   predikátum 754  
   predikátum, definíció jelölje 783  
   ~teszt 100, 103, 185, 914  
   újraformálás 510  
 celladekompozíció 1039, 1042  
   egzakt 1043  
 cellaelrendezés 107  
**CÉL-TESZT** 167  
**CES** 1058  
**CFG** *lásd* környezetfüggetlen nyelvtanok  
**CGP** *lásd* Conformant Graphplan  
**CHESS** 4.5 129  
   4.6 237  
**CHILL** 941, 947  
 Chinook dámaprogram 231, 239, 243  
 Chomsky normál alak 957  
**cHUGIN** 619  
 Church-Turing-tézis 39  
**Cigol** 815  
 ciklikusság-vágóhalmaz 200  
**CLAM** *lásd* Korlátozott Logikai Absztrakt Gép  
   Clark normál forma 425  
   Clark-lezárás 425, 426  
   CLASSIC 423, 424, 439  
   CLINT 815  
   CLIPS 378  
   CLP 358  
   CMAC 897  
   CMU Rover projekt 1063  
   CN2 815  
   CNF *lásd* konjunktív normál forma  
   CNLP 537
- COLBERT** 1065  
 Colossus 45  
 Conformant Graphplan (CGP) 537  
 CONTINUOUS-POP-AGENT 538  
**CONVINCE** 618  
 Core Language Engine 944  
 Croton Egg Farm 704  
**CWA** *lásd* zárt világ feltételezés 425, 429  
**CYC** projekt 435, 437  
**CYPRESS** 538
- Cs**
- CSATOL** 352, 355  
 csavarás 1027  
 cselekvés 100, 167  
   alkalmazható 454  
   dekompozíció 500  
   egyszerű 501  
   együttes 531  
   előfeltétel 454  
   implicit hatása 401  
   konkurens 531  
   következmény 454  
   lépésköltsége 101  
   megőrző 473  
   mozgáster 496  
   tudás-előfeltétel 413  
   tudáshatás 413  
 cselekvésérték 874  
 cselekvéshasznosság tábla 692  
 cselekvési séma 453  
 cselekvéskizároló axióma 482  
 cselekvéslefrő nyelv (ADL) 455, 487, 531  
 cselekvő komponens 749  
 csempézés algoritmus 857  
**Csernobil** 1059  
 csomagoló program 418  
 csomópont 579, 916  
   determinisztikus 587  
   döntési 691  
   előcsomóponja 120  
   klaszter 597  
   konziszenciája 193  
   leszármazottja 586  
   szivárgás 587  
   szülfje 579  
   véletlen 224, 691  
**CSP** *lásd* kényszerkielégítési probléma

- csukló  
rotációs 1027  
transzlációs 1027  
csúsztatós kirakójáték 104
- D**
- da Vinci Surgical Systems (sebész-robotok) 1060  
DAG *lásd* irányított, körmentes gráf 579  
DALTON 816  
dámagjáték 231  
DARKTHOUGHT 238  
DARPA fonetikus ábécé 61, 661, 672  
DART 61  
Dartmouth College 49  
Datalog tudásbázisok 343  
Davis–Putnam-algoritmus (DPPLL) 276, 291, 484  
DBH *lásd* dinamikus Bayes-háló  
DCG *lásd* definit klóz nyelvtan  
DDH *lásd* dinamikus döntési háló  
De Motu Animalium 27  
dedukciőelmélet 265  
deduktív  
adatbázis 378  
szintézis 375  
Deep Blue 60, 230  
Deep Space One 91 536  
Deep Thought 230, 238  
definíció 313  
definit klóz nyelvtan (DCG) 922, 943, 944, 951  
deformációs illeszkedés 1011  
deklarativ elfogultság 802  
dekódoló 1026  
deltaszabály 889  
d-elfálasztás 586  
demoduláció 369  
demodulátor 372, 384  
Dempster szabály 614  
Dempster–Shafer-elmélet 612, 614  
DENDRAL 55, 434  
DEVISER 535  
Dewey Decimális rendszer 390  
diagnosztikus szabályok 318  
diagramelemzések 919, 943  
Differenciál Gép (Babbage) 46  
differenciálegyenletek 1050  
differenciál hajtás 1028
- Digital Equipment Corporation (DEC) 57, 349  
dinamikai állapot 1027  
dinamikus Bayes-háló (DBH) 650  
dinamikus döntési háló (DDH) 724  
dinamikus logika 436  
dinamikus programozás 90, 129, 357, 497, 915  
nem folytatálagos 619  
dinamikusan stabil 1028  
diofantoszi egyenletek 203  
Diplomacy 215, 539  
direktíva 906  
diszjunkció 258  
diszkretizálás 588  
diszkriminációs háló 777  
diszparitás 999  
DNS 163  
dobáskényszer 233  
DOF *lásd* szabadságfokok  
dokumentumklaszterezés 963  
dokumentumosztályozás 963  
dominancia, sztochasztikus 687  
domináns stratégiák egyensúlya 729  
döntés  
egylépéses 678  
szekvenciális 678, 707  
döntéselemzés 698  
döntésemelélet 41, 549  
metakövetkeztetés 1100  
döntéshozó 698  
döntési fa 754  
legkisebb 757  
metszés 763  
döntési háló 578, 676, 691  
dinamikus 724  
döntési tönk 768  
DÖNTÉSI-FA-TANULAS 802  
DPPLL *lásd* Davis–Putnam algoritmus  
DRAGON 672  
drón 1060  
dualista elmélet 1081  
dualizmus (Descartes, R.) 37  
DVL 439  
DYNA architektúra 897
- E, É**
- Echelon-rendszer 1089  
EEG *lásd* elektroenzefalogramma (EGG) 43  
egyszintenció 597
- gráf 419  
kvantor 334  
példányosítás 334  
egyed 160  
egyedi cselekvés axióma 401  
egyedi elnevezések axiómája 401  
egyedi elnevezések feltételezés (UNA) 401, 425, 608  
egyedi fűzér axióma 411  
egyedítés 395  
egyedülálló szó 664  
egyenleti egyesítés 369  
egyenlőségszimbólum 311  
egyensúly 729, 879  
egyensúlyi táblaállás 222  
egyértelműsítés 417  
EGYESÍT 338  
egyesítés 333, 338  
legáltalánosabb 340  
egyesítési lépés 338  
egyeztetés 921  
egyretegű neurális háló 848  
egység 845  
egységes erőforrás azonosító 415  
egységlüggvény 393  
egységlő 277, 370  
egységprefériencia 370  
egységrezolúció 370  
egységtérjesztés 277  
**EGYSZERŰ-PROBLÉMA-MEGOLDÓ-**  
ÁGENS 99  
együttes  
tanulás 766  
terv 530  
ekvivalencia 259  
ekvivalens  
küverkezetés szempontjából 335  
El 916, 1004  
elágazási tényező 112  
effektív 149, 174  
elágazik- és-korlátoz 175  
élődetektálás 991  
elektroenzefalogramma (EEG) 43  
elemi esemény 551  
elemzés 910, 913  
fentről lefelé 913  
hatékony 915  
lentről felfelé 913  
El-HOZZAAD 916, 917  
ELIZA 1074, 1089  
eljárás  
csoportosító 597  
eljárásmód 538, 709

értékelés 718  
 iteráció 718  
 iteráció, aszinkron 720  
 iteráció, módosított 720  
 javítás 718  
 nem stacionárius 711  
 optimális 709  
 stacionárius 711  
 véges 712  
 vesztesége 717  
**élkonziszenciája** 192  
 fenntartása (MAC) 192, 196, 204  
 ellenőrzési játék 727  
 ellenőrző megfigyelés 632  
 ellenseg érv 168  
 elmélet  
     leíró 684  
     normatív 684  
 elméleti tudatlanság 547  
 elmosódott argumentum 411  
 eloszlás  
     béta 828  
     erősítők 176  
     feltételes Gauss 590  
     kevert Gauss 833  
     kevert 833  
     lineáris Gauss 589  
     logit 591  
     probit 591, 616  
     stacionárius 605, 634  
     standard normális 1110  
     szigmoid 619  
         többváltozós Gauss 644, 1110  
 elődállapot 461  
 előfeltétel  
     axióma 482  
     különböző 501  
     nyitott 466  
 előfordulási próba 340, 353  
 előjárószavak 911  
 ElőRE 633, 644, 722  
 előrecsatolt háló 846  
 előrefelé láncolás 273  
 előre-hátra algoritmus 636, 839  
 előrejelzés 632  
 előrenéző ellenőrzés 191  
 előrenyeresés 223  
 először-be-először-ki lásd FIFO  
 Első-ELEM 110  
 eltolássúly 845  
 ELUTASÍTÓ-MINTAVÉTELEZÉS 600  
 EM algoritmus 632, 669, 671, 832, 838, 839, 890, 956

strukturális 840  
 EMA\* 145  
 ember nélküli közúti jármű (ULV) 1023  
 ember nélküli légi jármű (UAV) 1023  
 emlékezet 300  
 empiricista 37  
 empirikus gradiens 893  
 ENIAC 46  
 Entscheidungsproblem 39  
 epifenomenális szereplő 1084  
 épizód 76  
 EQP 374  
 ERL-EL-KÉRDEZ 344, 345, 350  
 ERL-HL-KÉRDEZ 350, 351, 354  
 erőforrás 498  
     fogyóeszköz 498, 539  
     -kényszer 194  
 erőmérő 1026  
 erős MI 1073  
 erősítési tényező 1051  
 értékiteráció 713  
**ÉRTÉKITERÁCIÓ** 715, 717  
 ÉRTÉSÍT 521  
 érvényesség 264  
 érzékelés 66, 317  
     aktív 519, 984  
     automatikus 518  
     cselekvés 519, 694  
     sorozat 66  
 érzékelő 74, 984  
     modell 630  
 érzékenységvizsgálat 699  
 esemény 387, 591  
     általánosított 404  
     diszkrét 406  
     folytonos 406  
 eseménykalkulus 403  
 esetelapú következtetés 814  
 eshetőség 209  
 És-KERÉS 514  
 És-kiküszöbölés 265  
 És-párhuzamosság 356  
 És-VAGY gráf 274  
 És-VAGY-GRÁF-KERÉS 514, 541  
 észlelés 909  
 euklideszi távolság 842  
 Euler-gráf 178  
 Eurisko 816  
 evolúció 163  
 evolúciós stratégia 177  
 expanziófókusz 998

**F**

fa  
 dekompozíció 201  
 szélessége 202  
**FA-KERÉSÉS** 109, 111, 136, 140  
 faktorálás 269, 362  
**FASTFORWARD** 489  
**FASTUS** 967, 968, 980  
 fehérjetervezés 107  
 fekete doboz 183  
 feladat  
     előrevetítő 398  
     háló 488  
     környezet 72, 89  
     tervkészítési 398  
 felbonthatóság 680  
**FELDOLGOZ** 372  
 felfedezés 874, 883  
 felfedezési függvény 884  
 felfedező rendszerek 815  
 félíg előírható 336  
 feltétel–cselekvés szabály 81  
**FELSOROL-EGYÜTTES-KÉRDEZÉS** 561, 569, 592, 593  
 feltételes  
     függetlenség 617  
     feltételes lépés 512  
 feltételes valószínűségi tábla (FVT) 580, 581  
 feltételezés 433  
     egyedi név 608  
 feltételezésalapú igazság–  
 karbantartó rendszer (ATMS) 432, 440  
 feltétfeloldás 560  
 felügyelt tanulás 888  
 feltület  
     dőlése 995  
     Lambert-féle 989  
     lejtése 995  
     mintázat 1001  
 fennsík 156  
 fenomenológia 1079  
 fentől lefelé elemzés 913  
**FF (FASTFORWARD)** 489  
**FIFO** 113  
 Fifth Generation projekt lásd Japán  
 Ötödik Generációs projekt  
**FILTER** 372  
**FIND-TRANSFORM** 1014  
 finom mozgások tervezése (FMT) 1047

- fitness**  
 függvény 161  
 tájfelszín 177
- fix pont** 345
- fixálás** 1000
- fizikai szimbólumrendszer** hipotézis 50
- fMRI** 43
- FMT** *lásd* fibrom mozgások tervezése
- FOCUS** 815
- fogalomgenerátor** (Lull, R.) 27
- foglalási hálózat** 1063
- fogolydilemma** 728, 742
- FoII** 806, 807, 815, 817
- fókuszterek** 940
- folyamat** 406
- folyó**  
 esemény 397  
 esemény kalkulus 402, 407
- folytatás** 355
- folytonos beszéd** 666
- FOLYTONOS-RRT-ÁGENS** 525, 528
- folytonosság** 680
- fonéma** 660
- fonológia** 660
- FORBIN** 535
- fordítás**  
 előre szerkesztett 970  
 irodalmi 970  
 kordátozott forrás 970  
 memóriaalapú 972  
 nyers 970
- fordítási modell** 973
- forgatás** 1027
- formális**  
 nyelv 906  
 formális nyelvtan 911
- photometria** 988
- főnév**  
 megszámlálható 395  
 nem megszámlálható 395
- főnévi kifejezés** 907
- FREDDY** 107, 536, 1064
- Fredkin-díj** 238
- FRITZ** 231
- FRUMP** 980
- FSA** *lásd* véges állapotú automata
- FUF** generáló rendszer 945
- funcionális**  
 állapotazonosság elmélet 1093  
 tífüggyőség 799  
 mágneses rezonancia 43
- specifikáció elmélete** 1093
- functionalizmus** 1081
- fuzzy**  
 halmozelmélet 615  
 logika 547, 612, 615  
 szabályozás 616
- független részproblémák** 199
- függetlenség** 562, 566  
 abszolút 562  
 feltételes 566  
 marginális 562
- függőségi nyelvtan** 944
- függőségevezérelt visszalépés** 205
- függvény** 299  
 approximáció 887  
 egyszerű 608  
 komplex 608  
 -mentes literál 453  
 súlyozott lineáris 221  
 -szimbólumok 304
- FVT** *lásd* feltételes valószínűségi tábla
- G, Gy**
- Gamma függvény** 869
- GARI** 535
- Gacschning-heurisztika** 180
- Gauss-eloszlás** 1110  
 -ok keveréke 662
- Gauss-hibamodell** 652
- Gauss-Jordan-elimináció** 1108
- generáloképesség** 908
- genetikus**  
 algoritmus (GA) 54, 136, 160, 177  
 genetikus programozás, 177
- GENETIKUS-ALGORITMUS** 162
- geometriai tételebizonyító** 50, 380
- gépi**  
 evolúció 54  
 gépi fordítás 951  
 gépi látás 33
- GIB** 233
- Gibbs-mjntavételező** 606, 619
- Gittins-index** 884, 897
- GLAUBER** 816
- globális helymeghatározó**  
 rendszer (GPS) 34, 38, 50, 128, 487, 651, 1026  
 differenciális 1026
- GLR robot** 1065
- gó** (játék) 232
- Go4++** (játékprogram) 232
- Goemate program** 232
- GOFAL** 1077
- Göldbach-sejtés** 816
- GOLEM** 815
- GOLOG** 436, 1058
- Go-Moku** 231
- Good-Turing-similitás** 978
- Gödel nemteljességi tétele** 39, 1076
- Gödel-szám** 367
- görbületi diszkontinuitás** 1005
- GPS** *lásd* globális helymeghatározó rendszer  
 gradiens 164  
 empirikus 165
- gráf**  
 ÉS-VAGY 514  
 irányított, körmentes 579  
 soros tervkészítési 475  
 színezés 203
- GRÁF-BŐVÍTÉS** 476
- GRÁF-KERESÉS** 122, 136, 140, 218, 541
- gráfos modell** 578, 622
- Graph Traverser** 175
- GRAPHPLAN** 472, 476, 486, 537
- érzékelés** 537
- GRL** *lásd* általános robot nyelv
- Guiness Rekordok Könyve** 703
- Gus** 980
- gyakorlati tudatlanság** 547
- gyenge**  
 MI 1073  
 módszer 55
- győztes** 243
- H**
- ha-akkor szabály** 81
- Hacker** 488
- hallgató** 905
- halmaz** 315
- halmaz-szint** 475
- háló**  
 CPSC 588  
 egyszeresen összekötött 597  
 többszörösen összekötött 597
- hamis**  
 negatív 784  
 pozitív 784
- Hamming-távolság** 842

- Hansard korpusz 973, 983  
 hármashangzó 663  
 háromszög 358  
 háromszög egyenlőtlenség 140  
**HARPY** 176, 672  
 hash-tábla 341  
     kulcs 341  
 használható axióma 372  
**HASZNOSÁG** 213  
 hasznosság 86, 549  
     elv 681  
     normált 685  
 hasznosságbecslés, közvetlen 876, 895  
 hasznosságelmélet 549  
     többattribútumú 686  
 hasznosságfüggelenség 690  
     kölcsönös 690  
 hasznosságfüggvény 86, 211, 676, 751  
 határozott klóz 272  
 határvonal 142  
 hatásaxióma 398  
 hatásdiagram 618, 676, 691  
 hatásháló 578  
**HÁTRAFELÉ** 635  
 hátrafelé láncolás 273  
 háttérítudás 249  
 háztömb-távolság 148  
**HEARSAY-II** 672  
 hearts 228  
 Hebb-tanulás 48  
 hedonisztikus kalkulus 702  
 begyeric 156  
**HEGYMÁSZÁS** 155  
 Helpmate robot 1059, 1060  
 helyes következetés 256  
 helyesírás-javító 963  
**HELYETTESÍTÉS** 334, 337  
 helyettesítés 334  
     lista 312  
 helyettesíthetőség 680  
 helyezett automata 1065  
 helymeghatározás 1030  
     globális 1030  
     Markov 1064  
 Hempel igazoló elmélete 37  
**heptikus** visszacsatolás 1065  
 Herbrand-bázis 366  
**Herbrand-tétel** 345, 365, 366  
**Herbrand-univerzum** 365  
 Hesse mátrix 165  
 Heuristic Programming Project 56
- Heuristic Search Planner (HSP)** 489  
**heurisztika** 452  
     domináló 150  
     elfogadható 139, 462  
     fokszám 190  
     konziszenciája 140  
     konzisztens 150  
     leginkább megkötött változó 500  
     legjobban-korlátozott-változó 472  
     legkevésbé-korlátozó-érték 190  
     legkevesebb fennmaradó érték 190  
     legkorlátozottabb változó 346  
     minimális tartalék 499  
     monotonitása 140  
     nulla lépés 230, 238  
**heuristikus függvény** 137  
 hexapod 1053  
**HFH** lásd hierarchikus feldatháló  
 hiányos információ 210  
 hiba  
     átmeneti 652  
     függvény 1111  
     visszaterjesztés 853, 863  
 hibrid architektúra 1099  
 hiedelem  
     állapot 125, 229, 516, 549, 721, 1029  
     frissítés 431  
     mértéke 547  
     révízió 431  
**hierarchia**  
     taxonomikus 390  
**hierarchikus**  
     dékompozíció 500  
     feladatháló (HFH) 500, 536, 538  
     megerősítéses tanulás 1098  
     struktúra 1098  
 hipera szélesség 205  
**HipNav** 61  
 hipotézis 752, 818  
     általanostás 785  
     konzisztens 752  
     közelítőleg helyes 771  
     pillanatnyilag-legjobb 777, 784  
     prior 819  
     szűkítés 785  
     túlzott 814  
     valószínűségi értelemben  
     közelítőleg helyes 770  
 hipotézistér 752
- kifejezőképessége 753  
**HITECH** 237  
**Hív** 355  
 hivatkozó szöveg 415  
**HMM** lásd rejtett Markov-modell  
 holonomikus 1028  
 homeostatikus 866  
 homo sapiens 31  
 homofón 660  
**Hopfield-háló** 867  
**Hopkins Beast** 1063  
 horizont  
     -probléma 223  
     véges 711  
     végzetlen 711  
**Horn-klóz** 272  
**HOZZÁRENDELTELÉN-VÁLTOZÓ-KIVÁLASZTÁS** 188, 190  
**HSCP** 537  
**HSP** lásd **Heuristic Search Planner** 489  
**HSTS** 535  
**HUGIN** 618, 671  
 hurkos terjesztés 620  
 huzalozás 107
- I**
- IBM Model 3. 974, 982, 983  
 IBM RS/6000 230  
**ID3** 815  
 idő és ideidő 928  
 időbeli különbség 879, 896  
 időigény 111  
 időrendi visszalépés 195  
**IE** rendszer  
     attribútumalapú 967  
**IEEE P1600.1** 435  
 igazolás 428, 432  
 igazoló elmélet (Hempel) 37  
 igazságfüggvény 612, 617  
 igazság-karbantartó rendszer (TMS) 205, 431, 440  
     feltételezésselapú 432  
     igazolásalapú (JTMS) 432, 440  
 igazságírás 260  
 igazságítartó következetés 256  
 igei kifejezés 907  
 igék alkategóriákba osztása 923  
**ILP** lásd induktív logikai programozás  
**IMA\*** 144

- impasszolás 233  
 implementációs szint 250  
 implikáció 258  
 indexelés 340  
   predikátum 340  
 indikátorváltozó 851  
 indirekt szólásaktus 906  
 indító 777  
 indukált szélesség 205  
 indukció 37, 751  
 induktív tanulás 941  
 induktív következtetés  
   alapproblémája 752  
   tiszta 751  
 induktív logika 556, 573  
 induktív logikai programozás (ILP) 794, 803, 815, 941  
 információ 760  
   érték elmelet 694  
   gyűjtés 71  
   gyűjtés, rövidlátó 697  
   keresés 864, 951, 957  
   kinyerés 951, 966  
   nyereség 761  
   teljes, értéke 695  
 ingerszegénység 946  
**INKONZISZENTS-ÉRTÉKEKET-KIVESZ**  
 J93  
 integrálhatóság 1002  
 intelligens ágens 62  
 intelligens tudásalapú rendszerek 57  
 intencionális állapot 1082  
 intencionalitás 1079  
 International Phonetic Alphabet (IPA) 661  
**INTERPLAN** 488  
 interpretáció 304  
 invertált  
   index 965  
   invertált inga 891  
**IPA** *lásd* International Phonetic Alphabet  
**IPEM** 538  
**IPL** listakezelő nyelv 49  
**IPP** grátervező rendszer 489  
 irányításelmélet 47, 1051  
 irányított könnentes gráf (DAG) 579  
**IR**-rendszer  
   bool kulcsszó modell 958  
   felidézés 961  
   Pontosság 961  
   szózsák modell 960  
 ISA-kapcsolat 439  
**Isis** 535  
 ismeretelméleti megállapítás 301  
 ismerethány 612, 614  
**ISS** Nemzetközi űrállomás 1060  
 itéletkalkulus 248, 258  
 itéletlogikai attitűd 410, 1082  
 itéletszimbólum 258  
**ITEP** sakkkprogram 237  
**ITERATÍVAN-MÉLYÜLŐ-KERESÉS**  
 118, 149  
**ITOU** 815  
**IXTET** 489, 535
- J**
- Jacquard szövészéke 1063  
 Japán Ötödik Generációs projekt 57, 380  
 játék 41  
   determinisztikus ekvivalens 683  
   HEX 239  
   ismétlődő 734  
   játékfa 212  
   kártyajáték 228  
   kétszemélyes 209  
   kimenetele 728  
   koordinációs 730  
   megoldása 728  
   részleges információjú 735  
   természet elleni 513  
   zérusösszegű 731  
   zérusösszegű teljes  
   információjú 209  
 játékelmélet 41, 209  
 játékos 727  
 játékprogramok 229–233  
   bridzs 232  
   dáma 231  
   gó 232  
   jegy vektor 662  
   osttábla 232  
   Othello 232  
   akkord 230  
   jelenet 986  
   jellemző 153  
   jelitudomány 943  
   jelzések 943  
   jobbra asszociáció 936  
**JTMS** *lásd* Igazság-karbantartó rendszer, igazolásalapú jutalom 89, 708, 873, 1098  
 additív 711  
 alakítás 898  
 átlag 712  
  ~függvény 875  
 leszámlított 711  
  ~mátrix 727
- K**
- Kaissa program 237  
 kalah (játék) 237  
 Kalman-erősítésmátrix 647  
 Kalman-szűrő 627, 643, 1029, 1063  
 váltó 650, 674  
 kanonikus eloszlás 587  
 kapcsolati  
   bizonytalanság 609  
   módszer 378  
 karakterfüzérek 906  
 kártyajáték 228  
 kaszkádositott véges állapotú átalakító 968  
 katafora 937  
 kategória 390  
   anyag 396  
   diszjunkt 391  
   dolog 396  
 kategorikus nyelvtan 944  
 kauzális háló 865  
 k-DF (döntési fa) 773  
 k-DL (döntési lista) 773  
 kényszer 183  
   abszolút 186  
   -álapú általánosítás 814  
 bináris 186  
 egyenlőtlenségi 471  
 feljegyzés 205  
 hipergráf 186  
 lineáris 185  
 nemlineáris 186  
 nyelv 185  
 összekapcsolási 1040  
 preferencia 187  
 tanulás 197  
 terjesztése 191  
 unáris 186  
 kényszerítés 510  
 kényszerkezési probléma (CSP) 183, 189, 551  
 Boole 185

- folytonos tartományú 186  
 inkrementális megfogalmazás 185  
 kényszergráfja 184  
 kényszerlogikai-programozás 204  
 kép 986  
 képpont 986  
 képsegmentálás 994  
 képzési szabály 1112  
 KÉRDEZ 340  
 keresés 85, 99  
     A\* 139, 174, 175  
     alfa-béta 216, 230  
     B\* 238  
     dinamikus súlyozási technika 174  
     DTA\* 175  
     egyenletes költségű 114  
     egyensúlyi 222  
     egyszerűsített 145  
     elleniségek melletti 209  
     elsőnek-választott 157  
     EMA\* 175  
     gradiens leereszkedés 158  
     hegymászó 155, 170  
     heurisztikus 112, 174  
     heurisztikus útalgortítmus 179  
     IK 175  
     IMA\* 175  
     informált 97, 112, 136  
     iteratívan kifejti 175  
     iteratívan megnyíló 119, 132  
     iteratívan mélyülő 117, 144  
     keresési költség 112  
     kétirányú 119, 491  
     konspirációs szám 238  
     legjobbat-először 136  
     legmeredekebb emelkedő 155  
     lokális 136, 154  
     lokális nyaláb 159  
     mélységi 115  
     mélységekkel korlátozott 117  
     memoriakorlátozott 145, 175  
     mobó 137, 155  
     MTD(f) 238  
     nem informált 97, 112  
     nyaláb 159, 176  
     offline 166  
     online 136, 166  
     optimális 154, 174  
     összköltsége 112  
     párhuzamos 179  
     rekurzív legjobbat-először 144  
         RLEK 238  
         SSS\* 238  
         STAGE 176  
         szélességi 112  
         sztochasztikus 157  
         sztochasztikus nyaláb 160  
         tabukeresés 176  
         tanuló valós idejű 171  
         teljes 154  
         TRTA\* 178, 181  
         vak 112  
         valós idejű 178  
         véletlennél újraindítású 157  
         visszalépések 115, 188  
     keresési csomópont 109  
     keresési fa 108  
     keresési stratégia 109  
     keresési tér  
         elágazási tényezője 798  
     keresztezés 161  
     keresztkorreláció 997  
     keresztsvalidáció 765  
         hagy-jí-egyet 765  
     keret 56, 438, 662  
     keretaxióma 399  
     keretproblémá 399  
         következetetői 400  
         reprezentációs 400, 454  
     kernel 863  
     kemelesített algoritmus 860  
     kernelfüggvény 843  
     kernelmodell 843  
     kétkamerás térbeli látás 999  
     keverési idő 634  
     kevert eloszlás 833  
     kevert Gauss-eloszlás 833  
     kezdeti állapot 914  
     kezdőszimbólum 1112  
     kezelhetetlenség 39  
     kézfrás-felismerés 1007  
     kéz-szem gép 1064  
     KHF *lásd* kölesönben hasznosság-független  
     kialakuló viselkedés 533  
     kiátlagolás 560  
     kibontakozó viselkedés 1054  
     KIDS 381  
     kiejtési modell 664  
     kielegíthetőség 265  
         vizsgálata 480  
     kielégítő döntéshozatal 41  
     kiemelés 338  
     KiÉRTÉKEL 220, 222  
         KiÉRTÉKEL függvény 136, 210, 219,  
         887  
         kifejezés 907  
         kifejezsstruktúra 907, 943  
         KIFERT 111  
         kiinduló állapot 100, 185, 211  
         KIJELENTE 340  
         kjelentés 312  
         kimerítő felosztás 391  
         kínai szoba 1085  
         kinematika 1040  
             inverz 1040  
         kisbetű-nagybetű konverzió 962  
         kísérlet 875  
         kiszámítható függvény 39  
         kiterjesztés 922  
             lemma 365, 367  
         kiterjesztett  
             interpretáció 307  
             Kalman-szűrő 649, 1033  
             nyelvtanok 921  
             nyelvtanok generálóképessége 925  
             véges automata 1056  
         kiterjesztett véges automata (AFSM)  
         1054, 1056  
         kitermelés 883  
         k-konziszenciencia 193  
         k-közép klaszterezés 964  
         KKSZ *lásd* kiterjesztett Kalman-szűrő  
         KL-ONE 439  
         klóz  
             elsőrendű határozott 342  
             Horn 272  
             itétellogikai határozott 342  
             törzse 351  
         KO *lásd* korlátozott optimalitás  
         koartikuláció 664  
         koartikulációs hatás 663  
         kockavilág 458  
         kognitív  
             architektúra 349  
             pszichológia 44  
             robotika 436  
             tudomány 34, 45  
         koherenciarelációk 939  
         koherens szöveg struktúrája 939  
         Kolmogorov-féle axiomá 555  
         Kolmogorov-komplexitás 778  
         komunikáció 77, 298, 529, 533  
         905  
         lépések 909

- létrehozás 909  
 szándék 909  
 szintézis 909  
 kommutativitás 187  
 kompetitív arány 167  
 komplexitás 41
  - algoritmikus 778
  - Kolmogorov 778
 kompozíció 352  
**KOMPOZÍCIÓ** 352  
 kompozíciós
  - képesség 298
  - szemantika 926
 konfigurációs tér 1039, 1040
  - váza, 1044
 konfliktushalmaz 195  
 konfliktusvezérelt visszaugrás 196  
 konjugált prior 828  
 konjunkció 258  
 konjunkt sorrendezés 346  
 konjunktív lekérdezés 379  
 konjunktív normál forma (CNF) 269, 360  
 konneksionizmus 57, 844  
 kontúr 996  
 konvolúció 991  
**KONZISZTENS-MEGH?** 801  
 koordináció 529, 530  
**Korlátozott Logikai Absztrakt Gép (CLAM)** 380  
 korlátozott logikai programozás 358  
 korlátozott optimalitás (KO) 1101, 1102  
 korpusz 951  
 kostansszimbólumok 303  
**Kotek–McCarthy sakkprogram** 237  
 kovarianciamátrix 1111  
 Kowalski-forma 360  
 kölcsönös kizárási 473  
 kölcsönösen hasznosságfüggetlen (KHF) 690  
 kölcsönösen preferenciálisan független (KPF) 689  
 környezet 66
  - determinisztikus 75, 100
  - dinamikus 76, 1024
  - diszkrét 76, 99
  - egyágenses 77, 410, 529
  - együttműködő 529
  - epizódszerű 76
  - folytonos 76, 1024
  - generátor 78
 hozzáférhető 75  
 kooperatív 77, 209  
 megfigyelhető 75, 99  
 nem epizódszerű 75  
 nemdeterminisztikus 75  
 osztály 78  
 részlegesen megfigyelhető 707, 711, 720, 735, 740, 747, 1024  
 sorozatszerű 76  
 statikus 76, 99, 530  
 szemidinamikus 76  
 sziochasztikus 45, 1024  
 teljesen megfigyelhető 708, 720, 747, 837, 874  
 többágenses 77, 209  
 versengő 77, 529
  - verseny 209
 környezetfüggetlen nyelvtan (CFG) 943, 951  
 környezeti történet 708  
 köríllírás 429
  - prioritásos 430
 köteg 392  
 kötőszök 911  
 követés 1030  
 következmény
  - belső 503
  - diszjunktív 512, 540
  - elsődleges 502
  - feltételek 512
  - külső 501
  - másodlagos 502
 következő 385  
**KÖVETKEZET** 372  
 következetetés
  - alapértelmezésekben alapuló 611
  - automatizált 33
  - célvezérelt 235
  - helyes 345
  - inverz kapcsolatok 421
  - szabályalapú 612
  - szimbolikus valószínűségi 618
  - teljes 345
 következetetési keretprobléma 400  
 következetetési szabály 265
  - egzisztenciális bevezetés 382
  - egzisztenciális példányosítás 334
  - univerzális példányosítás 334
 követő állapot axióma 400  
 köztiltsó forma 929  
**Közlegelő tragédiája** 737  
 közömbösségi elv, Laplace-féle 556  
 közös szándék 533  
 központi határolószlá tétel 1111  
 köztes nyelv 971  
**KPF** lásd kölcsönösen preferenciálisan független  
**KPML** generáló rendszer 945  
 kritikus útvonal 496  
**KRK** sakkvégi játkra specializált gép 236  
**KRYPTON** 439  
 kudarc 111  
 kukucskál 762  
 kulcs 341  
 külsőség 737  
 küszöbüfüggvény 845  
 küszöbpont 765  
 kvadratikus dinamikus rendszerek 177  
 kvalifikációs probléma 400, 456, 509, 1077  
 kvalitatív fizika 394, 440  
 kvantálási tényező 661  
 kvantifikálás 929  
 kvantifikált term 930  
 kvantor 306
  - egzisztenciális 334
 kvázi-logikai forma 930  
**KVM** lásd kiterjesztett véges automata
- L**
- Lambert koszinusz-törvénye 989  
 Lambert-féle felület 989  
 láncolás
  - előrefelé 333, 378
  - hátrafelé 333, 350, 379
 láncszabály 582  
 látens szemantikai indexelés 978  
 láthatósági gráf 1064  
**LAWALY** 536  
**LEANTAP** 381  
 leértékelési tényező 876  
**LEGÁLIS-CSELEKVÉS** 130  
 leginkább megköött változó 500  
**LEGIOBBAT-ELŐSZÖR-KERESÉS** 137  
 legkisebb megkötés 464  
 legközelebbi-szomszéd 841, 863
  - 3-legközelebbi-szomszéd 861
  - k-legközelebbi szomszéd 843
 legrövidebb leíró hossz 778  
 legalvolszínűbb magyarázat 632

- legvonában mért távolság 137  
 lehetőségelmélet 621  
 lehetőségi axióma 398  
 lehetőséges világok 437, 607  
 lehűtés 158  
 leíró logika 419, 423
  - konziszenciavizsgálat 423
  - osztályozás 423
  - részvizsgálat 423
 lekérdezés 312, 958  
 lekérdező nyelv 958  
 lekötései lista 312  
 lentről felfelé elemzés 913  
 lépésváltás 213  
 leszámtolási tényező 712  
**LEVÁGÁS-TESZT** 222  
 leválasztás 612  
 levélcsomópont 110  
 levelezési fa 910  
**LEX** 791  
 lexikai-funkcionális nyelvtan (LFG) 944  
 lexikális többértelműség 934  
 lezárás 425  
 lezárt osztályok 912  
**LPG** *lásd* lexikai-funkcionális nyelvtan  
**LIFE** nyelv 380  
**LIFO** 115  
**Lighthill-tanulmány** 54  
 lineáris
  - programozás 166, 177, 186
  - régresszió 827
  - rezolúció 371
  - tervkészítés 488**Linguistic String Project** 944  
**Linus** 811  
**Lisp** 54, 412, 1058  
 listák 316  
 literál 258
  - alap 453
  - függvénymentes 453**LITERÁL-MEGVÁLASZTÁSA** 808  
**Loebner-díj** 1074  
**Logic Theorist** 49, 50, 128  
 logika 34, 248
  - alapértelmezett 430
  - Boole 39
  - dinamikus 436
  - elsőrendű 39, 248, 297, 299, 751
  - fuzzy 547, 612, 615
  - induktív 556, 573
  - leíró 419, 423
 logista 34  
 magasabb rendű 301  
 modális 411, 437  
 modelpreferencia 429  
 nemmonoton 429  
 operátor 411  
 pozitíciós 751  
 temporális 301, 435  
 logikai
  - ágens 248
  - ekvivalencia 264
  - következetetés 255
  - mindentudás 412
  - minimalizálás 393
  - összekötőjel 306
  - pozitivizmus 37
  - programozás 273, 333, 350**Logistello** program 232  
 logisztikus függvény 845  
 log likelihood 823  
 lokális optimum 729  
 lokálisan strukturált 583  
 lokalitás 612  
 lokalizáció
  - Monte Carlo 1032**LPG** tervkészítő 489  
**LUNAR** rendszer 945  
 lustaság 547
- M**
- MA\* (memóriakorlátozott A\* keresés) 145  
**MAC** *lásd* élkonzisztenca fenntartása 192, 204  
**MacHack** 6 sakkprogram 237  
 magasabb rendű logika 301  
 magikus halmaz 350, 379  
 magyarázat 433  
 magyarázatalapú általánosítás 233  
 magyarázatalapú tanulás (MAT)
  - 414, 814
 Mahalanobis-távolság 842  
 makróoperátor 536, 814  
 Manhattan-távolság 148, 180  
 manipulátor 1023  
 marginalizálás 560  
 Markov döntési folyamat 709, 873, 1046  
 részben megfigyelhető 537  
 Markov lánc Monte Carlo módszer 604, 617, 621, 671, 672  
 Markov-feltétel 629  
 Markov-folyamat 41, 629  
 Markov-lánc 605, 629
  - ergodikus 605
 Markov-modell
  - rejtett 58
 Markov-takaró 586, 625  
 Markov-tulajdonság 708  
**MAT** *lásd* magyarázatalapú tanulás matematikai
  - indukció 367
  - tételbizonyítás 380
 materialista elmélet 1081  
 materializmus 37  
 mátrix 1108  
**MAX** 212  
**MAX-ÉRTÉK** 214, 219  
 maximális szint 475  
 maximális várható hasznosság (MVH) 549, 677, 681, 702, 713  
 Maximális Várható Haszon elve 549  
 maximin egyensúly 732  
 maximin technika 731  
**maximum**
  - a posteriori 863
  - globális 154
  - lokális 156
  - norma 716
 maximum-likelihood 823, 863  
**MCC** (Microelectronics and Computer Technology Corporation) 57  
**MCMC** *lásd* Markov lánc Monte Carlo  
**MCMC-KÉRDEZ** 605, 606  
**MDF** (Markov döntési folyamat) 709, 1046  
 részlegesen megfigyelhető 720, 1046  
 megalapozottság 257  
 megbánás 685  
 megcáfolási teljesség 269  
 megerősítés 751, 873  
 megerősítéses tanulás
  - hierarchikus 899
 megerősítési mérték 573  
 megértési probléma 906  
 megfigyeléses állítás 37  
 megfigyelési modell 630  
 megfogás 1065  
 meghatározás 799  
 meghatározatlanság 612  
 megmaradási nyil 654

- megnyílkozás 905  
 megoldás 99, 111  
   ciklikus 515  
   kinyerés 476  
   optimális 101  
 megszámlálható főnév 395  
 mélységelesség 988  
**MÉLYSÉG-KORLÁTOZOTT-KERESÉS**  
 117  
 memoizálás 795  
 memók gyűjtése 357  
**MEMS-technológia** 1097  
 mentális állapot  
   szűk tartalom 1083  
   tág tartalom 1083  
 mentális modell 937  
 mentális objektum  
   szintaktikai elmélete 411  
**Mercer-tétel** 860  
**mereológia** 437  
**MERLIN** 438  
 mérték 393  
   rendezett 393  
**Message Understand Conference (MUC)** 980  
 mesterséges élet 177  
 mesterséges intelligencia 23, 31, 49  
**META-DENDRAL** 790  
 metafora 935, 946  
 metakövetkeztetés 234  
 metamorfózis nyelvtan 944  
 metaszabály 359  
 metónimia 935, 946  
**Metropolis-algoritmus** 177  
**Metropolis-Hastings-mintavételezés** 607  
 metszés 210  
 mezőátlag 620  
**MFF (minimális feszítő fa)** 180  
**MGONZ** 1074  
**MGSS\*** megközelítés 238  
**MH-1** kéz-szem gép 1064  
**MI** *lásd* mesterséges intelligencia  
**Microelectronics and Computer Technology Corporation** 57  
**Micro-Planner** 378  
**Microsoft Windows** 618  
   Microsoft Office 618  
   Nyomtató varázsló 618  
   Office segéd 618  
**mikrohalál** 686  
**mikrovilág** 51  
**MIN** 212  
 mindenutás 71  
**MIN-ÉRTÉK** 214, 219  
 minimális feszítő fa (MFF) 176, 180  
 minimális hosszúságú leírás 822  
**MINIMÁLIS-KONZISZTENS-MEGH**  
 801, 802  
 minimax  
   algoritmus 513  
   döntés 213  
   érték 213  
**MINIMAX-DÖNTÉS** 214, 220  
**MINIMAX-ÉRTÉK** 216  
**MIN-KONFLIKTUS** 197, 198  
 minősítési probléma 545  
 mintaadatbázis 151, 176  
   diszjunkt 152  
 mintaillesztés 346  
 mintakomplexitás 772  
 mintavételezés  
   elutasító 600  
 mintavételező  
   Gibbs-féle 606  
   Metropolis–Hastings 607  
 mintavételei frekvencia 661  
**MIS** 815  
**MLC** *lásd* Monte Carlo lokalizáció  
 modális logika 411, 437  
 modell 254, 607  
   átmeneti 708, 1030  
   együgyű Bayes 567  
   ellenőrzés 255  
   elimináció 379  
   időbeli valószínűségi 609  
   kvalitatív döntési 699  
   legközelebbi szomszéd 841  
   megfigyelési 721  
   mentes 886  
   minimál 426  
   mozgás 1030  
   naiv Bayes 567  
   okszági 699  
   relációs valószínűségi 607  
   stabil 428  
   szenzor 1030  
   többszörös ok keverék 978  
   választás 863  
 modellalapú következtető rendszer  
 319  
**Modus Ponens** 265  
   általánosított 337  
 mohó, végével felfedezés határán (VFHM) 883  
 mondal 248, 907  
 monista elmélet 1081  
 monitorozás  
   cselekvés 520, 521  
   terv 520, 523  
 monotonitás 267, 428, 680  
**Monte Carlo lokalizáció (MLC)**  
 1032  
 Monte Carlo módszer 598  
**Moore-törvény** 43  
 morfolás 1011  
 mozgás 996  
   felügyelt 1047  
   követő 1039  
   önbeállító 1047  
**MRS nyelv** 359  
**MRV heurisztika** 190  
**MUC (Message Understand Conference)** 980  
 multiplikatív hasznosságfüggvény 690  
**MUNIN** 618  
 munkatér-reprezentáció 1039  
 mutáció 161  
 működésmód 736  
   stratégiamentes 738  
   tervezés 727, 736  
**MVH** *lásd* maximális várható hasznosság  
**MYCIN** 56, 614, 621
- N**
- n* karú rabló 883  
 naiv Bayes-modell 825, 960  
**NASA** 378, 440, 536, 622, 1024  
**NASA Deep Space 1** űrhajó 485  
**NASA Remote Agent** 60  
 Nash-egyensúly 729  
**NASL** tervkészítő 538  
 navigációs függvény 1046  
 navigálás  
   tárgy menti 1047  
**NAVLAB** 60  
 negáció 258  
 negálás mint kudarc 427  
 négyzetté kiegészítés 645  
 nemdeterminisztikusság  
   korláatos 509  
   nem korláatos 509  
 nemellenőrzött 832  
 nemmonoton 429

- nemparaméteres tanulás 841  
 nemteljességi téTEL (Gödel) 359, 367  
 nemterminális 1112  
 nemzáró szimbólum 908, 1112  
 Nemzetközi ūrálomás 1060  
 NEL 439  
 neurális háló 48, 53, 57, 232, 844, 857  
 egy-rejtett-rétegű 861  
 egyrétegű előrecsatolt *lásd* perceptron  
 előrecsatolt 846  
 hardver 48  
 interpretációja valószínűségeként 850  
 perceptron 848–850  
 radiális bázisfüggvény 867  
 reprezentációs erje 54, 848  
 számítástechnika 844  
 tanulás 47–48, 856  
 többrétegű 54, 852–856, 863  
 többrétegű előrecsatolt 863  
 turbózott 861  
**NEURÁLIS-HÁLÓ-HIPOTÉZIS 851**  
**NEUROGAMMON 239**  
 neuron 42, 844  
     axon 42  
     dendrit 42  
     mesterséges 48  
     szinapszis 42  
     szóma 42  
 névelő 911  
 Newton–Raphson-módszer 165  
 Nine-Men's Morris (játék) 231  
**NIST (National Institute of Standards and Technology) 979**  
**NLP-rendszer 945, 947**  
**NOAH 488, 536**  
**NONLIN 488**  
**NONLIN+ 535**  
 normális eloszlás 1110  
     kovarianciamátrix 1111  
 normalizáló konstans 560  
**Novum Organum (Bacon, F.) 37**  
**NP-néhéz 129**  
**NP-teljes 40, 158, 439, 1006**  
**NSS sakkprogram 236**  
 nullhipotézis 764
- Ny**
- nyelv 905, 906–908  
     analízis 909
- cselekvés 906  
 észlelés 909  
 fordítás 53, 799, 969–977  
 formális 906  
 generálása 908  
 generálása DCG-kkel 933  
 gondolat 300  
 megértés 33, 48, 53, 905  
 megértés, korpuszalapú 951  
 modell 659, 937  
     természetes 34, 56, 75, 91, 247, 248, 299, 907  
 nyelvészeti 56  
 többértelemliség 299, 937  
     szinonimák 962  
     szövegértés 937  
 nyelvészeti 937  
     optimalitás elmélete 946  
     szárműtépes 48  
 nyelvhasználat, beágazott 943  
 nyelvi modell 660, 937, 973  
     bigram 951  
     n-gram 952  
     trigram 951  
     unigram 951  
     valószínűségi 951  
 nyelvi modellezés 959  
 nyelvtan 907, 943, 1112  
     definit klóz 922  
     indukciós tanulása 941, 943, 946  
     kiterjesztése 943  
     kiterjesztett 921  
     környezetérzékeny 908  
     környezetfüggetlen 908  
     valószínűségi  
     környezetfüggetlen (PCFG) 954, 956  
     reguláris 908  
     rekurzíván felsorolható 908  
     szókinccsel ellátott  
     valószínűségi 978
- nyereségarány 765  
 nyereségsfüggvény 211  
 nyesés 143, 210, 216  
     alfa-béta 216  
     hattátlanság nyesése 230  
 nyílt hurkú rendszer 100  
 nyílt világ feltételezés 480, 518  
 nyitott kód 355  
 nyitott lista 122  
 nyitott osztályok 912  
 nyomatékmérő 1026
- O**
- objektum 299  
 összetett 392  
 objektumfelismerés 995  
     fényességalapú 1009  
     jellemzőalapú 1009  
 objektumorientált programozás (OOP) 420  
 Ockham beretvája 752, 757  
 odometria 1026  
 ok-okozati szabályok 319  
 okozati háló 578  
 okozati kapcsolat  
     kiterjesztése 526  
 oktános 1005  
 oldallépés 156  
**Online-Mélységi-Ágens 170, 171**  
**ONTIC 374**  
 ontológia 321  
     felső 388  
     ontológiaszervezés 387  
 ontológiai meghatározottság 300  
**OOP *lásd* objektumorientált programozás**  
**Open Mind Initiative 435**  
 operációkutatás 41  
 operátor 100  
**O-PLAN tervkészítő 508, 535, 536**  
**OPS-5 programozási nyelv 349, 378**  
 optikai folyam 996  
 optikai karakterfelismerés 909  
 optimális  
     agykárosodás 857  
     döntés 211, 214  
     hatékonyságú algoritmus 143  
     stratégia 212  
     szabályozás elmélet 177  
     szabályozó 1050  
 optimalitás  
     korlátozott 1101  
     aszimptotikus korlátozott 1102  
 optimalizálás  
     korlátozott 165  
     problémák 154  
 optimizmus  
     bizonytalanság mellett 172  
**OPTIMUM-AIV 535**  
 ostábla 78, 210, 224, 232  
 oszd-meg- és-uralkodj 175  
 osztályozás 754

- osztályozás, nem ellenőrzött 832  
**OTTER** tételebizonyító 371, 372, 374, 381, 384
- Ö**
- önérzékelő 1026  
 öröklődés 390, 608  
     többszörös 420  
 örökösi bűntetés 734  
 összefestüles 127  
 összegzés 1105  
 összehúzás 716  
 összekötőjel 258  
 összetett mondat 258
- P**
- P3 robot 1024  
 PageRank algoritmus 979  
 pályatervezés 1039  
**PARADISE** 235  
 paraméteres tanulás 841  
 paraméterfüggetlenség 829  
 paramétertanulás 822  
 paramoduláció 369  
 Pareto-dominált 729  
 Pareto-optimális 729  
 párhuzamos elosztott feldolgozás 844  
 partició 391  
 Pascaline (Pascal) 36  
**PATHFINDER** 618  
 PCFG *lásd* nyelvtan,  
     valósínliségi  
     környezetfüggetlen  
 PDDL *lásd* tervkészítési terület  
 definíciós nyelv  
 Peano-axiómák 314, 345  
 példa  
     hamis negatív 784  
     hamis pozitív 784  
**PÉLDA-KITERJEDÉS** 807  
 példányalapú  
     modell 863  
     tanulás 841  
 Pengi 538  
**PENMAN** 945  
 perceptron 53, 848–850, 866  
     Gamba 866  
     konvergencia tétele 53
- küszöbüfüggvény 848  
 madeline 866  
 reprezentációs ereje 54, 848  
 perem 110  
 peremeloszlás 559  
**PILLANATNYILAG-LEGJOBB-TANULÁS** 785  
**PLAN-ERSI** 535  
**PLANEX** 538  
**PLANNER** 56, 378, 379  
 póker 78, 228  
 polifa 597, 617  
 polinom  
     súlyozott 751  
**POMDP** 537  
 ponttétek 221  
 ponttól-pontig 1039  
 populáció 160  
 porszívóvilág 67, 103, 512  
 potenciáltér 1044, 1052  
 pozícióbecslés 1009  
 pragmatika 907  
 pragmatikus értelmezés 910, 932  
 predikátum  
     általánosítási hierarchia 790  
     időtlen, örök 397  
     kiterjedése 783  
 predikátmuszimbólumok 304  
 preferencia  
     függetlenség 689  
     monoton 682  
     sorrend 548  
     stacionárius 711  
     szemántika 945  
     tranzitív 679  
 prefix 315  
 prior  
     egyenletes 822  
     hipotézis 819  
     konjugált 828  
**PRIOR-MINTA** 599, 658  
 prioritásos végigsöprés 881  
 probléma 100  
     8-királynő 104, 155  
     adatasszociációs 1033  
     állapotterre 100  
     dekompozíció 452  
     ellenfél probléma 123  
     elrablásos 1030  
     eshetőségi 123, 126  
     selfedezési 124, 166  
     inkrementális megfogalmazása 105
- interneten keresés 107  
 játékprobléma 102  
 komutatív 187  
 körutazási probléma 106  
 majdnem dekomponálható 452  
 megfogalmazás 98  
 megoldása 101  
*n*-királynő 176  
**NP** 1107  
 optimalizációs 154  
 ramifikációs 401  
 relaxált 150, 176, 463  
 részprobléma 151  
 szenzor nélküli 123, 124  
 teljes-állapot megfogalmazása 105  
 utazó tigynők 106, 180  
 útkeresési probléma 106  
 valós világbeli 103  
**VLSI** elhelyezési probléma 107, 129  
 vonalcímkezési 1004
- problémaosztály  
     #P 1107  
     #P-teljes 1107  
     co-NP 1107  
     co-NP-teljes 1107  
     NP-teljes 1107  
     P-TÁR 1107
- problémater leíró nyelv 487  
 procedurális  
     kibővítés 418  
     kiegészítés 422  
**PRODIGY** 536  
 produkción 349  
 produkciós  
     rendszer 333, 349, 378  
     szabály 81  
**PROGOL** 810, 812, 815  
 Prolog 352, 428, 944  
 Prolog Technológia  
     Tételebizonyító (PTTP) 373, 381  
**PROSPECTOR** 621  
**PROVERB** 61  
**PRS** 538  
 PSPACE-teljes 464, 487  
 pszeudoél 1004  
 pszichológiai következetetés 440  
 PTTP *lásd* Prolog Technológia  
 Tételebizonyító  
**PUCINI** 538  
**PUMA** 1063

**Q**

QA3 436, 487  
 QALY 686, 702  
*Q*-függvények  
 paraméterezeit 892  
 QLISP 380  
*Q*-tanulás 874, 886  
 qualia 1083  
 Cubic 231

**R**

RI 57, 349, 378  
 rablók problémája 883  
 racionális 32  
 ágens 66, 68, 70  
 cselekvés 62  
 racionálisztika 31  
 korlátozott 36, 1101  
 számitható 1101  
 tökéletes 1101

RADFT (relevanciaalapú döntési  
 fa tanuló algoritmus) 802, 815  
 radiális bázisfüggvény 867  
 Radio Rex 672  
 ramifikációs probléma 401  
 randomizálás 83  
 RAP 538  
 RAPS (reaktív akcióterv rendszer)  
 1058  
 Rauch-Tung-Striebel-simítás 671  
 RDF 435  
 reaktív 1053  
 reciprokrang 962  
 redukálás 428  
 redundáns lépés 526  
 referenciaelmélet 39  
 referenciaiális

álláthatóság 411  
 index 932, 945

referenciapálya 1050

referenciapont 1031

referenciaszabályozó 1050

reflektanciatérkép 1002

reflektív architektúra 1100

REFLEXSZERŰ-ÁGENS 82

REFLEXSZERŰ-ÁGENS-ÁLLAPOT 84

REFLEXSZERŰ-PORSZÍVÓ-ÁGENS 81

regresszálás 462

regresszió 754

lineáris 827

regressziós fa 766

reguláris kifejezés 967

reifikálás 390, 410, 421

REINFORCE 898

rejtett Dirichlet-alkotáció 978

rejtett egység 847

rejtett Markov-modell (HMM) 58,

627, 639, 838, 864, 956, 1029

rekurrents háló 846, 867

Hopfield-háló 867

rekurzív függés 609

rekurzív legjobbat-először keresés

(RLEK) 144, 145, 175, 182

REKURZÍV-LEGIOBBAT-ELŐSZÖR-

KERÉS 145

REKURZÍV-MKK 117

REKURZÍV-VISSZALÉPÉS 188

reláció 299

zajos-VAGY 587

relációs valószinűségi modell

(RVM) 607, 621

relevancia-visszacsatolás 963

Remote Ágens 91, 375, 536

rendez 384

REPOP 489

reprezentációs

keretprobléma 400

tétel 689

REPRODUKCIÓ 162

részben rendezett tervkészítés

(RRT) 488

részecél függetlenség 463

részecskezűrő 657

Rao-Blackwellized 1064

részsemény 404

részfeladat megosztás 504

részleges kiértékelő módszer 814

részlegesen megfigyelhető Markov

döntési folyamatok (RMMDF) 720,

1046

részprogram 899

rete algoritmus 349

réteg 847

modellező 1057

reaktív 1057

szekvenciális 1057

végrejártó 1057

retorikai struktúra elmélet (RST)

946

Reversi (számítógépe játék) 232

rezolució 51, 267, 359

inverz 809

lineáris 810

rezolució

SL 379

SLD 379

rezoluciós lezárt 271

rezoluciós lezárt 366

rezolvens 809

RLEK lássd rekurzív legjobbat-

először keresés

RMM lássd rejtett Markov-modell

RMMDF lássd részlegesen megfí-

gylhető Markov döntési folyama-

tok 720, 1046

Robbins-algebra 374

Robocup 1064

robot 1023, 1062

differenciál hajtású 1028

dinamikusan stabil 1028

holonomikus 1028

humanoid 1024

mobil 1023

nemholonomikus 1028

statikusan stabil 1028

szinkron hajtású 1028

robotfoci 1061

robotika 33, 1063

robotnavigáció 107

ROC görbe 961

rotációs csukló 1027

RRT lássd részben rendezett terv-

készítés

RSA algoritmus 375

RST lássd retorikai struktúra elmélet

Rubik-kocka 128

rúdegyensúlyozási feladat 891

RUP rendszerek 440

RVM lássd relációs valószinűségi

modell

**S**

SAINT 51

sajátmozgás 997

sakk 78, 230

SAM 374, 381

SAPA 535

Sapir-Whorf-hipotézis 300

sásztrai szanszkrit 434

SATPLAN 481, 486, 489, 494, 537,

811

sztochasztikus 537

Scrabble™ 240

segédszavak 925

- segédváltozó 186  
 selejtezés 161  
 séma 162  
     példánya 162  
 sémabeszerzés 814  
**SEQUITUR** 941, 947, 957  
**SETHEO** tételbizonyító 381  
**SGP** gráftervező 489  
*Shakey*, a robot 28, 51, 487, 493, 536, 538, 1063, 1065  
**SHRDLU** 53, 56, 378, 458  
 sikertelenségi tagadás 353  
 simítás 632, 667, 952  
     adj hozzá egyet 952  
     Good-Turing 978  
     lineáris interpoláción alapuló 952  
     törölt interpolációs 978  
     Witten-Bell 978  
**SIEPE** 535, 536, 538  
**SKETCHPAD** 203  
**Skolem-függvény** 361  
 skolemizáció 335, 361  
**Skolem-konstans** 335  
**SLD-rezolúció** 379  
**SL-rezolúció** 379  
**SMODELS** 439  
**SNARC** 48  
**SNLP** 488  
**SOAR** 59, 349, 378, 536, 814, 1099  
 Socratic következtető 374  
**Sojourner** mobil robot 1024  
 sorba rendezhető részcel 485  
**SORBAÁLLÍTÓ** 111  
**SORBAÁLLÍTÓ-MIND** 111  
 sorozatos újramintavételezés fontossági mintavételezéssel 672  
 sorrendezhetőség 680  
 sortrendi áramkör 283  
**SORT-LÉTREHOZ** 110  
 sörétkamra 986  
 spam 981  
**SPASS** 381  
**SPIKE** 536  
**SPIN** 375  
**SQL** 425  
**SRI** *lásd* Stanford Research Institute  
 stacionárius eloszlás 634  
**STAHL** 816  
**STAN** 489  
 standard normális eloszlás 1110  
 Standard Upper Ontology Working
- Group 435  
**Stanford Cart** 1063  
 Stanford Research Institute (SRI) 1063  
 statikusan stabil 1028  
 stílusmérés 981  
 stratégia 76, 212, 727, 1046  
     domináns 728  
     erősen dominál 728  
     énték 893  
     gradiens 893  
     gyengén dominál 728  
     keresés 892, 896  
     kevert 728  
     kiértékelési feladat 875  
     megfelelő 899  
     szernet-szemért 735  
     sztochasztikus 893  
     tiszta 727  
 stratégiaiteráció  
     algoritmus 875  
     módosított 877  
 stratégiaprofil 729  
**STRIPS** 487, 453, 495, 531, 536, 536, 538, 741, 814  
 STRIPS feltételezés 454  
 strukturális EM 840  
 strukturális többérműség 934  
**STUDENT** 52  
**SUFM** *lásd* szekvenciális fontossági mintavételezésű újramintavételezés  
 síly 845  
 sílyozott kétréti illesztés 1012  
**SÚLYOZOTT-MINTA-VISSZATÉTEL** 658  
 Sussman anomália 488, 492  
 sűrítés 672  
 sűrűségbecslés 841  
 System for Interactive Planning and Execution Monitoring 538  
**SYSTRAN** 970
- Sz**
- szabad akarat 1081  
 szabadságfok (DOF) 1026, 1027  
     effektív 1028  
     irányítható 1028  
 szabály  
     átíró 908  
     Bayes 40, 546
- delta 889  
 Dempster 614  
 diagnosztikus 318  
 feltétel-cselekvés 81  
 ha-akkor 81  
 képzési 1112  
 következetlési 265  
 láncszabály 582  
 metaszabály 359  
 ok-kozat 319  
 produciós 81  
 szituáció-cselekvés 81  
 szorzat 554  
 Widrow-Hoff 889, 897
- SZABÁLY-ILLESZTÉS** 82  
 szabályozáselmélet 649, 671, 672, 710, 741, 865, 891  
 adaptív 876  
 szabályozó 1050, 1051, 1052  
     optimális 1050  
     P 1051  
     PD 1051  
     PID 1052  
     referencia- 1050  
     stabil 1051  
     szigorúan stabil 1051  
 szabályséma 925  
 szabványos minőségű életév 686  
 szakértő rendszer 56  
 számítógépes  
     grafika 984  
     nyelvészeti 48  
     tanulás elmélet 770  
 Számítógépes Bridzsbajnokság 233  
 Számítógépes Sakkvilágbajnokság 230, 237  
 számososság 304  
 szándékok 91  
 szándékolt interpretáció 304  
 származtatási analógia 814  
 szegmentáció 666, 953  
 szekvenciális fontossági  
     mintavételezésű  
     újramintavételezés (SUFM) 672  
 szemantika 254, 907, 926  
 szemantikai  
     értelemezés 910, 926, 943  
     többérműség 934  
 szemantikus háló 419, 435  
 Szentpétervári paradoxon 705  
 szenzor 1023  
     aktív 1025

- inercia 1026  
passzív 1025
- szeparálható, lineárisan 863
- szerencsés rábukkanás 523
- szigmoid függvény 845
- szignifikanciateszt 764
- szillagizmus 376
- szílletű algoritmus 1064
- szimbolikus valóságnisúgi következetés (SzVK) 618
- szimbólumfelosztás 484
- szimplex algoritmus 177
- szimulált lehűtés 136, 158, 177, 205
- SZIMULÁLT-LEHŰTÉS** 159
- szimultán helymeghatározás és térképezés (SZLT) 1034
- szinguláris kiterjesztés 223
- szinkrón hajtás 1028
- szinkronizáció 530
- szinonima 417, 944, 962
- szintköltség 475
- szintaktikai analízis 913 többérmelőseg 934
- szintaktikus édesség 315
- szintaxis 254, 258
- szintézis 374
- Szisztematikus Nyelvtan** 945
- szituáció 397
- szituáció–cselekvés szabály 81
- szituációkalkulus 396 hatásaxiómá 398 keretaxiómá 399 követő állapot axióma 400 lehetőségi axióma 398
- szkeletonzállás 1039, 1044
- SZLT** szimultán helymeghatározás és térképezés
- szó 906
- szoftbot 75, 91
- szoftmax függvény 893
- szóilleszkedési vektor 977
- szókincs 911, 965
- szólásaktus 905, 943 deklaratív 906
- szorzás, pontonkénti 595
- szorzatszabály 554
- szótövesítés 962
- szöghelyzet dekódoló 1026
- szöveg 937
- szövegértés 937, 943
- szövetség 215
- Szputnyik 53
- sztereolátás 996
- sztochasztikus viselkedés 77
- szubsztancia temporális 406 tébeli 406
- szupport vektor gép 857, 861
- szűlői feltétel 581
- szűrés 431, 632, 721, 898, 1029
- szűrő Kalman-szűrő 1032, 1033 kiterjesztett Kalman-szűrő 1033 részecske-szűrő 1032
- szűróalgoritmus 1097
- SzVK** lásd szimbolikus valóságnisúgi következetés 618
- T**
- T4 535
- tábla 672
- táblaarchitektúra 672
- táblázatos logikai programozás 357, 380
- TABLÁZAT-VEZÉRLÉSÜ-ÁGENS** 80
- TACAIR-SOAR 378
- TAG (Tree-Adjoining Grammar) 944
- taktilis 984, 1026
- találati lista 965
- támogató halma 370, 371
- tanítási görbe 855
- tanító halma 756, 762 ismétléses 767 súlyozott 767
- tanulás 71, 90, 524 aktív 874 Bayes-tanulás 819 ellenőrzői 750 feltételek törlése 786 felügyelt 888 gépi tanulás 33 gyenge 768 hatáthalmaz 788 hierarchikus megerősítéses 899 induktív 153 jelölt eltávolítási algoritmus 787 konstruktív indukciós 806 kumulatív 813 legkisebb megkötés elvű algoritmus 787
- magyarázatalapú 536, 793
- MAT** 793
- maximum-likelihood 863
- megerősítéses 750, 873
- metaszintű 147
- nem ellenőrzött 750
- nemparaméteres 841
- paraméter 822
- paraméteres 841
- passzív 874
- példányalapú 841, 898
- pillanatnyilag legjobb hipotézis 784
- Q**-tanulás 886
- RAT** 793
- relevanciaalapú 793
- TIT** 794
- tudásalapú deduktív 794
- verziótér 787
- tanulási faktor 850 görbe 762
- tanulási probléma nemrealizálható 753 realizálható 753
- tanuló komponens 749
- tanuló valós idejű A\* (TRTA\*) 171, 178
- tárgyterület 302, 311
- tárigény 111
- TÁROL** 340, 383
- társadalmi törvényszerűség 532
- tartalomalapú képkeresés 1007
- TARTOMÁNY-ÉRTÉK SORRENDEZÉSE** 188
- tartóhiba-modell 654
- TAUM-METEO** 970, 980
- tautológia 264
- távlatpont 987
- TAVOLÍTSD-EL-AZ-ELEMET** 111
- távolság euklideszi 842 Hamming 842 Mahalanobis 842
- távolságmérő pásztázó 1025
- taxonómia 390
- Taylor-sorfejtés 1033
- TD-GAMMON** 232, 239
- technológiai szingularitás 1091
- teljes adat 822

- állapotmegadás 185  
 él 916  
 információ értéke 695  
 teljes információ értéke (TİE) 695  
 teljesítménymérés 1105  
 teljesítménymérték 69, 73, 89  
 teljesség 111, 256  
 teljességi tétel 359  
 temporális logika 301, 435  
 tény 272, 548  
     átnevezése 344  
 tényállás 548  
 térbeli következetés 440  
 térképszínezési probléma 347  
 term 305  
 termékenység 974  
 természetes  
     dedukció 377  
     fajta 394  
     nyelvek 907  
         nyelvfeldolgozás 33, 48  
**TERMINÁLÁS-TESZT** 219  
**terminális** 1112  
**terv**  
     felismerése 533  
     kimenet 548  
     konzisztszencie 467  
     könyvtár 501  
     részben rendezett 465  
     sorarendezése 465  
     teljesen rendezett 464  
     üres 465  
 tervkészítés 85, 235, 451  
     alkalmazkodó 509, 537, 1047  
 érzékelőmentes 509  
 esetalapú 536  
 eshetőségi 510  
 feltételes 510  
 folytonos 510  
 hierarchikus feladathálón  
     (HFH) 500, 899  
 klasszikus 509, 451  
 multiágens 529  
 regressziós 461  
 többtestű 530  
 tervkészítési gráf 472  
 tervkészítési terület definíciós nyelv  
     (PDDL) 456, 487, 542  
**TERVKÉSZÍTŐ** 521  
 tervkészítő, reaktív 538  
 tervsérülés 527  
 test-elmé probléma 1081  
 teszthalmaz 761  
 tételelbizonyító  
     nem teljes 356  
     rendszer 333  
 texel 1001  
 textúra 996, 1001  
 textúragradiens 1002  
 ThEO 1099  
 Three Mile Island 1059  
**TİE** lásd teljes információ értéke  
 695  
 tiltólistás szó 965  
 tisztán induktív következetés 751  
**TKBÉ** (Teljesítmény, Környezet,  
 Beavatkozók, Érzékelők) 73, 92  
**TMS** lásd igazság-karbantartó  
 rendszer  
 tokenizálás 968  
 totális függvények 303  
 többattribútumú hasznosságelmélet  
 686  
 többérműség 299, 943  
     fejlődása 910, 934, 936  
     lexikális 934  
     szemantikai 934  
 többségsfüggvény 848  
 többváltozós Gauss-eloszlás 1110  
 tömörített erdő 920  
**TRANSZFORMÁCIÓ-KERES** 1021  
 transzhumanizmus 1091  
 transzlációs csukló 1027  
 transzpozíció 218  
 transzpozíciós tábla 218  
 tranzitivitás 680  
**TREC** konferencia 966  
 Tree-Adjoining Grammar lásd  
**TAG** 944  
 triéder 1004  
 trigram 667  
**TRTA\*** lásd tanuló valós ideű A\*  
**TRTA\*-ÁGENS** 172  
**T-SCHED** 535  
**TSP** 106, 129, 180, 181  
 tudás  
     akusztikai modell 937  
     ~alapú 886  
     ~bázis 248, 249, 262  
     ~darabolás 814  
     ~forrás 672  
     ~háló 578  
     ~ítéletállítás 518  
     komplítálása 814  
     megszerzése 320  
     mentális modell 937  
 nyelvi modell 937  
     ~reprezentáció 33, 48  
     ~reprezentációs nyelv 248  
     ~szint 249  
     ~tervezés 320  
 tudatosság 1082  
 tulajdonság 299  
     belső 396  
     kinyerés 984  
     külső 396  
 túlgenerál 912  
 túllilleszkedés 763, 821, 856  
 turbó dekódolás 620  
 turbózás 767  
**Turing-automata** 39  
**Turing-gép** 1076  
**Turing-teszt** 32, 35, 63, 1074  
     teljes 33  
**TWEAK** 488
- U, Ü**
- UAV lásd ember nélküli légi jármű  
**UCPOP** 488  
 udvarias felháborodás 1079  
**ÚJ-KLÓZ** 807  
**ÚJ-LITERÁLOK** 808  
 újrahasznosítható 498  
 újraír 384  
 újratervezés 510  
**ÚJRATERVEZŐ-ÁGENS** 521  
**ÚJ-VÁLTOZÓ** 355  
**ULV** lásd ember nélküli közúti jármű  
**UNA** lásd egyedi megnevezés felháborodás  
 Unimate, robot 1063  
 univerzális  
     kvantor 334  
     nyelvtan 946  
     példányosítás 334  
**UnPOP** 489  
**UOSAT-II** 535  
**URL** 415  
**URP** lásd Utility Reasoning Package  
     út  
         költsége 185  
     utalásfeloldás 937  
     Utility Reasoning Package (URP)  
 702

útköltés 101, 103  
 utolsónak-be-elsőnek-kí *lásd LIPO*  
 útvonal 354, 356  
 -konziszenciá 193  
 UWL 537  
 ÜRES? 110  
 ütemezési feladat 495  
 ütemterv 496

## V

vágási teszt 220  
 vágóhalmaz-kondicionálás 201  
 vagyak 91  
 VAGY-párhuzamosság 356  
 válaszhalmaz 428  
 -programozás 428  
 választidő 962  
 választási pont 354  
 validáció

kereszvalidáció 861

váll 156

valós idejű MI 1099

valószínűleg közelítőleg helyes  
 (VKH) 799  
 -tanuló 770

valószínűség 1109

a posteriori 548, 554

a priori 548, 552

előzetes 548

feltétel nélküli 548, 552

feltételes 548, 554

utólagos 548

valószínűség-elmélet 40, 301

valószínűség-eloszlás 553, 579

együttes 553

teljes 553

feltételes 579

valószínűségi eloszlásfüggvény

1110

valószínűségi háló 28, 59, 578

-kvalitatív 688

valószínűségi környezetfüggetlen

nyelvtan (PCFG) 954, 978

valószínűségi következetetés 559

valószínűségi modell 1110

valószínűségi súlyozás 601, 617

valószínűségi sűrűségfüggvény

1110

valószínűségi úthálózat 1045

VALÓSZÍNÜSEGI-SÚLYOZÁS 601

valószínűség-sűrűségfüggvény 553

valószínűség-számítás 547  
 változó 183

átnevezése 339

bizonyítékváltozó 591

célváltozó 591

diszkrét 185

dolgok 932

eliminálás 617, 655

értéke 183

értéktartomány 551

futási idejű 413

rejtett 591

statikus 1113

tartománya 183

véges tartományú 185

végétlen tartományú 185

véletlen 550

várakozási sor 110

várható

érték 221, 226

hasznosság 677

pénzügyi érték 682

várhatóérték-maximalizálás (EM)

algoritmus 669, 956

várhatóminimax 226, 239

VÁRHATÓMINIMAX 226, 725

variációs

közeliítés 620

paraméter 620

VAX 378

VC-dimenzió 778, 872

VC-kapcsolat 439

védett tartomány 466

végállapot 211

VÉGÁLLAPOT-TESZT 222

végbeavatkozó 1027

véges állapotú automata (FSA) 968

véges automata

-kiterjesztett 1054

véges tárgyterület 358

végigöprés, prioritás 881

végrehajtás 99

-monitorozás 510

végétlen felfedezés határán mohó

(VFHM) 883

végtessz 211

vektor 1108

vektorkvantálás (VK) 662

vektortér

-modell 965, 979

-hisztogram 1065

véletlen változó 550

-diszkrét 551

folytonos 551

logikai 551

véletlen vándorlás 170

vergődés 147

verifikáció 374

verseny 533

verziótér

összeomlik 790

VERZIÓ-TÉR-MÓDOSÍTÁS 787, 789

VERZIÓ-TÉR-TANULÁS 787

vezítés

-perspektivikus 987

-skálázott flüggőleges sík 987

vezéről 90

VFHM *lásd* mohó, végtelen felfedezés határán, végtelen felfedezés határán mohó

Vickrey-árverés 738

világ modell 937

virtuális 861

-számláló 829

VISELKEDÉSI NYELV 1057

VISSZAÁLLÍT-ÚTVONAL 355

visszafelé terjesztés 213

visszajegyzés 205

visszalépés

-dinamikus 205

VISSZALÉPÉS-KERESÉS 188

visszatekintés 632

visszaugrás 195

visszaverődés

-diffúz 989

-tükörz 989

visszavezetés

-ítéleti logikai állításokra 336, 608

visszavon 353

VISSZAVON 431

Viterbi-algoritmus 953

VK *lásd* vektorkvantálás

VKH *lásd* valószínűleg közelítőleg

-leg helyes

-vonalcímke 1003, 1004

-vonalkeresés 165

vonzat 254

-kényszer 791

-treláció, inverz 810

voronoi-gráf 1044

## W

WALKSAT 428, 484, 486, 489,

494, 1107

WAM 355, 379	X	zajos-VAGY 587, 588
WARPLAN 488, 537		zárás 373
Warplan-C 537	XCON 349	záró (vagy terminális) szimbólum
Warren Absztrakt Gép 355	XML 435	906, 1112
whist 228	XTAG projekt 944	zárt lista 122
Widrow–Hoff-szabály 889, 897	Z	zárt világ feltételezés (CWA) 425, 429, 518
Witness-algoritmus 741	Z-3 46	zavarodottság 953
Witten–Bell-simítás 978	zaj 758	zongoraszállítók 1064
Wordnet 944, 980	zajos-MAX 588	zsákolás 777

# I. RÉSZ

# MESTERSÉGES

# INTELLIGENCIA

---

# 1. BEVEZETÉS

Amelyben megkíséreljük megmagyarázni, hogy miért tartjuk a mesterséges intelligenciát olyan területnek, amit igenis érdemes tanulmányozni, és amelyben azt is megkíséreljük eldöntení, hogy vajon mi is ez a terület.

Az ember **Homo sapiens**nek – gondolkodó embernek – nevezte el a saját fajtáját, mert mentális képességeink annyira fontosak a számunkra. Évezredek óta próbáljuk megérteni azt, hogy *hogyan gondolkozunk*. Azt, hogy az anyag néhány maréknyi mennyisége hogyan képes a saját magánál sokkal nagyobb és bonyolultabb világot észlelni, megérteni, a világ alakulását megjósolni és manipulálni. A **mesterséges intelligencia** (*artificial intelligence*), röviden MI (AI), még tovább is megy: az intelligens entitások megértése mellett ilyen entitások építésével is próbálkozik.

Az MI az egyik legújabb tudományterület. A munka közvetlenül a második világháború befejeztével kezdődött. A mesterséges intelligencia elnevezés 1956-ban született meg. A molekuláris biológiával egyetemben az MI az a terület, amiről más tudományterületek kutatói úgy nyilatkoznak, hogy „én legszívesebben ezzel foglalkoznék”. Egy fizikushallgató teljes joggal gondolhatja, hogy Galilei, Newton, Einstein és mások már minden jó ötettel rég előálltak. Ezzel szemben az MI-ben az Einstein-kaliberű egyéni ségek előtt még szabad az út.

Az MI jelenleg az általános rendeltetésű területektől, mint az észlelés és a tanulás, egészen olyan specifikus feladatokig, mint a gépi sakk, a matematikai tételelbizonyítás, a gépi költészet vagy az orvosi diagnózis, a legkülönbözőbb részterületek óriási választékát öleli át. Az MI az intellektuális képességeket igénylő feladatokat rendszerez és automatizálja, és így potenciálisan releváns az értelmes emberi cselekvés minden területén. Ilyen értelemben az MI igazán univerzális tudomány.

## 1.1. MI AZ MI?

Eddig arról beszélünk, hogy miért izgalmas az MI, de nem mondta meg, hogy *mi is* valójában? Az MI definícióit, nyolc jelenlegi tankönyvet követve, az 1.1. ábra mutatja. Ezek a definíciók két dimenzió mentén értelmezhetők. Az ábra felső részében levők a *gondolati folyamatokat* és a *következtetést célozzák*, míg az alsó részben levők tárgya a *viselkedés*. A bal oldali definíciók a sikert az *emberi teljesítményhez* mérik, míg a jobb oldaliak mércéje az intelligencia egy *ideális* koncepciója, amit mi *racionálitásnak* (*rationality*) fogunk nevezni. Egy rendszer racionális, ha tudásához viszonyítva helyesen cselekszik.

Emberi módon gondolkodó rendszerek	Racionálisan gondolkodó rendszerek
<p>„Izgalmas újszerű kísérlet, hogy a számítógépet gondolkodásra készessük... tudatos gépek, e fogalom teljes és szó szerinti értelmében” (Haugeland, 1985)</p> <p>„Az emberi gondolkodással asszociálható olyan aktivitások [automatizálása], mint pl. a döntéshozatal, a problémamegoldás, a tanulás,...” (Bellman, 1978)</p>	<p>„A mentális képességek tanulmányozása számítási modellek segítségével” (Charniak és McDermott, 1985)</p> <p>„Az észlelést, a következtetést és a cselekvést biztosító számítási mechanizmusok tanulmányozása” (Winston, 1992)</p>
Emberi módon cselekvő rendszerek	Racionálisan cselekvő rendszerek
<p>„Az olyan funkciókat teljesítő gépi rendszerek létrehozásának a művészete, amelyhez az intelligencia szükséges, ha azt emberek teszik” (Kurzweil, 1990)</p> <p>„Annak tanulmányozása, hogy hogyan lehet a számítógéppel olyan dolgokat művelni, amiben pillanatnyilag az emberek a jobbak” (Rich és Knight, 1991)</p>	<p>„Számítási intelligencia az intelligens ágensek tervezésének a tanulmányozása” (Poole és társai, 1998)</p> <p>„Az MI... a műtárgyak intelligens viselkedésével foglalkozik” (Nilsson, 1998)</p>

1.1. ábra. Az MI néhány meghatározása négy kategóriába szervezve

Az idők folyamán minden irányzat követőre talált. De ahogy ez várható volt, feszültség uralkodik az embercentrikus és a racionalitáscentrikus irányzatok között.<sup>1</sup> Az embercentrikus irányzat szükségképpen empirikus tudomány, hipotézisekkel és empirikus igazolással.

A racionalitáscentrikus megközelítés a matematikára és a mérnöki tudományokra támaszkodik. Mindegyik csoport rágalmazta is, de segítette is a többi csoport munkáját. Nézzük most a négyféle megközelítést részletesebben.

## Emberi módon cselekedni: Turing-teszt megközelítés

A Turing-teszttel (Turing-test) Alan Turing javasolta azzal a céllal, hogy az intelligenciának egy kielégítő munkadefiníciót adjon (Turing, 1950). Ahelyett hogy az intelligenciára jellemző kvalitások hosszú és feltehetően vitás listáját megadná, Turing egy tesztet javasolt, amelynek alapja a vitathatatlanul intelligens entitástól – egy embertől – való megkülönböztethetetlensége. A számítógép akkor állja ki a próbát, ha az emberi

<sup>1</sup> Meg kell jegyeznünk, hogy az emberi és a racionális viselkedés megkülönböztetése nem jelenti, hogy az emberek szükségszerűen „iracionálisak”, „emocionálisan instabil”, „elnebeteg” értelemben. Azonban fel kell figyelni arra, hogy nem vagyunk tökkéletesek. Nem vagyunk valamennyien sakkmesterek, azok sem, akik a sakk minden szabályával tisztában vannak. Sajnálatos módon nem is vizsgázik mindenki jelesre. Az emberi következetést terhelő szisztematikus hibák egy részét Kahneman sorolja fel (Kahneman és társai, 1982).

kérdező néhány írásos kérdés feltevése után nem képes eldönteni, hogy az írásos válaszok egy embertől vagy egy géptől érkeznek-e. A tesztet a 26. fejezet tárgyalja részletesen, ahol azzal is foglalkozunk, hogy intelligensnek mondható-e a számítógép, ha teljesíti a tesztet. Jelenleg igen sok feladatot jelent egy gépet úgy programozni, hogy teljesítse a tesztet. A számítógépnek a következő képességekkel kellene rendelkeznie:

- **természetes nyelvfeldolgozás (natural language processing)** a sikeres angol (illetve más emberi) nyelvű párbeszédhez;
- **tudásreprezentáció (knowledge representation)** az ismert vagy hallott információ tárolására;
- **automatizált következtetés (automated reasoning)**, hogy a tárolt információt kérdezések megválaszolására és új következtések levonására használjuk;
- **gépi tanulás (machine learning)** az új körülményekhez való adaptálódáshoz, a mintázatok detektálására és általánosítására.

A Turing-teszt a kérdező és a számítógép közötti fizikai kölcsönhatást szándékosan kerülte, mert egy személy *fizikai* szimulációja az intelligenciához nem szükséges. Az ún. **teljes Turing-teszt (total Turing-test)** azonban videojelet is fel kell dolgozzen, hogy a kérdező tesztelni tudja az illető érzékelési képességeit, valamint tartalmazza annak a lehetőségét is, hogy a kérdező az objektumokat átadhassa „egy nyílásban keresztül”. A teljes Turing-teszt teljesítéséhez a számítógépnek szüksége lesz:

- **gépi látásra (computer vision)**, az objektumok érzékeléséhez és
- **robotikára (robotics)** az objektumok mozgatásához.

Ez a hat terület nagyjából lefedi az MI-t. Turing becsületére válik, hogy egy olyan tesztet talált ki, amely 50 év múltával is releváns maradt. Az MI-n belül a kutatók sok erőfeszítést mégsem fejtettek ki a Turing-teszt teljesítése érdekében, abban a hiedelemben, hogy a mögötte sorakozó elvek tanulmányozása fontosabb, mint egy példány duplikálása. A „mesterséges repülés” kutatása akkor járt sikerrel, amikor a Wright testvérek abbahagyta a madarak utánzását, és az aerodinamikát kezdték tanulmányozni. A repüléssel foglalkozó könyvek nem azt a célt fogalmazzák meg, hogy olyan „gépeket kellene építeni, amelyek annyira hasonlóan repülnek a galambokhoz, hogy ezzel akár más galambokat is képesek megtéveszteni”.

## **Emberi módon gondolkodni: a kognitív modellezés**

Ha azt szeretnénk kijelenteni, hogy egy program emberi módon gondolkodik, valamilyen módon meg kellene határoznunk, hogy az emberek hogyan gondolkodnak. Ehhez az emberi elme működési mechanizmusának *belsejébe* kellene tudnunk belénezni. Két módja van ennek: önelemzés révén – az átsuhanó gondolataink megragadásával – vagy pszichológiai kísérletekkel. Ha egyszer majd rendelkezünk az elme elegendően részletes elméletével, lehetségesse válik az elmelet számítógépes programmal való kifejezése. Ha a program bemeneti és kimeneti, valamint időzítése az emberi viselkedéssel meggyeznek, ez egyben bizonyíték arra, hogy a program bizonyos mechanizmusai feltehetően

az emberben is megtalálhatók. Így például Newell és Simon, akik az „általános problémamegoldót (General Problem Solver, GPS)” (Newell és Simon, 1961) kifejlesztették, csupán azzal, hogy a programjuk helyesen oldja meg a problémákat, nem voltak elégedettek. Jobban érdekelte őket, hogy összehasonlítsák a program következetiséi lépéseiit az ugyanazon a feladaton dolgozó emberek lépéseiivel. Az MI számítógépes modelljeit és a pszichológia kísérleti technikáit a **kognitív tudomány (cognitive science)** interdiszciplináris területe kapcsolja össze azáltal, hogy kísérletet tesz precíz és verifikálható elméletek megfogalmazására az emberi elme működéséről.

A kognitív tudomány önmagában is vonzó terület, annyira, hogy egy önálló enciklopédia létrehozásának is van értelme (Wilson és Keil, 1999). A könyvünkben meg sem kíséreljük leírni, hogy az emberi megismerésből mit tudunk már. Esetenként azonban kitérünk az MI-technikák és az emberi gondolkodás közötti hasonlóságokra és különbségekre. Az igazi kognitív tudomány – szükségszerűen – a valódi emberek, illetve állatok kísérleti kutatásán alapul, mi pedig feltételezzük, hogy az olvasó a kísérleteihez csupán számítógéppel rendelkezik.

Az MI-kutatás kezdeti szakaszában a megközelítéseket sokszor össze is keverték. Egy szerző állíthatta például, hogy ha egy algoritmus egy feladaton jól vizsgázik, *akkor* jó modellje az emberi képességeknek, és mégfordítva. A mai szerzők a kétféle igényt elkülönítik. Ez a megkülönböztetés mind az MI, mind a kognitív tudomány gyorsabb fejlődését tette lehetővé. Az MI és a kognitív tudomány folyamatosan termékenyítően hatnak egymásra, különösképpen a látás és a természetes nyelv vonatkozásában. A látás a közelmúltban különösen sokat fejlődött a neurofiziológiai eredményeket és a számítási modelleket összefogó integrált megközelítés révén.

## Racionálisan gondolkodni: a gondolkodás törvénye

Arisztotelész görög filozófus volt az elsők egyike, aki megkísérelte a „helyes gondolkodás”, azaz a megcáfoltatlan következetési folyamatok törvényekbe foglalását. Híres **szillogisztikája (syllogisms)** olyan mintákat szolgáltatott a következetési sémahez, amelyek helyes premisszákból mindenkor mindenkor helyes következményekre vezettek. Például „Szókratész egy ember; minden ember halandó; azaz Szókratész halandó.” Ezekről a gondolkodási törvényekről feltételezték, hogy az elme működését irányítják, és ezek indították el a **logika (logic)** tudományának kialakulását.

A 19. században a logikusok – a logika tudományát művelők – megadták a világ mindenennemű objektumára és az azok közötti relációkra vonatkozó állításoknak a precíz megfogalmazását. (Hasonlítsuk ezt a közönséges aritmetikai jelölésrendszerhez, ami főleg a számokon értelmezett egyenlőségek és egyenlőtlenségek kifejezésére alkalmas.) 1965-ben léteztek már olyan programok, amelyek – legalább elvben – képesek voltak *tetszőleges*, logikai jelölésekkel kifejezett problémához megadni a probléma megoldását.<sup>2</sup> Az MI-n belül uralkodó **logicista (logicist)** hagyomány azt reméli, hogy ilyen programokra alapozva intelligens rendszereket képes létrehozni.

Ennek a megközelítésnek két alapvető baja van. Először is a logikai jelölésrendszer által igényelt formális elemekkel informális tudást kifejezni nem ilyen egyszerű,

<sup>2</sup> Ha nincs megoldás, a program a keresést esetleg soha nem fejezi be.

különösen ha a tudás nem 100%-osan biztos. A másik az, hogy nagy a különbség egy probléma „elvi”, illetve gyakorlati megoldása között. Már a néhány tíz tényt kitevő problémák megoldása is kimerítheti egy tetszőleges számítógépes rendszer számítási erőforrásait, hacsak valamilyen módon nincs vezérelve, hogy melyik következetési lépésekkel próbálkozzék először. Annak ellenére, hogy ez a két probléma a számítógépes következető rendszer *bármilyen* elven történő fejlesztésénél felbukkan, először a logicista hagyományban jelent meg.

## Racionálisan cselekedni: a racionális ágens

Egy **ágens** (*agent*) nem más, mint valami, ami cselekszik (az *ágens* szó forrása a latin *agere* – cselekedni). Számítógépes ágensektől azonban elvárjuk, hogy legyenek más jellemzői is, amelyekben különböznek a „mezei” programuktól. Ilyen jellemzők például az autonóm vezérlés felügyelete cselekvés, a környezet észlelése, a hosszabb idejű tartós létezés, a változásokhoz történő adaptáció és mások céljainak az átvétele. Egy **racionális ágens** (*rational agent*) a legjobb kimenetel érdekében vagy – bizonytalanság jelenlétében – a legjobb várható kimenetel érdekében cselekszik.

A „gondolkodás törvénye” megközelítésben a hangsúly teljes egészében a korrekt következetésekben volt. Korrekt következetések meghozatala egyes esetekben a racionális ágensek részét képezi, hiszen a racionális cselekvés egyik módja, hogy logikusan következtetve olyan következményekre jutunk, hogy adott cselekvés a céljaink elérését biztosítja. A korrekt következetés azonban nem egész a racionalitásnak, mert gyakran megesik, hogy akkor is cselekedni kell, amikor bizonyíthatóan korrekt cselekvés nem létezik. A racionális cselekvésnek olyan esetei is vannak, amikor következetésnek nyoma sincs. Így például a forró kályhától a kezét elrántó reflexszerű cselekvés sikeresebb, mint a tudatos megfontolásból adódó lassabb mozdulat.

A Turing-teszthez szükséges minden képesség lényeges a racionális cselekvés megvalósítása szempontjából. A tudásreprezentálás és a következetés képessége szükséges ahhoz, hogy a helyzetek széles spektrumában helyes döntésre juthassunk. Természetes nyelven megfogalmazott érthető szövegek kimondásával egy komplex társadalomban is elboldogulunk. Tanulásra nem a művelteg kedvéért van szükség. A világ működéséről alkotott jobb elképzélések hatékonyabb stratégiákhoz vezethetnek a környezetünkkel való kapcsolat kialakításában. A vizuális érzékelés nem azért szükséges, mert látni szórakoztató dolog, hanem mert így jobb elképzelésünk lesz arról, hogy egy cselekvéssel mit érhetünk el. Egy ízletes falat látványra például segít abban, hogy tudatosan érte nyúljunk.

Emiatt az MI-nek mint racionális ágensek tervezésének a tanulmányozása két előnyt jelent. Először is ez a megközelítés a „gondolkodás törvénye” megközelítésnél általánosabb, hiszen a korrekt következetés csupán egyike a racionalitást biztosító mechanizmusoknak. Másodszor, tudományosan jobban is kezelhető, mint azok a megközelítések, amelyek az emberi viselkedésen vagy emberi gondolkodáson alapulnak, mert a racionalitás mértéke jól definiált és teljesen általános. Az emberi viselkedés viszont egy bizonyos specifikus környezethez jól adaptálódott, és részben egy olyan bonyolult, zömében nem ismert evolúciós folyamat eredménye, amely valószínűleg még messze áll a tökéletességtől. *Könyvünk tehát a racionális ágensek általános elveire és a létrehozásukhoz szükséges komponensekre koncentrálódik.* Látni fogjuk, hogy a problémamegfogalmazás



Iátszólagos egyszerűsége ellenére, hihetetlen sok következménnyel kell szembenéznünk, ha megkíséreljük a problémát megoldani. Néhány ilyen következményről részletesebben a 2. fejezet szól.

Egy fontos dologról nem szabad elfeledkezni: rövidesen meglátjuk, hogy összetett környezetben a tökéletes racionalitást – mindig helyesen cselekedni – lehetetlen elérni. A számítási szükségletek egyszerűen túl nagyok. A könyv nagyobb részében azonban azzal a munkahipotézissel fogunk élni, hogy a tökéletes döntéshozatal megértése jó kiindulópont. A probléma így egyszerűbbé válik, és megfelelő keretet nyújt a területhez tartozó alapozó anyag többségéhez. A korlátozott racionalitással (**limited rationality**) – azaz megfelelően cselekedni, miközben az összes kívánt számítás elvégzésére nincs elegendő idő – explicit módon a 6. és a 17. fejezet foglalkozik.

## 1.2. A MESTERSÉGES INTELLIGENCIA ALAPJAI

Ebben a részben azon tudományterületek rövid történetét mutatjuk be, amelyek ötleteikkel, nézőpontjukkal, módszereikkel hozzájárultak az MI-hez. Mint minden történetnél, most is csak kevés személyről és eseményről tudunk megemlékezni, másokat, akik, illetve amelyek szintén fontosak, figyelmen kívül hagyunk. Az eseményeket egy kérdéssorozat körül rendeztük sorba. Ugyanakkor azt a benyomást sem szeretnénk kelteni, mintha ezek a kérdések a terület által kizártlagosan vizsgált kérdések lennének, vagy, hogy ezek a tudományterületek célkitűzéseikben mind az MI irányába fejlődtek volna.

### Filozófia (i. e. 428-tól napjainkig)

- Lehet-e formális szabályok révén helyes konklúziókhoz jutni?
- Hogyan emelkedik ki a mentális elme a fizikai agyból?
- Honnan jön a tudás?
- A tudás hogyan vezet a cselekvéshez?

Arisztotelész (i. e. 384–322) volt az első, aki megkísérelte precízen megfogalmazni az elme racionális részén uralkodó törvényszerűségeket. A korrekt következtetéseket leíró szövegek informális rendszerét fejlesztette ki, ami elvben lehetővé tette a konklúziók gépies származtatását, ha a kezdeti premisszák egyszer adottak voltak. Sokkal később Ramon Lull (élt 1315-ig) javasolta, hogy a hasznos következtetést éppenséggel egy gépezetre rá lehetne bízni. Az általa javasolt „fogalmi kerekék” a könyv borítóján láthatók. Thomas Hobbes (1588–1679) felvetette, hogy a következtetés olyan, mint egy numerikus számítás, vagyis hogy „csendes gondolatainkban kivonunk és összeadunk”. Eközben a számítások automatizálása már a legjobb úton haladt. 1500 körül Leonardo da Vinci (1452–1519) megtervezett – bár meg nem épített – egy mechanikus kalkulátort. A kalkulátor közelmúltbeli rekonstrukciója bebizonyította, hogy a tervezet működőképes volt. Az első ismert számítógépet 1623 körül Wilhelm Schickard (1592–1635) német tudós konstruálta, bár a Blaise Pascal (1623–1662) által 1642-ben épített Pascaline ismertebb. Pascal azt írta, hogy „az aritmetikai gép olyan hatást produkál, ami közelebbinek

tűnik a gondolathoz, mint az állatok összes cselekvése". Gottfried Wilhelm Leibniz (1646–1716) egy mechanikus berendezést épített, amely nem a számokon, hanem a fogaalmakon végzett műveleteket, bár működési köre igen korlátozott volt.

Most, hogy van már elképzelésünk arról, hogy az elme formális, racionális részét milyen elvek alapján lehet leírni, a következő lépésként az elmét fizikai rendszerek tekintjük. René Descartes (1596–1650) elemzte először világosan az elme és az anyag közötti különbséget és az ebből adódó problémákat. Az elme tiszta fizikai megközelítésének egyik problémája az, hogy kevés helyet hagy a szabad akarat számára. Ha az elmét csupán fizikai törvények uralják, nincs több szabad akarata mint egy szklának, amely „eldönti”, hogy a föld középpontja felé fog esni. Annak ellenére, hogy Descartes a következtetés hatalmának szószólója volt, ő volt az is, aki a **dualizmust (dualism)** javasolta. Azt tartotta, hogy az elmnének (vagy a léleknek) van olyan része, amely nem része a természetnek és nem alanya a fizikai törvényeknek. Ezzel szemben úgy érezte, hogy az állatok nélkülözik ezt a dualista természetet, lényegében gépeknek tekintendők. A dualizmus alternatívája a **materializmus (materialism)**, amely azt tartja, hogy éppen az agynak a fizikai törvények szerinti működése *valósítja meg* az elmét. A szabad akarat nem más, mint az a mód, ahogyan a választási lehetőségek észlelése a döntési folyamat számára megjelenik.

Ha az elme a benne levő ismereteken operáló fizikai eszköz, akkor a következő probléma a tudás forrásának a meghatározása. Francis Bacon (1561–1626) *Novum Organum*<sup>3</sup> induló empiricista (*empiricist*) mozgalmat John Locke (1632–1704) kijelentése jellemzi: „Nincs semmi a megértésben, ami előbb ne létezne az érzékszervekben.” David Hume (1711–1776) *A Treatise of Human Nature* c. művében (Hume, 1739) javasolja azt a fogalmat, amit ma **indukciónak (induction)** nevezünk, nevezetesen hogy az általános elveket a komponenseik ismétlődő kapcsolataiból emeljük ki. Ludwig Wittgenstein (1889–1951) és Bertrand Russell (1872–1970) munkájára alapozva a Rudolf Carnap (1891–1970) vezette híres Bécsi Kör kialakította a **logikai pozitivizmus (logical positivism)** doktrínáját. Ez a doktrína azt állítja, hogy minden tudást végső soron az érzékszervi bemeneteknek megfelelő **megfigyeléses állításokon (observation sentences)** alapuló logikai elmélétekkel meg lehet magyarázni.<sup>4</sup> Rudolf Carnap és Carl Hempel (1905–1997) **igazoló elmélete (confirmation theory)** megkísérletre megvilágítani, miként gyarapítja a tudást a tapasztalat. Carnap könyve, a *The Logical Structure of the World* (Carnap, 1928) a tudás elemi tapasztalatból történő kinyerésére egy explicit számítási eljárást definiált. Ez volt talán az első olyan elmélet, amely az elmét számítási folyamatként ábrázolta.

Az elme filozófiai képének utolsó eleme a tudás és a cselekvés kapcsolata. Az MI szempontjából ez a kérdés elsődleges fontosságú, mert az intelligencia cselekvést is és következtetést is igényel. Továbbá ahhoz, hogy világos legyen előttünk, hogyan kell egy olyan ágenst építeni, amelynek cselekvései jogosak vagy racionálisak lesznek, meg kell értenünk, hogy hogyan igazolhatók a cselekvések. Arisztotelész a cselekvések azért tartotta jogosnak, mert a célok és a cselekvések kimenetelei között logikai kapcsolat lehető fel (az alábbi idézet záró része a könyv borítóján is megtalálható):

<sup>3</sup> Az Arisztotelész-féle *Organon*, azaz a gondolkodás eszközének egy újabb változata.

<sup>4</sup> Ehben a megközelítésben, megvizsgálva a szavak értelmét illetve kísérleteket végezve, vagy igazolni, vagy cáfolni lehet minden értelmes állítást. Mivel ez – a szándéknak megfelelően – a metafizika többségét kizára, a logikai pozitivizmus bizonyos körökben nem volt népszerű.

De hogyan lehetséges, hogy a gondolatot néha követi cselekvés, és néha nem, néha mozgással és néha anélkül? Úgy tűnik, mintha ugyanaz a doleg történne itt is, mint ami a változatlan objektumokra vonatkozó következtetések meghozatalánál. De ebben az esetben az eredmény egy spekulatív állítás ... holott itt a premisszából következő konklúzió egy cselekvés. ... Kellene egy ruha, a köpeny egy ruha. Kell egy köpeny. Ami kell, azt elő kell állítanom, és kell egy köpeny. Elő kell állítanom egy köpenyt. A konklúzió tehát, hogy „elő kell állítanom egy köpenyt” nem más, mint egy cselekvés. (Nussbaum, 1978, 40. o.)

A *Nicomachean Ethics*ben (III. könyv, 3, 1112b) Arisztotelész a téma tovább folytatja, egy algoritmust javasolva:

Nem a célokat, hanem az eszközöket mérlegeljük. Az orvos nem azt fontolatja, hogy gyógyítson-e, a szónok sem, hogy meggyőző-e, az államférfi sem azon hézitál, hogy jogot és törvényt teremtsen, és senki sem a célt mérlegeli. mindenki feltételezi a célt, és azt fontolatja, hogy az hogyan, milyen eszközökkel érhető el, és hogy azáltal könnyen és a lehető legjobban érhető-e el. És ha valamelyen eszközzel elérhető a cél, azt fontolatjuk, hogy azzal az eszközzel *hogyan*, milyen módon érhetjük el, amíg a legsébő okhoz el nem jutunk. Ez az ok a selfedezés sorában a legutolsó... és ami az elemzés rendjében az utolsó, a megtörténés rendjében elsőnek tűnik. És ha lehetetlenségezhet jutunk, a keresést feladjuk, például ha pénzre van szükségünk, azt viszont megszerezni nem lehet. Ha azonban valami lehetségesnek tűnik, akkor megpróbálkozunk vele.

Arisztotelész megközelítését 2300 évvel később Newell és Simon a GPS-programjukban implementálták. Ma ezt regressziós tervkészítő rendszernek neveznénk (lásd 11. fejezet).

A céllapú analízis hasznos, de nem segít akkor, amikor a célt több cselekvéssel is el lehet érni, vagy pedig, ha annak eléréséhez egy cselekvés sem elegendő. Antoine Arnauld (1612–1694) helyesen adta meg azt a kvantitatív formulát, hogy ilyen esetekben milyen cselekvéshez kellene folyamodni (lásd 16. fejezet). John Stuart Mill (1806–1873) *Utilitarianism* c. művében (Mill, 1863) a racionális döntés kritériumának a gondolatát az emberi cselekvés minden aspektusára kiterjesztette. A döntéshozatal egy jobban formalizált elméletéről a következő részben lesz szó.

## **Matematika (kb. 800-tól napjainkig)**

- Melyek az érvényes következtetések formális szabályai?
- Mi az, ami kiszámítható?
- Hogyan vagyunk képesek a bizonytalan információ alapján következtetéseket levonni?

A filozófusok számba vették az MI fontosabb ötleiteit, de ahhoz, hogy a formális tudományra rátérhessünk, szükséges volt legalább három alapvető területnek – a logikának, a számítástechnikának és a valószínűségnak – bizonyos szintű matematikai formalizálása.

A formális logika elképzelése egészen az ókori Görögország filozófusaihoz nyúlik vissza (lásd 7. fejezet), de a matematikai kialakulása George Boole (1815–1864) munkásságával kezdődött igazán, aki részleteiben kidolgozta az ítélet- vagy Boole-logikát

(Boole, 1847). 1879-ben Gottlob Frege (1848–1925) kiterjesztette a Boole-logikát, hogy az objektumokat és relációkat is tartalmazhasson. Ezzel megalkotta azt az elsőrendű logikát, amely lényegében manapság a legalapvetőbb tudásreprezentációs rendszer.<sup>5</sup> Alfred Tarski (1902–1983) egy referenciaelméletet vezetett be, amely azt mutatja meg, hogy hogyan rendelhetők össze a logika és a valós világ objektumai. A következő lépés annak a kiderítése volt, hogy hol vannak a logikának és a számításoknak az alkalmazási határai.

Az első nem triviális algoritmusnak (**algorithm**) a legnagyobb közös osztót kiszámító euklideszi algoritmust tekintik. Az algoritmusok önmagában való tanulmányozása al-Hvárizmi, a 9. századi perzsa matematikus idejéig nyúlik vissza, ainek a művei az arab számokkal és az algebrával is megismerték Európát. Boole és mások a logikai dedukció algoritmusával foglalkoztak, a 19. század végén pedig már erőfeszítések történtek az általános matematikai érvelés logikai dedukcióként való formalizálására. 1900-ban David Hilbert (1862–1943) 23 problémából álló listát publikált, amelyről – helyesen – azt vélte, hogy matematikus társait a század nagy részében le fogja kötni. Utolsó problémája azt firtatta, vajon létezik-e a természetes számokra vonatkozó tetszőleges logikai állítás igazságát eldöntő algoritmus. Ez a híres ún. *Entscheidungsproblem* (eldönthetőségi probléma). Hilbert lényegében azt a kérdést tette fel, hogy a hatékony bizonyítási eljárásoknak vannak-e elvi korlátjai. Kurt Gödel (1906–1978) 1930-ban megmutatta, hogy a Frege és Russell-féle elsőrendű logikában létezik hatékony eljárás tetszőleges igaz állítás bizonyítására. Az elsőrendű logika azonban nem elegendő a matematikai indukció kifejezéséhez, ez utóbbi viszont a természetes számok jellemzéséhez szükséges. 1931-ben Gödel meg is mutatta, hogy valóban léteznek elvi korlátok. **Nemteljességi tétele** (*incompleteness theorem*) kimutatja, hogy minden, a természetes számok tulajdonságait kifejezni képes nyelvben léteznek igaz, de abban az értelemben eldönthetetlen állítások, hogy az igazságuk algoritmikusan nem mutatható ki.

Ez az alapvető eredmény úgy is interpretálható, hogy léteznek az egész számokon értelmezhető olyan függvények, amelyek algoritmikusan nem írhatók le, magyarán nem számíthatók ki. Ez motiválta Alan Turingot (1912–1954), hogy megkísérítse a kiszámítható függvények pontos jellemzését. Ez a megfogalmazás kissé problematikus, mert egy számítási procedúrának vagy egy hatékony eljárásnak a formális definícióját valójában lehetetlen megadni. Általánosan elfogadott azonban, hogy a Church–Turing-tézis, amely azt állítja, hogy a Turing-automata (Turing, 1936) képes minden kiszámítható függvényt kiszámítani, elegendő definíciót szolgáltat. Turing azt is kimutatta, hogy vannak olyan függvények, amelyeket semmilyen Turing-automata nem képes kiszámítani. Így például nincs olyan automata, amely általánosságban el tudná dönteni, hogy egy adott program egy adott bemenetre választ fog-e adni, vagy pedig örökké futni fog.

Ámbár az eldönthetetlenség és a kiszámíthatóság fontos fogalmak a számítások megértéséhez, a kezelhetetlenségnek (**intractability**) sokkal nagyobb hatása volt. Durván fogalmazva a problémák egy osztályát kezelhetetlennek mondjuk, ha az egyes problémapéldányok megoldásához szükséges idő legalább exponenciálisan nő a problémapéldányok nagyságától függően. A bonyolultság polinomiális és exponenciális növekedése közötti különbségre először az 1960-as évek közepén hívták fel a figyelmet

<sup>5</sup> Az elsőrendű logika Frege javasolta felírása soha nem terjedt el. A nyilvánvaló okát könnyen megérjük, ha a borítón lévő példára pillantunk.

(Cobham, 1964; Edmonds, 1965). A kétfajta növekedés közötti különbség fontos, mert az exponenciális növekedés következménye, hogy közepes nagyságú problémapéldányokat sem lehet értelmes időkorláton belül megoldani. Az intelligens viselkedés generálásának általános problémáját tehát inkább kezelhető, mint kezelhetetlen részproblémákra kellene lebontanunk.

Hogyan lehet egy kezelhetetlen problémát felismerni? A Stephen Cook és Richard Karp által útjára indított **NP-teljesség (NP-completeness)** elmélete megadja rá a választ (Cook, 1971; Karp, 1972). Cook és Karp a kanonikus kombinatorikus keresési és következtetési problémák nagy osztályára belátta, hogy azok NP-teljesek. Bármilyen problémaosztály, amire egy NP-teljes problémaosztály visszavezethető, minden bizonytalán kezelhetetlen lesz (és bár még senkinek sem sikerült bebizonyítani, hogy az NP-teljes problémák szükségszerűen nem kezelhetők, kevés tudós van más véleményen). Ezek az eredmények élesen elütnek a számítógép megjelenését kísérő „elektromos szuperagy – Einstein-nél gyorsabb” lelkesedéstől. Az egyre növekvő számítási sebesség ellenére az intelligens rendszerekre minden jellemző lesz az erőforrások óvatos és kényes használata. Durván fogalmazva, a világ egy *rendkívül* nagy problémapéldány! Az utóbbi években az MI segített megérteni, hogy az NP-teljes problémák egyes példányai miért nehezek, holott mások könnyűnek bizonyulnak (Cheeseman és társai, 1991).

A logikán és a számításokon túlmenően a matematika harmadik nagy hozzájárulása az MI-hez a **valószínűség-elmélet (probability)**. A valószínűség gondolatát – a szerencsejátékok lehetséges kimeneteivel leírva – először az olasz Girolamo Cardano (1501–1576) fogalmazta meg. A valószínűség hamarosan felbecsülhetetlen komponense lett az összes kvantitatív elmeletnek, segítséget adva a bizonytalan mérések és hiányos elmeletek kezeléséhez. Pierre Fermat (1601–1665), Blaise Pascal (1623–1662), James Bernoulli (1654–1705), Pierre Laplace (1749–1827) és mások az elmeletet tovább fejlesztették, és új statisztikai módszereket vezettek be. A valószínűségek felfrissítési szabályát – új tények hatására – Thomas Bayes (1702–1761) javasolta. A Bayes-szabály és az azon alapuló Bayes-analízis az MI-rendszerekben alkalmazott bizonytalan következetés korszerű megközelítésének az alapja.

## Gazdaságtan (1776-tól napjainkig)

- Hogyan kell döntenünk, hogy a hasznunk maximális legyen?
- Mit kell tennünk, ha mások esetleg nem segítőkészek?
- Hogyan kell döntenünk, ha a haszonhoz csak a távoli jövőben jutunk el?

A közgazdaságtan tudománya 1776-ban kapott indító lökést, amikor is Adam Smith skót filozófus (1723–1790) megjelentette az *An Inquiry into the Nature and Causes of the Wealth of Nations* c. művét. Amíg az ókori görögök és mások a közgazdasági gondolkodáshoz járultak hozzá, addig Smith volt az első, aki a gazdaságtant tudományként kezelte, a gazdaságokat a saját gazdasági jólétet maximálizáló különálló ágensekből állóknak képzelve. Az emberek többsége azt véli, hogy a gazdaság lényege a pénz. A közgazdák azonban azt fogják állítani, hogy ők igazából azt tanulmányozzák, hogy az emberek hogyan döntenek, hogy kívánatos eredményekhez jussanak. A „kívánatos eredmények”,

avagy a **hasznosság (utility)** matematikai kezelését elsőként Léon Walras (1834–1910) formalizálta. Ezt fejlesztette tovább Frank Ramsey (Ramsey, 1931) és később a *The Theory of Games and Economic Behavior* c. művében Neumann János és Oskar Morgenstern (Neumann és Morgenstern, 1944).

A **döntéselmélet (decision theory)**, amely a valószínűség-elméletet és a hasznossági elméletet kapcsolja össze, a bizonytalanság melletti (gazdasági vagy egyéb) döntéshozatalhoz ad formális és teljes keretet – azaz azokban az esetekben, amikor a valószínűségi leírás megfelelően írja le a döntéshozó környezetét. Ez a megközelítés a „nagy” gazdaságokhoz jó, ahol az ágenseknek nem kell más ágensek (mint entitások) cselekvéseit figyelembe venni. „Kis” gazdaságokban a helyzet inkább egy **játékra (game)** hasonlít: az egyik játékos cselekvései lényeges befolyással lehetnek a másik játékos hasznosságára (akár pozitív, akár negatív értelemben). A Neumann és Morgenstern által kifejlesztett **játékelmélet (game theory)** (lásd még Luce és Raiffa, 1957) olyan meglepő eredményeket is tartalmaz, miszerint az egyes játékokban a racionális ágensnek véletlenszerűen kell cselekednie, vagy legalábbis olyan módon, hogy az az ellenfelek számára véletlennek tűnjön.

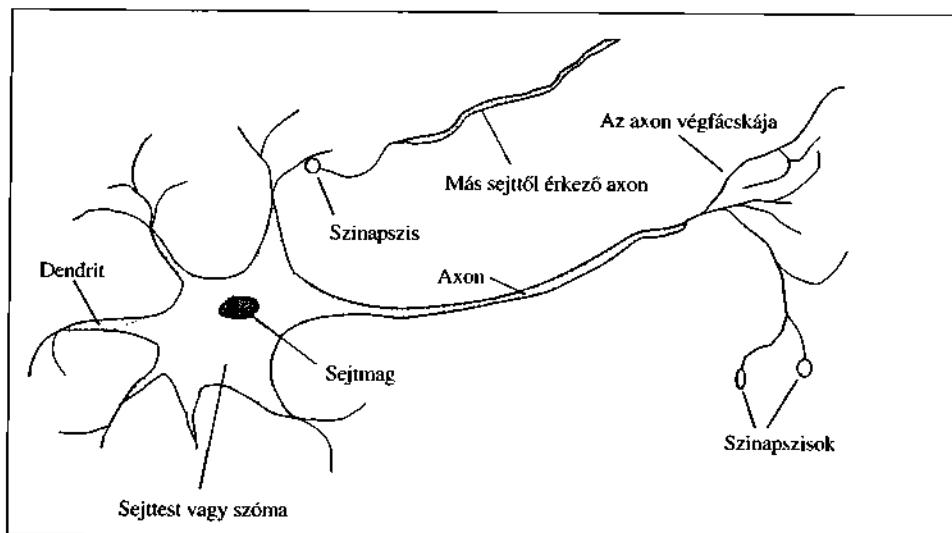
Harmadik kérdésünkkel, vagyis, hogy hogyan kell racionálisan döntenni, ha a haszon nem azonnali, hanem több egymást követő cselekvés sorozatának az eredménye, a közgazdászok többsége nem foglalkozott. Ezt a feladatot az **operációkutatás (operational research)** területén vizsgálták, mely Angliában látott napvilágot, amikor a II. világháborúban a radartelepítést igyekezték optimalizálni, és amely később a komplex menedzsment döntéshozatal területén talált polgári alkalmazásokra. Richard Bellman (Bellman, 1957) a **Markov döntési folyamatoknak (Markov decision processes)** nevezett szekvenciális döntési problémákat formalizálta. Mi ezeket a 17. és 21. fejezetben tanulmányozzuk.

A közgazdaságtan és az operációkutatás nagyban hozzájárult a racionális ágens fell fogásunkhoz, mégis az MI-kutatás évekig egészen más pályán haladt. Az egyik ok a racionális döntéshozatal látszólag igen nagy **komplexitása (complexity)** volt. Herbert Simon (1916–2001), az MI-kutatás egyik úttörője azzal nyerte meg 1978-ban a közgazdasági Nobel-díjat, hogy egy korai munkájában kimutatta, a **kielégítő döntéshozatalon (satisfying)**, azaz az „elegendően jó” döntésekben alapuló modellek a tényleges emberi viselkedés jobb leírói, mint azok, amelyek hosszadalmas számításokkal meghatározott optimális döntések eredményeképp születtek (Simon, 1947). Az 1990-es években az ágensrendszerek területén a döntéselméleti technikák iránt fokozott érdeklődés mutatkozott (Wellman, 1995).

## **Neurális tudományok (1861-től napjainkig)**

- Hogyan dolgozza fel az információt az agy?

A **neurális tudományok (neuroscience)** az idegrendszer, de különösen az agyat tanulmányozzák. Annak egzakt módja, hogy hogyan keletkeznek a gondolatok az agyban, a tudomány egyik nagy feladványa. Évezredek óta tudjuk, hogy az agy valahogyan részes a gondolatnak, hiszen egy erős fejbeütés a mentális funkciók károsodásához vezethet. Hosszú ideje azt is tudjuk, hogy az emberi agy valahogyan más. I. e. 335-ben



**1.2. ábra.** Az idegsejt vagy neuron részei. minden neuron sejttestből vagy szőmából áll, amely sejtmagot tartalmaz. A sejttestből ágazik el néhány dendritnek nevezett és egy hosszú, axonnak nevezett szál. Az axon jó messzire elér, sokkal messzebbre, mint ahogy ezt a jelen ábra sugallja. Az axonok tipikusan 1 cm hosszúak (ez a sejttest átmérőjének 100-szorosa), de elérhetik az 1 m-t is. Egy neuron 10–100 000 más neuronnal tart kapcsolatot a szinapszisoknak nevezett csatlakozásokon keresztül. A jeleket neuronról neuronra egy összetett elektrokémiai reakció továbbítja. A jelek rövid távon szabályozzák az agy aktivitását, hosszú távon pedig befolyásolják a neuronok helyzetét és a kapcsolódási rendszertük. Úgy tartják, e mechanizmusok képezik az agyi tanulás alapját. Az információfeldolgozás zöme az agykéregben történik, ami az agy külő rétege. A feldolgozás szervezeti egysége, úgy tűnik, egy kb. 0,5 mm-es átmérőjű szövetoszlop, amelynek magassága az egész kéregre kiterjed, ez az embereknél kb. 4 mm. Egy oszlop kb. 20 000 neuront tartalmaz.

Arisztotelész azt írta, hogy „az összes állat közül méretéhez képest az ember agya a legnagyobb”.<sup>6</sup> Ennek ellenére csak a 18. század közepétől ismerték fel széles körben, hogy a tudat az agyban lakozik. Korábbi jelöltek között szerepelt a szív, a lép és a tobozmirigy.

Paul Broca (1824–1880) afázia (beszédzavar) kutatása agysérült pácienseknél 1861-ben lökést adott e területnek, és meggyőzte az orvosi szakmát, hogy az agynak léteznek lokálizált, konkrét kognitív funkciókért felelős részei. Konkrétan azt mutatta ki, hogy a beszéd előállítása a bal félteke, most Broca-féle területnek nevezett részére korlátozódik.<sup>7</sup> Ekkor tudták már, hogy az agy idegsejtekből – **neuronokból (neuron)** – áll, de 1873-ig, amikor Camillo Golgi (1843–1926) kifejlesztett egy festési módszert, nem tudták az agyban az egyedi neuronokat megfigyelni (lásd 1.2. ábra). Ezt a módszert alkalmazta Santiago Ramon y Cajal (1852–1934) az agy neurális struktúráit feltáró úttörő munkájában.<sup>8</sup>

<sup>6</sup> Azóta sikerült felfedezni, hogy az egyes delfin- és cetfajtáknál az agy relatív nagysága még nagyobb. Most úgy tartják, hogy az emberi agy tekintélyes nagyságát hűtőrendszerének fejlettisége teszi lehetővé.

<sup>7</sup> Sarkan Alexander Hoodot (Hood, 1824) idézik, mint lehetséges elsődleges forrást.

<sup>8</sup> Golgi kitartott abban a hiedelmében, hogy az agyi funkciók egy folytonos közegben valósulnak meg, amibe a neuronok be vannak ágyazva. Ezzel ellentétben Cajal a „neurondoktrinát” képviselte. 1906-ban mindketten (megosztott) Nobel-díjban részvültek, de a díj átvételekor meglehetősen ellenséges hangnemű beszédeket tartottak.

Ma már vannak adataink arról, hogy az agy egyes területei és az általuk kontrollált, vagy a szenzorikus bemeneteket generáló test részei között milyen leképezések vannak. Az ilyen leképezések hetek alatt képesek drasztikus változásokra, ráadásul egyes állatoknál úgy tűnik, többszörös leképezéseket találhatunk. Azt azonban még nem teljesen értjük, hogy az egyes területek miként veszik át más részek funkcióit, ha azok megsérülnek. Az egyedi emlékek tárolásáról kis hiján semmilyen elméletünk nincs.

Az ép agy aktivitásának mérése 1929-ben kezdődött, amikor Hans Berger kidolgozta az elektroenzefalogramma (EEG) elvét. A funkcionális mágneses rezonancia (fMRI) közelműltbeli kifejlesztése (Ogawa és társai, 1990) eddig nem látott részletességgel tárja az idegek kutatók elő az agy aktivitását, olyan méréseket is lehetővé téve, amelyek érdekes módon korrelálnak a zajló kognitív folyamatokkal. Az ismereteket kiegészítí az egyedi sejt szintjén történő neuronaktivitás rögzítésének fejlődése. A nagymértékű fejlődés ellenére még messze vagyunk attól, hogy megértsük, hogy e kognitív folyamatok bármelyike valójában hogyan is működik.

	Számítógép	Emberi agy
Számítási egységek	$1 \text{ CPU} \cdot 10^8 \text{ kapu}$	$10^{11} \text{ neuron}$
Tárolóegységek	$10^{10} \text{ bit RAM}$	$10^{11} \text{ neuron}$
	$10^{11} \text{ bit diszk}$	$10^{14} \text{ szinapszis}$
Ciklusidő	$10^{-9} \text{ mp}$	$10^{-3} \text{ mp}$
Sávszélesség	$10^{10} \text{ bit/mp}$	$10^{14} \text{ bit/mp}$
Memoriámodosítások száma másodpercenként	$10^9$	$10^{14}$

**1.3. ábra.** A számítógépek számítási erőforrásainak (2003 körül) és az agy erőforrásainak durva összehasonlítása. A könyv első kiadása óta a számítógépekre jellemző számok legalább egy 10-es szorzóval megnöttek, és várható, hogy ez a növekedés ebben az évtizedben is folytatódik. Az agyra jellemző számok az utóbbi 10 000 évben nem változtak.

Az igazán csodálatos észrevétel az, hogy *egyszerű sejtek gyűjteménye elvezethet a gondolathoz, a cselekvéshez és a tudathoz*, vagy más szavakkal: az *agytól eljutunk az elnéhez* (Searle, 1992). Az egyedüli elvi alternatíva a miszticismus: hogy létezik egy misztikus világ, ahol megvalósul az elme a fizikai tudományok hatáskörén túl.

Az agy és a számítógép igen eltérő feladatakat végeznek és eltérő tulajdonságokkal rendelkeznek. Az 1.3. ábra azt mutatja, hogy az emberi agyban tipikusan 1000-szer több neuron van, mint ahány kapu található egy korszerű számítógép processzorában. A Moore-törvény<sup>9</sup> alapján meg lehet járni, hogy egy processzorban lévő logikai kapuk száma az agyi neuronok számát 2020 körül éri el. Az ilyen jóslatokból persze vajmi kevés következetésre lehet jutni. Ráadásul a tárolókapacitásbeli különbség eleinte szép a kapcsolási sebességen és a párhuzamosságban mutatkozó különbségekhez képest. Egy számítógépes chip nanomásodperceken alatt hajt végre egy utasítást, a neuronok ennél milliószor lassabbak. Az agy ennél többet is kompenzálg. minden neuron és szinapszis egyidejűleg aktív, holott a legkorszerűbb számítógépeknek is csak egy vagy

<sup>9</sup> A Moore-törvény azt állítja, hogy az egy négyzetcentiméterre eső tranzisztorok száma minden 1–1,5 évben megduplázzódik. Az emberi agy kapacitása durván 2–4 millió évenként duplázzódik.



legfeljebb néhány processzora van. Így annak ellenére, hogy a számítógép a nyers kapcsolási sebességben milliószor gyorsabb, az agy végeredményben százezer-szer gyorsabban oldja meg a feladatait.

## Pszichológia (1879-től napjainkig)

- Hogyan gondolkognak és cselekszenek az emberek és az állatok?

Azt mondhatjuk, hogy a tudományos pszichológia Hermann von Helmholtz német fizikus (1821–1894) és tanítványa, Wilhelm Wundt (1832–1920) munkásságával kezdődött. Helmholtz tudományos módszereket alkalmazott az emberi látás tanulmányozásánál, és a *Handbook of Physiological Optics* c. művét még ma is úgy tekintik, mint „az emberi látás fizikájának és fiziológiának máig egyedüli legfontosabb tanulmányát” (Nalwa, 1993, 15. o.). Wundt 1879-ben a Lipcsei Egyetemen nyitotta meg a kísérleti pszichológia első laboratóriumát. Wundt ragaszkodott azon kísérletek szigorú ellenőrzéséhez, amelyekben munkatársai érzékelési, illetve asszociációs feladatokat végeztek, hogy közben a kísérletet végzők gondolkodási folyamatait is meg lehessen figyelni. A gondos feltügylet révén a pszichológia inkább tudománnyá vált, de az adatok szubjektív volta miatt valószínűtlen volt, hogy egy kísérletező képes legyen valaha is a saját elméleteit megcáfolni. Ezzel szemben az állati viselkedést tanulmányozó biológusok nem önmagukra vonatkozó adatokkal dolgoztak, és egy objektív módszertant dolgoztak ki, amit H. S. Jenkins (Jenkins, 1906) a *Behavior of Lower Organisms* c. nagy hatású műve mutat be. Ezt a nézőpontot az emberekre kivetítve a John Watson (1878–1958) vezette **behaviorista** mozgalom (**behaviorism**) a mentális folyamatok minden elméletét elutasította, azzal érvelve, hogy önelemzésből lehetetlen megbízható bizonyítékot szerezni. A behavioristák ragaszkodtak ahhoz, hogy az állatot erő érzést (vagy *ingert*) és az eredményül kapott cselekvést (másképpen *választ*) szigorúan objektív mércék szerint tanulmányozzuk. Olyan gondolati konstrukciókat, mint a tudást, a hiedelmeket, a célokat és a következetes lépéseiit elutasították, áltudományos, „népi pszichológiának” tartva azokat. A behaviorizmus eredményesnek bizonyult a patkányoknál és a galamboknál, de az emberek megértésében kevésbé volt sikeres. A pszichológiára azonban – különösképpen 1920 és 1960 között az Egyesült Államokban – erős hatást gyakorolt.

Az a **kognitív pszichológiát** (**cognitive psychology**) jellemző alapvető nézet, miszerint az agy egy információfeldolgozó eszköz, egészen William James (1842–1910)<sup>10</sup> munkáihoz vezethető vissza. Helmholtz is ragaszkodott ahhoz az elképzeléshez, hogy az érzékelésben egy tudat alatti logikai következetésnek is van szerepe. A kognitív álláspontot az Egyesült Államokban háttérbe szorította a behaviorizmus, de a Frederic Bartlett (1886–1969) vezette Cambridge-i Alkalmazott Pszichológiai Intézetben (Cambridge's Applied Psychology Unit) a kognitív modellezés virágzhatott tovább. Kenneth Craiknek, aki Bartlett hallgatója, majd utódja volt, a *The Nature of Explanation* c. műve (Craik, 1943) határozottan visszaállította az olyan „mentális” fogalmak legitimitációját, mint a hiedelmek és a célok, azzal érvelve, hogy ezek ugyanúgy tudományos

<sup>10</sup> William James Henry James író testvére volt. Azt mondják, hogy Henry úgy írt irodalmat, mintha pszichológia lenne, William viszont a pszichológiát írta úgy, mintha irodalom lenne.

fogalmak, mint a nyomás vagy a hőmérséklet, amikor gázokról beszélünk, miközben tudjuk, hogy a gázok molekulákból állnak, és azoknak sem nyomásuk, sem hőmérsékletük nincs. Craik megfogalmazta a tudásalapú ágens működésének három kulcsfontosságú lépését: (1) az ingert egy belső reprezentációra le kell fordítani, (2) e reprezentációt kognitív folyamatok manipulálják, és új belső reprezentációkat származtatnak belőle, amelyek (3) ismét cselekvésre fordítódnak vissza. Craik világosan megmagyarázta, hogy ez a megközelítés miért jó elgondolása egy ágensnek:

Abban az esetben, ha egy szervezet magában hordja a külső valóság és a saját lehetséges cselekvéseinek „kisméretű modelljeit”, képes különféle alternatívákat kipróbálni, a számára legjobb mellett dönteni, a jövőbeli helyzetekre azok bekövetkezése előtt reagálni, a múltbeli események ismeretét a jelen és a jövő kezelésében felhasználni, és a felmerülő szükséghelyzetekre minden vonatkozásban kimerítőbb, biztonságosabb és kompetensebb módon reagálni. (Craik, 1943)

Miután Craik 1945-ben egy biciklibalesetben meghalt, munkáját Donald Broadbent folytatta. Műve, a *Perception and Communication* (Broadbent, 1958) a pszichológiai jelenségek néhány első információfeldolgozó modelljét írja le. Közben, az Egyesült Államokban, a számítógépes modellezés fejlődése elvezetett a kognitív tudomány (cognitive science) kialakulásához. Ez a terület mondhatni egy tudományos találkozóval indult 1956 szeptemberében az MIT-n. (Látni fogjuk, hogy ez éppen két hónappal később követte azt a konferenciát, ahol az MI „megszületett”.) A találkozón George Miller a *The Magic Number Seven*, Noam Chomsky a *Three Models of Language*, Allen Newell és Herbert Simon pedig a *The Logic Theory Machine* c. előadásokat tartották. Ez a három befolyásos előadás azt mutatta, hogy a számítógépes modelleket hogyan lehet bevetni a memória, a nyelv és a logikai gondolkodás pszichológiájába. Manapság a pszichológusok közt teljesen elfogadott, hogy „a kognitív elméletnek olyannak kell lennie, mint egy számítógépes programnak” (Anderson, 1980). Ezen azt értik, hogy az elméletnek egy kognitív funkció megvalósításához az információfeldolgozó mechanizmust részletesen le kell írnia.

## Számítógépes tudományok (1940-től napjainkig)

- Hogyan lehet hatékony számítógépet építeni?

A mesterséges intelligencia sikeréhez két doleg szükséges: intelligencia és valamilyen mesterségesen létrehozott termék, „műtermék”. A választott műtermék a számítógép lett. A korszerű digitális számítógépet, majdnem egy időben és egymástól függetlenül, három, a II. világháborúban harcoló országban találták fel. Az első működőképes számítógép az elektromechanikus Heath Robinson<sup>11</sup> volt, amit Alan Turing munkacsoportja 1940-ben épített azzal a kizárolagos céllal, hogy a német üzeneteket dekódolhassák, 1943-ban ugyanez a csapat fejlesztette ki a Colossust, egy elektroncsöves, nagy teljesítményű, általános célú gépet.<sup>12</sup> Az első működőképes programozható számítógép

<sup>11</sup> Heath Robinson karikatúrareajzoló volt, aki a minden nap feladatakhöz, mint például a vajaskenyér-kenéshez, kitalált szeszélyes és abszurd módon bonyolult szerkezetetől volt híres.

<sup>12</sup> A háború után Turing szerette volna ezeket a számítógépeket az MI-kutatáshoz – például az egyik legelső sakkprogramhoz (Turing és társa, 1953) – felhasználni. Erőfeszítéseinek azonban a brit kormány vetett gátat.

a Z-3, a német Konrad Zuse 1941-ből származó találmánya volt. A lebegőpontos számokat is Zuse találta ki, és ő volt az, aki az első magas szintű programozási nyelvet, a Plankalkült kifejlesztette. Az Egyesült Államokban az első elektronikus számítógépet, az ABC-t, John Atanasoff és végzős hallgatója, Clifford Berry állították össze 1940 és 1942 között az Iowai Állami Egyetemen. Munkájukat nemigen támogatták, elismerést sem kaptak érte. A mai számítógép előfuturai közül a legnagyobb hatású gép az ENIAC volt, amelyet egy titkos katonai projekt részeként John Mauchly és John Eckert részvételével a Pennsylvaniai Egyetem egy csoportja fejlesztett ki.

Az azóta eltelt fél évszázad során a számítógéphardver minden újabb generációja sebességben és kapacitásban növekedést, árban esést hozott. A teljesítőképesség mintegy 18 havonta megduplázódik, és ez így lehet még egy vagy két évtizedig. Utána molkuláris gépekre vagy valamilyen más technológiára lesz szükségünk.

Természetesen voltak számítóberendezések az elektronikus számítógépek előtt is. A 17. századból származó legkorábbi automatizált gépekről már a filozófiánál volt szó. Az első programozható gép Joseph Marie Jacquard 1805-ben felfalált szövőszéke volt, amely lyukaszott kártyákon tárolta a szövőmintához szükséges utasításokat. A 19. század közepe táján Charles Babbage (1792–1871) két gépet tervezett, de egyiket sem fejezte be. A könyvünk borítóján is látható Difference Engine rendeltetése matematikai táblázatok számítása lett volna mérnöki és tudományos feladatokhoz. Végül is ezt a gépet 1991-ben megépítették, és a londoni Science Museumban működés közben bemutatták (Swade, 1993). A Babbage-féle Analitikus Gép (Analytical Engine) sokkal ambiciozabb terv volt. Címezhető memoriájával, tárolt programjával és feltételes ugrásaival ez volt az első, univerzális számításokra alkalmas műtermék. Ada Lovelace, Babbage kolléganője, Lord Byron költő lánya volt talán a világ első programozója. (Az Ada programozási nyelv róla kapta a nevét.) Ada programokat írt a befejezetlen Analitikus Gépre. sőt azon is spekulált, hogy a gép sakkjátéakra vagy zenekomponálásra is képes lehet.

Az MI a számítógép-tudományok szoftveroldalának is adósa, hiszen ez szolgáltatja az operációs rendszereket, a programozási nyelveket és a korszerű programok írásához szükséges eszközöket (és a róluk szóló cikkeket). Ez azonban egy olyan terület, ahol az adósságot törlesztették is. Az MI-kutatásokban megjelent sok ötlet visszakerült a számítógépes tudományok „fő áramlatába”, beleértve az időosztásos (time-sharing) operációs rendszereket, az interaktív értelmezőket, a személyi számítógépeket ablakokkal és egérrel, a gyors fejlesztést lehetővé tevő fejlesztői környezeteket, a láncolt listás adatszerkezeteket, az automatikus tárolókezelést és az objektumorientált, szimbolikus, funkcionális és dinamikus programozás kulcsfontosságú fogalmait.

## Irányításelmélet és kibernetika (1948-tól napjainkig)

- Hogyan működhet egy műtermék a saját irányítása mellett?

Az első önszabályozó gépeket – egy vízi órát, amely olyan szabályozóval volt ellátva, hogy a víz átfolyását konstans, megjósolható értéken tartotta – az alexandriai Ktesibios (i. e. kb. 250) építette. Ez a találmány megváltoztatta annak definícióját, hogy egy műtermék mire képes. Korábban csak az élő dolgokról tartották, hogy képesek a viselkedésüket a környezeti változások hatására módosítani. Az önszabályozó vissza-

csatolt szabályozó rendszerek további példái a James Watt (1736–1819) építette gőzgépszabályozó vagy a Cornelis Drebbel (1572–1633) által felfalált termosztát. Cornelius Drebbel volt az is, aki a tengeralattjárót felfalálta. A stabil visszacsatolt rendszerek elméletét a 19. században fejlesztették ki.

A ma **irányításelméletnek** (**control theory**) nevezett terület keletkezésében Norbert Wiener (1894–1964) játszott központi szerepet. Wiener zseniális matematikus volt, aki többek közt Bertrand Russell-lel is együttműködött, mielőtt az érdeklődését a biológiai és mechanikai szabályozó rendszereknek és ezek a kognitív mechanizmusokkal való kapcsolatának szentelte. Craikhoz hasonlóan (aki szintén használt szabályozási rendszereket mint pszichológiai modelleket). Wiener és kollégái, Arturo Rosenblueth és Julian Bigelow kihívást jelentettek a behaviorista ortodoxia számára (Rosenblueth és társai, 1943). Nézetük szerint a célorientált viselkedés a „hibát” – az aktuális állapot és a célállapot közötti különbséget – minimalizáló szabályozó mechanizmusból fejlődik ki. Az 1940-es évek végén Wiener, Warren McCulloch-val, Walter Pittssel és Neumann Jánossal együtt egy sor konferenciát szervezett, ahol a kognitív folyamatok új matematikai és számítási modelljeivel foglalkoztak, és a behaviorista tudományok területén sok tudósra voltak hatással. Wiener *Cybernetics* c. könyve (Wiener, 1948) bestseller lett, és ráébresztette a széles publikumot az intelligens mesterséges gépek lehetőségére.

A korszerű irányításelmélet, és különösen ennek a *sztochasztikus optimális szabályozás* nevű ága olyan rendszerek kifejlesztését tűzte ki célul, amelyek egy **célfüggvényt** (**objective function**) maximalizálnak az időben. Ez nagyjából megegyezik az MI-ről alkotott képünkkel: optimálisan viselkedő rendszereket fejleszteni. Akkor miért lett az MI és az irányításelmélet két különböző kutatási terület, hiszen megalkotói még szoros kapcsolatban is álltak egymással? A válasz forrása az a szoros kapcsolat, amely a résztvevők által ismert matematikai technikák és az ezeknek megfelelő, mindenkor világképben megjelenő problémák halmaza között alakult ki. Az irányításelmélet eszközei, a mátrixalgebra és az analízis folytonos változók rögzített halmazai által leírható rendszerekhez vezettek. Az egzakt analízis ráadásul tipikusan csak *lineáris* rendszerek esetében lehetséges. Az MI-t részben azért alkották meg, hogy az irányításelmélet 1950-es évekbeli matematikájától meg tudjanak szabadulni. A logikai következtetés és a számítás eszközei lehetővé tettek az MI-kutatók számára, hogy olyan problémákat is figyelembe vegyenek, mint a nyelv, a látás és a tervkészítés, amelyek az irányításelméleti célkitűzéseken teljesen kívül estek.

## Nyelvészeti (1957-től napjainkig)

- Mi a nyelv és a gondolat kapcsolata?

B. F. Skinner 1957-ben publikálta a *Verbal Behavior* c. művét. A könyv e terület egyik legjobb szakértőjének tollából a nyelvtanulás behaviorista megközelítésének átfogó és részletes tárgyalását adta. Furcsamód a könyvismertető ugyanolyan híres lett, mint maga a könyv, és a behaviorizmus iránti érdeklődést csaknem kiirtotta. Az ismertető írója Noam Chomsky volt, aki épp akkor publikálta a saját elméletéről szóló *Syntactic Structures* c. könyvét. Chomsky kimutatta, hogy a behaviorista elmélet nem kezeli a nyelvben meglevő kreativitást. Az elmélet nem tudta megmagyarázni, hogy egy

gyerek miként képes olyan mondatokat megérteni vagy megformálni, amilyeneket korábban soha nem hallott. Chomskynak, Pánini (kb. i. e. 350) indiai nyelvészeg viszszatekintő, szintaktikai modellekre alapozó elmélete viszont magyarázatot tudott adni erre, és a korábbi elméletekkel ellentétben kellően formális volt ahhoz, hogy legalább elvben programozható is legyen.

A modern nyelvészeti és az MI nagyjából ugyanabban az időben „született meg” és együtt fejlődött, a nyelv használatára összpontosító számítógépes nyelvészettel (computational linguistics) vagy természetes nyelvfeldolgozásnak (natural language processing) nevezett hibrid területen találkozva egymással. A nyelv megértésének problémája rövidesen sokkal bonyolultabbnak bizonyult, mint ahogy ez 1957-ben látszott. A nyelv megértéséhez meg kell érteni a témát és a kontextust is, nem elegendő elhelyezni a mondat struktúrájának a megértése. Ez persze triviálisnak tűnhet, azonban az 1960-as évekig mégsem volt általánosan elfogadott. A tudásreprezentációhoz (knowledge representation) – amely annak a kutatása, hogy a tudást hogyan fejezzük ki a számítógép által feldolgozható formában – tartozó kezdeti kutatások zöme a nyelvhez kötődött és a nyelvészeti kutatásokból táplálkozott, azok viszont a nyelv évtizedes filozófiai elemzéseivel voltak kapcsolatban.

## 1.3. A MESTERSÉGES INTELLIGENCIA TÖRTÉNETE

A bevezető anyaggal a hátunk mögött most már készek vagyunk arra, hogy felvázoljuk a szó szoros értelmében vett mesterséges intelligencia fejlődését.

### A mesterséges intelligencia érlelődése (1943–1955)

Az első olyan eredményt, amit ma általánosan MI-eredménynek ismernek el, Warren McCulloch és Walter Pitts érte el (McCulloch és Pitts, 1943). Három forrásból merítettemek: az alapszintű fiziológiai és az agyi neuronok működésére vonatkozó ismeretekből, az ítéletkalkulus Russell és Whitehead-féle formális elemzéséből és Turing számításelméletéből. Egy mesterséges neuron modellt javasoltak, ahol minden neuron vagy „bekapcsolt”, vagy „kikapcsolt” állapotban lehet, és ahol az átkapcsolás „be” állapotba akkor történik, amikor a neuront kellő számú szomszédos neuron stimulálja. A neuron állapotáról azt tartották, hogy „ténylegesen azzal a logikai állítással ekvivalens, amely a megfelelő ingert kiváltotta”. Kimutatták például, hogy összekapcsolt neuronok valamilyen hálózatával minden kiszámítható függvény előállítható, és hogy egyszerű hálóstrukturákkal az összes logikai műveletet (ÉS. VAGY. NEM stb.) is elő lehet állítani. McCulloch és Pitts azzal is felvetette, hogy egy megfelelően kialakított háló képes lehet tanulni is. Donald Hebb egy olyan egyszerű értékfrissítő szabályt mutatott be a neuronok közötti összeköttetések erősségeinek módosítására, amely lehetővé teszi a tanulást (Hebb, 1949). Tanulási szabálya, amit **Hebb-tanulásnak** (Hebbian learning) nevezünk, máig érvényes hatású modellnek bizonyult.

1951-ben a Princeton Egyetem matematika tanszékén két végzős hallgató – Marvin Minsky és Dean Edmonds – megépítette az első neurális számítógépet. A SNARC-nak elnevezett gépben 3000 elektroncső és a B–24 bombázó automatapilóta mechanizmusa egy

40 neuronból álló hálózatot szimulált. Minsky PhD-bizottsága szkeptikus volt, vajon egy ilyen munkát matematikának lehet-e nevezni, de Neumann János (aki tagja volt a bizottságnak) állítólag úgy nyilatkozott, hogy „ha ez nem is matematika most, valamikor az lesz”. Minsky volt később az, aki nagy hatású tételeivel kimutatta a neuronhálós kutatás korlátait.

Sok kezdeti eredményt lehetne MI-nek nevezni, azonban egy teljes elköpzelést az MI-ról 1950-ben Alan Turing fogalmazott meg a *Computing Machinery and Intelligence* c. cikkében. Itt vezette be a Turing-teszt, a gépi tanulás, a genetikus algoritmusok és a megerősítéses tanulás fogalmakat.

## A mesterséges intelligencia megszületése (1956)

Princeton volt az otthona az MI egy másik befolyásos személyiségeinek, John McCarthynak. Az egyetem befejezése után McCarthy a Dartmouth College-ba került, ami a téma kör hivatalos szülőhelye lett. McCarthy meggyőzte Minskyt, Claude Shannont és Nathaniel Rochestert, hogy segítsenek neki azokat az amerikai kutatókat összehozni, akik érdekeltek az automataelméletben, a neurális hálókban és az intelligencia kutatásában. 1956 nyarán egy két hónapos munkatalálkozót szerveztek Dartmouthban. Összesen tíz résztvevő gyűlt össze, beleértve Trenchard More-t Princetonból, Arthur Samuelt az IBM-től, valamint Ray Solomonofft és Oliver Selfridge-et az MIT-ből.

A pálmát a Carnegie Tech.<sup>13</sup> két kutatója, Allen Newell és Herbert Simon vitte el. Bár másoknak is voltak ötletei, és néhány esetben konkrét alkalmazásra – dámajátékra – voltak programjaik is, Newell és Simon már egy következető programmal, a Logic Theorist (LT)-vel rendelkezett.<sup>14</sup> Erről Simon azt állította, hogy „egy olyan programot találtunk fel, amely képes nemnumerikusan gondolkodni, és ezzel meg is oldottuk a tiszteletre méltó anyag-szellem viszony problémáját”. Nem sokkal a munkatalálkozó után a program képes volt bebizonyítani a Russell és Whitehead *Principia Mathematica* c. művének 2. fejezetében foglalt tételek többségét. Azt mondják, Russell el volt ragadatva, amikor Simon megmutatta neki, hogy az egyik tétel esetén a program rövidebb bizonyítással állt elő, mint amit a *Principiában* közöltek. A *Journal of Symbolic Logic* szerkesztői kevésbé hatotta meg a dolog. A Newell, Simon és a Logic Theorist szerzői hármastól származó cikket elutasították.

A dartmouthi munkatalálkozó új áttöréshez ugyan nem vezetett, de a fontos személyiségek bemutatkoztak egymásnak. A következő húsz évben ők, továbbá hallgatóik és kollégái az MIT-n, a CMU-n, a Stanfordon és az IBM-nél lesznek azok, akik meghatározó szerepet töltenek be az MI területén. A munkatalálkozó talán legtartósabb eredménye az volt, hogy elfogadták a terület McCarthy által kreált új nevét, azaz a **mesterséges intelligenciát (artificial intelligence)**. A „számítási racionalitás” talán jobb név lett volna, de az „MI” név azóta is megmaradt.

Ha bepillantunk a dartmouthi munkatalálkozót javasló anyagba (McCarthy és társai, 1955), látjuk, miért volt szükségszerű, hogy az MI egy külön területté váljon. De miért

<sup>13</sup> Most Carnegie Mellon Egyetem (Carnegie Mellon University, CMU).

<sup>14</sup> Az LT megrásához Newell és Simon az IPL-t, a listakezelő nyelvet is feltalálta. Nem rendelkezvén fordítóval, programjukat kézzel fordították le gépi kódra. Hogy a hibákat elkerüljék, párhuzamosan dolgoztak, és minden utasítás megírásánál a bináris számokat egymásnak átkiabálták, hogy így biztosra menjenek.

nem lehetett az MI-kutatást az irányításelmélet, az operációkutatás vagy a döntéselmélet keretein belül tartani, amikor ráadásul ezek célkitűzései nagyon hasonlók az MI célkitűzéseihez? Vagy az MI miért nem lett a matematika egyik ága? Az első válasz az, hogy az MI a kezdetek óta sajátjának tekintette az olyan emberi képességek duplikálását, mint a kreativitás, az önéflesztés és a nyelv használata. Ezekkel a kérdésekkel semmilyen más terület nem foglalkozott. A másik válasz a módszertanban rejlik. Az említett területek közül tisztán csak az MI tekinthető a számítógépes tudományok egy ágának (bár az operációkutatás szintén súlyt helyez a számítógépes szimulációkra). Az MI az egyetlen olyan terület, ahol bonyolult, változó környezetben autonóm módon működő gépek építése a cél.

## Korai lelkesedés, nagy elvárások (1952–1969)

Az MI korai évei – bizonyos kereteken belül – bővelkedtek a sikerekben. Ha figyelembe vesszük azoknak az időknek a primitív számítógépeit és programozási eszközeit, továbbá azt, hogy még néhány évvel korábban is csupán aritmetikai feladatok elvégzésére tartották alkalmasnak a számítógépet, megdöbbentő volt, hogy a számítógép akár csak távolról is okosnak tűnő dologra lehet képes. Értelmiégi körökben, összességében, inkább azt szerették volna hinni, hogy „a gép  $X$ -re soha nem lesz képes” (az  $X$ -ek, Turing által kigyűjtött hosszú listája a 26. fejezetben található). Az MI kutatói természetesen erre azzal válaszoltak, hogy egymás után demonstrálták az  $X$ -eket. A modern MI-kutatók közül néhányan úgy említik ezt az időszakot, mint a „Nézze uram, biz’ isten, magától megy!” idejét.

Newell és Simon kezdeti sikereit az általános problémamegoldó program, a General Problem Solver, GPS követte. A Logic Theoristtal ellentétben ezt a programot eleve úgy terveztek, hogy az emberi problémamegoldás protokolljait imitálja. Az derült ki, hogy a program által kezelhető feladványok osztályán belül, a részcélok és a lehetséges cselekvések megfontolásának sorrendje tényleg hasonlított ahhoz, mint ahogy a hasonló problémákon dolgozó emberek cselekszenek. Így, a GPS volt talán az első, az „emberi módon gondolkodni” megközelítést megtestesítő program. A GPS és az azt követő programok sikere arra készítette Newellt és Simont (Newell és Simon, 1976), hogy megfogalmazzák híres **fizikai szimbólumrendszer hipotéziséit** (**physical symbol system**), amely azt állítja, hogy „a fizikai szimbólumrendszerök az általános intelligens cselekvés szükséges és elégsges eszközeivel rendelkeznek”. Arra gondoltak, hogy minden, intelligenciát felmutató rendszernek (legyen az ember vagy gép) képesnek kell lennie arra, hogy szimbólumokból álló adatstruktúrákat manipuláljon. Később látni fogjuk, hogy e hipotézist több irányból is megtámadták.

Az első néhány MI-programot az IBM-nél Nathaniel Rochester és kollégái fejlesztették ki. Herbert Gelernter egy olyan geometriai tételelbizonyító programot (Geometry Theorem Prover, Gelernter, 1959) írt, mely sok matematikus hallgató által trükkösnek talált tételeit tudott bebizonyítani. 1952-től kezdve Arthur Samuel játszájátékot játszó programokat írt, amelyek végül megtanultak egy erős amatőr versenyzői szinten játszani. Eközben sikerült megcáfolnia, hogy a számítógép csak arra képes, amire utasítják, hiszen programja gyorsan megtanult nála is jobban játszani. A program tv-bemutatása 1956 februárjában igen nagy hatást keltett. Turinghoz hasonlóan Samuelnek is csak nehezen

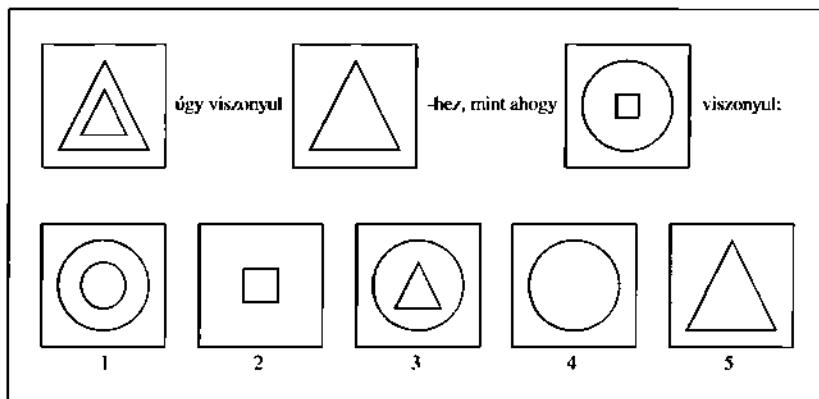
sikerült gépidőt szereznie. Éjszaka dolgozott, az IBM számítógépes üzemében végtesztelésre váró gépeket használva. A kétszemélyes játékокkal a 6. fejezet, a Samuel által használt tanuló technikával és annak továbbfejlesztésével a 21. fejezet foglalkozik.

John McCarthy Dartmouth-ból átment az MIT-re, és ott egyetlen év alatt, a történelminek nevezhető 1958-as évben, három kulcsfontosságú eredményt ért el. Az 1. számú MIT AI Lab Memóban definiálta a Lispet, amely elsődleges MI-programozási nyelvvé nötte ki magát. A Lisp a második legrégebbi nyelv, amely még használatban van, a Fortrannál csak egy évvel fiatalabb. A Lisp esetén McCarthy rendelkezett már a szükséges eszközzel, de a ritka és drága számítógépes erőforrásokhoz való hozzáférés számára is komoly problémát jelentett. Így aztán az MIT-n McCarthy és mások kitalálták az időosztást. Szintén 1958-ban McCarthy *Programs with Common Sense* címen cikket publikált, amelyben az Advice Takert írta le. Ez egy hipotetikus program, amit az első teljes MI-rendszernek tekinthetünk. A Logic Theoristhez és a Geometry Theorem Proverhez hasonlóan McCarthy programja is tudást használt fel egy probléma megoldásának megtalálásához. Azonban másokkal ellentétben, ennek a programnak a világra vonatkozó általános tudással kellett rendelkeznie. McCarthy megmutatta például, hogy néhány egyszerű axióma elegendő ahhoz, hogy programja képes legyen tervezet generálni arra vonatkozóan, hogyan kell a repülőtérré kiemenni ahhoz, hogy a repülőgépet le ne küssük. A programot úgy tervezte, hogy képes legyen normális működés közben új axiómákat is elfogadni, és ennek eredményeként átprogramozás nélkül új területeken is kompetenciát mutatni. Az Advice Taker ily módon a tudásreprezentáció és a következetés leglényegesebb elveit testesítette meg, miszerint hasznos, ha rendelkezünk a világot és az ágens cselekvéseinek eredményét leíró explicit és formális reprezentációval, és képesek vagyunk ezt a reprezentációt deduktív módon manipulálni. Figyelemre méltó, hogy 35 év múltával még mennyire releváns maradt az 1958-as cikk.

1958 volt az az év is, amikor Marvin Minsky az MIT-re ment át. Kezdeti együttműködése McCarthyval nem tartott sokáig. McCarthy a reprezentációra és a formális logikai következetésre tette a hangsúlyt, Minskyt inkább az érdekelte, hogy a programok működőképesek legyenek, majd végül logikaellenes álláspontra helyezkedett. 1963-ban McCarthy a Stanfordon megalakította az ottani MI-labot. Kutatási programja – amely arra irányult, hogy a logikát felhasználja a legvégső Advice Taker építésében – lökést kapott, amikor J. A. Robinson felfedezte a rezolúciót, az elsőrendű logika teljes bizonyítási eljárását (lásd 9. fejezet). A Stanfordon a kutatás hangsúlyozottan a logikai következetés általános módszereire irányult. A logika alkalmazásaihoz tartoztak Cordell-nek a Green kérdését megválaszoló és tervkészítő rendszerei (Green, 1969b), továbbá Shakey robotikus projektje az új Stanfordi Kutatóintézetben (Stanford Research Institute, SRI).<sup>15</sup> Ez a projekt – amit részletesebben a 25. fejezetben tárgyalunk – volt az első, amely a logikai következetést és a fizikai aktivitást teljes egészében integrálta.

Minsky a hallgatók egész sorát irányította, akik a megoldásokhoz láthatóan intelligenciát igénylő, korlátos problémákkal foglalkoztak. Ezeket a korlátos problématerületeket később mikrovilágoknak (*microworlds*) nevezték el. James Slagle SAINT nevű programja (Slagle, 1963a) képes volt az első éves analízis tanfolyamra jellemző, zárt

<sup>15</sup> Az angol nyelvben a tervkészítő tervezés (planning) és a termékter készítő tervezés (design) szépen elkülönül. Hogy a pontos jelentést hangsúlyozzuk, az angol „planning” fordításánál a továbbiakban „tervkészítés” használunk. (A ford.)

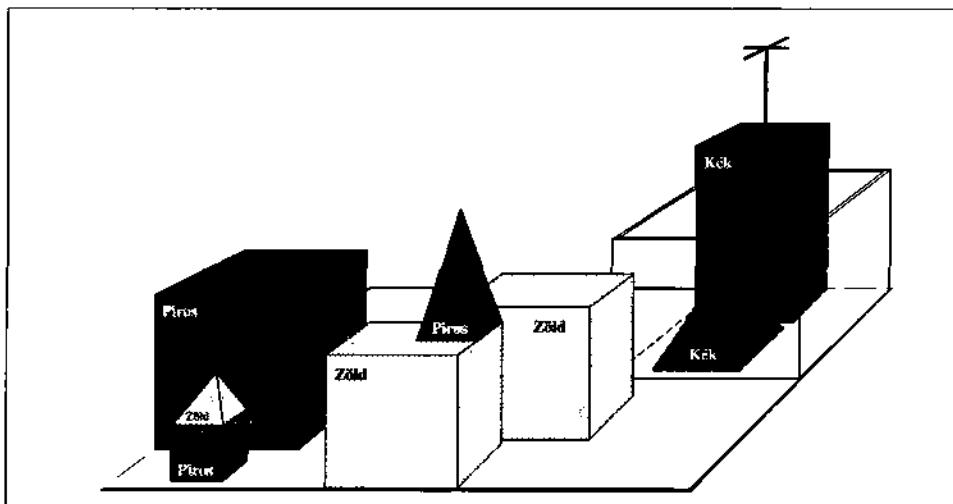


1.4. ábra. Egy konkrét probléma, amelyet Evans ANALOGY programja megoldott

alakra hozható integrálszámítási feladatokat megoldani. Tom Evans ANALOGY programja (Evans, 1968) az IQ-tesztekben előforduló (lásd 1.4. ábra) geometriai analógia jellegű problémákat oldotta meg. Daniel Bobrow STUDENT programja (Bobrow, 1967) olyan algebrai feladványokat oldott meg, mint például az alábbi:

Ha Tamás ügyfeleinek a száma kétszerese az általa közzétett hirdetések 20%-a négyzetének, és Tamás 45 hirdetést adott közzé, akkor hány ügyfele van Tamásnak?

A mikrovilágok legismertebbje a kockavilág lett, amely egy asztalra (vagy gyakrabban egy szimulált asztalra) helyezett tömör geometriai testekből áll (lásd 1.5. ábra). Az ilyen világban értelmezett feladat a kockák egy bizonyos átrendezése egy olyan robotkar segítségével, amely egyszerre egyetlenegy kockát képes megfogni. A kocka-



1.5. ábra. Egy elrendezés a kockavilágban. A SHRDLU robot (Winograd, 1972) éppen sikeresen teljesítette a „keresd meg azt a blokkot, ami a kezedben tartottnál magasabb, és tudd a dobozba” utasítást.

világ otthont adott David Huffman gépi látási projektjének (Huffman, 1971), David Waltz gépi látási és kényszerterjesztés-kutatásának (Waltz, 1975), Patrick Winston tanulási elméletének (Winston, 1970), Terry Winograd természetes nyelvfeldolgozási programjának (Winograd, 1972) és Scott Fahlman tervkészítő programjának (Fahlman, 1974).

A McCulloch és Pitts neurális hálóin alapuló kezdeti kutatás szintén virágzott. Winograd és Cowan eredményei megmutatták, hogy a nagyszámú elem hogyan képes együttesen egy egyedi fogalmat reprezentálni, miközben növeli a párhuzamosságot és a robosztusságot (Winograd és Cowan, 1963). Hebb tanulási módszereit Bernie Widrow (Widrow és Hoff, 1960; Widrow, 1962) fejlesztette tovább, aki a **perceptronokat** vezette be (Rosenblatt, 1962). Rosenblatt bebizonyította **perceptron konvergencia tételeit** (**perceptron convergence theorem**), kimutatva ezzel, hogy tanulási algoritmusával a perceptron súlyait úgy módosítani, hogy az tetszőleges bemeneti adatokhoz illeszkedjen, feltéve, hogy ilyen illeszkedés egyáltalán lehetséges. Ezekről a kérdésekről a 20. fejezetben írunk majd.

## Egy adag realitás (1966–1973)

Az MI kutatói már a kezdetekben sem voltak szégyenlősek a várható sikereiket illetően. Gyakran idézik Herbert Simonnak a következő, 1957-ből származó kijelentését:

Sem meglepni, sem sokkolni senkit nem célok – de a legegyszerűbben összefoglalva azt mondhatom, hogy a világban léteznek ma már gondolkodó, tanuló és kreatív gépek. E képességek rohamosan fog fejlődni, és – a közeljövőben – az általuk feldolgozott problémák köre összemérhető lesz azokkal a problémákkal, amelyekkel az emberi elme eddig megküzdött.

Bár vitatható, hogy „közeljövőnek” mi tekinthető, Simon néhány más előrejelzése konkrétabb volt. Megjósolta, hogy tíz éven belül a számítógép sakkvilágbanak lesz, és hogy a gép fontos új matematikai tételeket fog bebizonyítani. Ezek a jóslatok, ha nem is tíz, hanem inkább negyven év múltával (közelíti leg vagy teljesen), de beigazolódtak. Simon magabiztossága a korai MI-programok egyszerű példaproblémákon felmutatott sikereiből táplálkozott. Ezek a korai rendszerek azonban majdnem minden esetben számalmasan csödöt mondtak, ha szélesebb körben vagy netán nehezebb problémákra akarták őket bevetni.

A nehézség egyik forrása az volt, hogy a korai programok az általuk kezelt problémáról sokszor kevés vagy szinte semmi tudást nem tartalmaztak, és csupán egyszerű szintaktikai manipulálással értek el sikereket. Egy tipikusnak mondható történet a korai gépi fordítással kapcsolatos. A gépi fordítást a Nemzeti Kutatási Alap (National Research Council) bőkezűen finanszírozta azért, hogy a Szputnyik 1957-es kilövését követően meggyorsítsák az orosz tudományos cikkek fordítását. Kezdetben azt vélték, hogy az angol és az orosz nyelvtanra alapozó egyszerű szintaktikai transzformációk és az elektronikus szótárra alapozó szóbehaylók elegendő lesz a mondat pontos értelmének átadásához. Valójában a fordításhoz a téma általános ismerete szükséges, hogy feloldhassuk a kétértelműségeket, és a mondat jelentését megállapítsuk. „A szellem készséges, de a test gyenge” („the spirit is willing but the flesh is weak”) híres vissza-

fordítása „a vodka jó, de a hús romlott”-ra („the vodka is good but the meat is rotten”) a tapasztalt nehézségeket világosan érzékelte. Egy tanácsadó bizottság 1966-os jelentése azt állapította meg, hogy az „általános tudományos szöveg fordítása még nem megoldott, és a közeljövőben e téren gyors előrehaladás nem is várható”. Az egyetemi gépi fordítási projekteknek a kormány általi finanszírozását Amerikában teljesen megszüntették. Manapság a gépi fordítás egy nem tökéletes, ám széles körben alkalmazott eszköz műszaki, kereskedelmi, kormányzati és internetdokumentumok esetében.

A másik nehézséget az jelentette, hogy sok olyan probléma, amelyeket az MI által kísérletek megoldani, kezelhetetlen volt. A korai MI-programok többsége úgy dolgozott, hogy a problémára vonatkozó alapvető tényeket gépen reprezentálva, megoldó lépés-szekvenciákat próbáltak ki, a különféle lépéskombinációkkal addig kísérletezve, amíg nem leltek rá a helyesre. A korai programok azért voltak használhatók, mert a mikrovilágok csak kevés objektumot, és ebből addódóan nagyon kevés lehetséges cselekvést és nagyon rövid megoldási sorozatokat tartalmaztak. Az NP-teljesség elmeletének megfogalmazása előtt általában azt tartották, hogy a nagyobb problémákra „felskálázni” csupán gyorsabb hardver és nagyobb memória kérdése. A rezolúciós tételbizonyítás kifejlesztését kísérő optimizmus például hamarosan lelohadt, amikor a néhány tucat ténynél többet igénylő tételeket nem sikerült bebizonyítani. Az a tény, hogy egy program egy megoldás megtalálására elvben alkalmas, nem jelenti azt, hogy a program bármi olyan mechanizmust is tartalmaz, amely a megoldás gyakorlati megvalósításához szükséges.

A korlátozott számítási kapacitás illúziója nem csak a problémamegoldó programokra korlátozódott. A gépi evolúció (machine evolution), amelyet most genetikus algoritmusoknak (genetic algorithms) nevezünk (Friedberg, 1958; Friedberg és társai, 1959), területén végzett korai kísérletek azon a kétségtelenül helyes feltevésen alapultak, amely szerint ha egy gépi kódú programot megfelelően kicsi mutációk révén változtattunk, tetszőleges, egyszerű feladatot jól megoldó programhoz juthatunk el. Az ötlet tehát az volt, hogy véletlen mutációkkal próbálkozva, a program viselkedését javító mutációkat tartuk meg. Azonban ezernyi óra gépidő ellenére sem sikerült szinte semmilyen előrehaladást kimutatni. A korszerű genetikus algoritmusok jobb reprezentációkat használnak, és több sikerre is vitték.

Az volt az MI ellen irányuló legfontosabb kritika a Lighthill-tanulmányban (Lighthill, 1973), hogy nem képes lekiüzdeni a „kombinatorikus robbanást”. A tanulmány alapján a brit kormány, kettő kivételével az összes egyetemen minden MI-kutatási támogatást visszavont (a szóbeszéd kissé más és színesebb képet fest a nyomdafestéket nem tűrő politikai ambíciókról és a személyes ellenségeskedésről).

A harmadik nehézség forrását az intelligens viselkedés generálásához használt alapvető struktúrák fundamentális korlátai jelentették. Minsky és Papert *Perceptrons* c. könyve (Minsky és Papert, 1969) például azt bizonyította be, hogy bár a perceptron (a neurális háló egy egyszerű formája) megtanulhat minden, amit képes reprezentálni, vajmi keveset képes reprezentálni. Így például a két bemenetű perceptron nem lehet megtanítani arra, hogy a bemeneteinek különbözőségét felismerje. És bár a szerzők eredményei bonyolultabb, többrétegű hálókra nem vonatkoztak, a neurális hálók kutatásának finanszírozása rövidesen majdnem nullára esett vissza. A sors íróiája, hogy a többrétegű neurális hálók későbbi, az 1980-as években történt óriási feltámadását hozó új visszaterjesztéses tanuló algoritmust éppen 1969-ben fedezték fel először (Bryson és Ho, 1969).

## Tudásalapú rendszerek: a hatalom kulcsa? (1969–1979)

A problémamegoldásnak az a képe, amely az MI-kutatás első évtizedében alakult ki, egy olyan általános célú kereső mechanizmus volt, amely a teljes megoldás megtalálásának érdekében szekvenciába fűzte az elemi következetési lépéseket. Az ilyen megközelítéseket **gyenge módszereknek** (weak methods) nevezték, mert annak ellenére, hogy általánosak, a problémák nagy vagy nehéz példányaira nem skálázhatók fel. A gyenge módszerek alternatívája az erőteljesebb, területspecifikus tudás használata, amely lehetővé teszi a nagyobb granuláltságú következetési lépések megvalósítását, és szükebb szakérői tárgyterületeken a tipikus konkrét problémák megoldását. Ahhoz, hogy egy nehéz problémát megoldjunk, mondhatni majdnem kész válasszal kellene rendelkeznünk.

E megközelítés egyik korai példája a DENDRAL program volt (Buchanan és társai, 1969). A programot a Stanfordon fejlesztették ki, ahol Ed Feigenbaum (Herbert Simon volt hallgatója), Bruce Buchanan (a számítógépes szakemberből lett filozófus) és Joshua Lederberg (Nobel-díjas genetikus) összefogtak, hogy a tömegspektrométer által szolgáltatott adatokból a molekuláris struktúra kinyerésének problémáját megoldják. A program bemeneti adatai a molekula alapképlete (például  $C_6H_{13}NO_2$ ) és a tömegspektrum voltak. A spektrum megadta a molekula bizonyos részeinek a tömegét, amikor a molekulát elektronsugárral bombázták. A tömegspektrum tartalmazhatott például  $m = 15$ -nél egy csúcsot, amit a metil ( $CH_3$ ) molekularésszel lehetett azonosítani.

A program naiv verziója a molekula képletével konzisztens minden lehetséges struktúrát előállított. Ezt követően minden egyes struktúrához megjósolta a megfelelő megfigyelhető tömegspektrumot, és ezt hasonlította össze az aktuálisan megfigyelt spektrummal. Ahogy várható volt, nagyobb molekulák esetén az eljárás gyorsan kezelhetetlenné vált. A DENDRAL kutatói analitikus vegyészekhez fordultak segítségért. Azt találták, hogy a vegyészek a spektrumban a molekulában található elterjedt részstruktúráakra utaló, jól ismert csúcsmintákat keresik. Így például a keton ( $C = O$ ) alcsoport (amely 28 súlyú) felismeréséhez az alábbi szabály volt használatos:

- ha két olyan csúcs,  $x_1$  és  $x_2$  létezik, hogy**  
 (a)  $x_1 + x_2 = M + 28$  ( $M$  a teljes molekula tömege);  
 (b)  $x_1 - 28$  egy magas csúcs;  
 (c)  $x_2 - 28$  egy magas csúcs;  
 (d)  $x_1$  és  $x_2$  közül legalább egy csúcs magas.  
**akkor ketoncsoport van jelen.**

Azzal, hogy felismerjük, hogy egy konkrét részstruktúra a molekula része, a lehetséges struktúrajelöltek száma nagyon nagy mértékben csökken. A DENDRAL-rendszer hatékony volt, mert:

Az ilyen problémák megoldásához szükséges összes elméleti tudást sikerült (a rendszer spektrumjósító komponenseiben) leképezni az általános formáról („elsődleges ismeretek”) egy hatékony speciális formára („szakácskönyv”) (Feigenbaum és társai, 1971).

A DENDRAL fontossága abban rejlik, hogy vitathatatlanul ez volt az első sikeres *tudásintenzív* rendszer. Szakértelmét a nagyszámú speciális rendeltetésű szabály biztosította.

A későbbi rendszerekben szintén megjelent a McCarthy-féle Advice Taker egyik fő gondolata – a (szabályformájú) tudás és a következtető komponens határozott elkülönítése.

Okulva az ilyen leckén, Feigenbaum és mások a Stanfordon belekezdték a heurisztikus programozási projektbe (Heuristic Programming Project, HPP) azzal a céllal, hogy megvizsgálják, a szakértőrendszernek (*expert systems*) ez az új módszertana milyen mértékben alkalmazható az emberi szakértelem más területein. A következő komoly erőfeszítés az orvosi diagnózis területén született meg. Feigenbaum, Buchanan és dr. Edward Shortliffe várrel kapcsolatos fertőzések diagnosztizálására fejlesztették ki a MYCIN-rendszert. 450 szabályával a MYCIN elérte az egyes szakértők hatékonyságát és a kezdő orvosoknál lényegesen jobb teljesítményt nyújtott. A MYCIN két fő vonatkozásban különbözött a DENDRAL-tól. Először is, a DENDRAL szabályaival ellentétben, a MYCIN-szabályok származtatásához nem létezett semmilyen általános elméleti modell. A szabályokat a szakértők kiterjedt kikérdezése révén kellett beszerezni, akik viszont a szabályokat könyvekből, más szakértőktől és közvetlen tapasztalatokból merítették. A másik különbség abból eredt, hogy a szabályoknak tükrözniük kellett az orvosi ismeret bizonytalanságát. A MYCIN-ben bizonyossági tényezőknek (*certainty factors*) (lásd 14. fejezet) nevezett bizonytalanságkezelő mechanizmust alkalmaztak, amelyről akortájt úgy tűnt, jól tükrözi a tényállás diagnózisra gyakorolt hatásának orvosi megítélesét.

A tárgytartomány ismeretének fontossága nyilvánvaló volt a természetes nyelvfelismerés területen is. Bár Winograd természetes nyelvfelismerő rendszere, a SHRDLU igen élenk érdeklődést keltett, a szintaktikai elemzéstől való függősége néhány, a kezdeti gépi fordításban már tapasztalt problémához vezetett. A program képes volt a kétértelműségen felülkerekedni, és a névmási szerkezeteket megérteni. Ez azonban azért volt lehetséges, mert a programot kifejezetten egy adott tárgytartományhoz – a kockavilág-hoz – fejlesztették ki. Néhány kutató, köztük Eugene Charniak, Winograd végzős társa az MIT-ról, felvette, hogy a természetes nyelv robusztus felismerése a világról szóló általános ismereteket és ezen ismeretek általános felhasználási módszereit igényli.

A Yale-en, Roger Schank, a nyelvesszé lett MI-kutató, ezt a nézetet még jobban hangsúlyozta, azt állítván, hogy „olyan doleg, mint a szintaxis pedig nincs”, ami ugyan sok nyelvész felháborított, de egyben egy hasznos eszmecsérét is elindított. Schank és hallgatói a természetes nyelvet felismerő programok egész sorát építették meg (Schank és Abelson, 1977; Wilensky, 1978; Schank és Riesbeck, 1981; Dyer, 1983). A hangsúly azonban kevésbé magára a nyelvre, sokkal inkább a nyelv megértéséhez szükséges tudás reprezentálására és a vele való következetésre helyezték. E problémakörbe tartozott a sztereotip helyzetek reprezentálása (Cullingford, 1981), az emberi memória szervezésének leírása (Rieger, 1976; Kolodner, 1983) és a célok, tervezek megértése (Wilensky, 1983).

A valós alkalmazások széles körű elterjedése a működőképes tudásreprezentációs sémkák iránti igények növekedéséhez vezetett. Számos különböző reprezentációs és következtető nyelvet fejlesztettek ki. Egyes megoldások a logikán alapultak – például a Prolog nyelv, amely Európában, és a PLANNER nyelvcsalád, amely az Egyesült Államokban lett népszerű. Mások, a Minsky által bevezetett keretek (*frames*) (Minsky, 1975) ötletét követve, inkább strukturált megközelést tettek választottak. Egybegyűjtöttek bizonyos eseménytípusokra vagy objektumokra jellemző tényeket, majd azokat a biológiai taxonómiaira hasonlító nagy taxonomikus típushierarchiákba rendezték.

## Az MI iparrá válik (1980-tól napjainkig)

Az első üzletileg sikeres szakértőrendszeret, az R1-et a Digital Equipment Corporation-nél (McDermott, 1982) alkalmazták. A rendszer az új számítógépes rendszerek megrendeléseit segítette konfigurálni, és 1986-ra évi mintegy 40 millió dollár megtakarítást jelentett a cégnek. 1988-ra a DEC MI-csoportja már 40 szakértőrendszeret állított üzembe, és több ilyen rendszer üzembe állítása folyamatban volt. A DuPont cégnél 100 ilyen rendszer üzemelt, és folyamatban volt további 500 rendszer fejlesztése. Az ezekből származó becsült megtakarítás elérte az évi 10 millió dollárt. Majdnem minden nagyobb amerikai cég saját MI-csoporttal rendelkezett, és vagy használta, vagy tanulmányozta a szakértőrendszer technológiát.

1981-ben a japánok meghirdették az „ötödik generációs” (Fifth Generation) projektjüket – egy 10 éves tervet a Prolog nyelvet gépi kódként használó, intelligens számítógépes rendszerek építésére. Válaszul az Egyesült Államokban létrehozták az MCC (Microelectronics and Computer Technology Corporation) kutatótársulatot, amelynek célja a nemzeti versenyképesség biztosítása volt. Mindkét esetben az MI egy olyan általánosabb erőfeszítés része lett, amely a chiptervezésre és az ember–gép interfész kutatására is irányult. Az MCC és az Ötödik Generáció MI-komponensei azonban az ambiciózus célkitűzéseket mégsem tudták elérni. Nagy-Britanniában az Alvey-jelenős visszaállította a Lighthill-jelenős következtében leállított finanszírozást.<sup>16</sup>

Minden egybevéve az MI-iparnak az 1980-as néhány millió dolláros forgalma 1988-ra 2 milliárd dollárra nőtt. Rövidesen ezután az „MI tele” periódus következett be, amikor sok cég belebukott abba, hogy extravagáns ígéreteit nem tudta teljesíteni.

## A neurális hálók visszatérése (1986-tól napjainkig)

Bár a számítógép-tudomány az 1970-es évek végén a neurális hálók téma köről megfeledkezett, a kutatás más területeken folytatódott. A fizikusok, mint például Hopfield, a statisztikus mechanika módszereit használták, hogy a hálók tárolási és optimalizálási tulajdonságait elemezzék (Hopfield, 1982), úgy kezelve az egyszerű neuronok együttesét, mint atomok együttesét. A pszichológusok, David Rumelhartot és Geoff Hintont is beleértve, folytatták a memória neurális hálós modelljének kutatását. Mint ezt a 20. fejezetben megmutatjuk, az igazi lökés az 1980-as évek derekán történt, amikor legalább négy különböző kutatócsoport újra feltalálta a visszaterjesztéses tanuló algoritmust, azt az algoritmust, amit először Bryson és Ho írtak le 1969-ben. Az algoritmust számos tanulóproblémára alkalmazták mind a számítógépes tudományokban, mind a pszichológiában. Az eredmények széles körű bemutatására a nagy érdeklődést keltő *Parallel Distributed Processing* c. (Rumelhart és McClelland, 1986) gyűjteményes kötetben került sor.

Az intelligens rendszereknek ilyen, ún. **konnekcionista (connectionist)** modelljeit egyesek a Newell és Simon javasolta szimbolikus modellek, valamint a McCarthy és mások által alkalmazott logicista megközelítés közvetlen versenytársának vélték (Smo-

<sup>16</sup> A kínos helyzet elkerülésére intelligens tudásalapú rendszerek (Intelligent Knowledge-Based Systems, IKBS) néven új kutatási területet definiáltak, hiszen a mesterséges intelligenciát hivatalosan törölték.

lensky, 1988). Hogy egy bizonyos szinten az ember szimbólumokkal operál, nyilvánvalónak tűnhet. Sőt Terrence Deakon *The Symbolic Species* c. művében (Deakon, 1997) ezt a képességet az embereket *definiáló jellemzőnek* javasolja. A legmegrögzöttebb konnektionisták azonban kérdőre vonták a tekintetben, hogy a kognitív folyamatok részletes modelljében a szimbolikus manipulációnak van-e egyáltalán valamilyen valós magyarázó szerepe. Ez a kérdés megválaszolatlan maradt, és a jelenlegi álláspont az, hogy a konnektionista és a szimbolikus megközelítések egymás kiegészítői, nem pedig versenytársak.

## Az MI tudománnyá válik (1987-től napjainkig)

A legutóbbi években az MI-kutatásnak mind tartalmában, mind módszertanában lényeges változások álltak be.<sup>17</sup> Mostanság inkább megszokott létező elméletekre építeni, mint teljesen újakat javasolni, az állításokat az intuíció helyett inkább szigorúan vett tételekre, illetve komoly kísérleti bizonyítékokra alapozni, továbbá a lényeges eredményeket nem játékproblémákon, hanem valós feladatokon bemutatni.

Az MI-t részben a létező kutatási területek – mint az irányításelmélet és a statisztika – korlátaival szembeni kitörési vágyból alapították meg. Most azonban az MI ezeket a területeket igyekszik magában foglalni. David McAllester szavaival (McAllester, 1998):

Az MI korai szakaszában plauzsibilisnak tűnt, hogy a szimbolikus számítások új formái, például a keretek és szemantikus hálók, a klasszikus elméletek nagyobb részét elavulttá tették. Ez egyfajta elszigetelődéshez vezetett, ahol az MI a számítási tudomány többségétől el lett választva. Ezzel az izolacionizmussal most szakítunk. Fel kell ismerni, hogy a gépi tanulási nem szabad elszigetelni az információtelméletetől, hogy a bizonytalanság mellettől következtetést nem szabad elszigetelni a sztochasztikus modellezéstől, hogy a keresést nem szabad elszigetelni a klasszikus optimalizálástól és szabályozástól, és hogy az automatikus következtetést nem szabad elszigetelni a formalis módszerektől és a statikus elemzéstől.

Módszertanát tekintve az MI-ben végre a tudományos megközelítés uralkodott el. Hogy egy hipotézist elfogadhattunk, szigorú empirikus kísérleteknek kell alávetni, és az eredmények relevanciáját statisztikailag kell verifikálni (Cohen, 1995). Manapság a kísérletek reprodukálhatóságát az internet és a megosztott tesztadat- és programkódtárak szavatolják.

Ez a folyamat a beszédfelismerés területén jól látható. Az 1970-es években igen sok különböző architektúrát és megközelítést próbáltak ki. Ezek közül sok *ad hoc* jellegű és gyenge volt, amelyek működését csupán néhány, erre a célra megválasztott példán demonstrálták. A legutóbbi években a **rejtett Markov-modelleken** (*hidden Markov models*, HMM) alapuló megközelítések uralják e területet. A HMM-ek két aspektusa lényeges. Először is szigorú matematikai elméleten alapulnak. Ez lehetővé tette, hogy

<sup>17</sup> Vannak, akik ezeket a változásokat úgy jellemzétek, mint az **elegánsak (neats)** győzelmet a **szakadtak (scruffies)** felett. Az elegánsak úgy gondolják, hogy az MI-elméleteknek szigorú matematikai alapokon kell állniuk, míg a szakadtak inkább sok ötletről próbálnának ki, programokat írnának, és megnéznék, mi működőképes ezekből. Mindkét megközelítés fontos. Az elegancia felé történő eltolódás annak a jele, hogy a terület egy stabil és érett szintet ért el (más kérdés persze, hogy ezt a stabilitást új „szakadt” gondolatok mennyire bolygatják fel).

a beszédkutatók a más területeken kifejlesztett több évtizedes matematikai eredményekre építsenek. Másodsor, e modellekkel valós és nagyméretű beszédgyűjteményt felhasználó tanulási folyamat során hozzák létre. Ez biztosítja robusztus működésüket. A szigorú vakesztek a rejtegtől Markov-modellek folyamatos javulását mutatják. A beszédechnológia és a vele rokon kézírás-felismerés útban van a széles körű ipari és fogyasztói alkalmazások felé.

Ez a trend a neurális hálókra is igaz. Az 1980-as években a kutatás többsége arra irányult, hogy kitapasztaják, a hálókkal meddig lehetne el, és hogy megtanulják, a hálók a „hagyományos” technikáktól miben különböznek. A jobb módszertan és az elméleti háttér révén eljutottak ahhoz, hogy most a hálókat össze lehet hasonlítani a megfelelő statisztikai, alakfelismerési és gépi tanulási technikákkal, és az adott alkalmazáshoz meg lehet választani a leginkább sikeres kecsegtetőt. Az ilyen fejlődés eredményeképpen az **adatbányászat** (**data mining**) technológia virágzó új iparrá nőtte ki magát.

Judea Pearl *Probabilistic Reasoning in Intelligent Systems* c. műve a valószínűség- és a döntéselmélet MI-n belüli újból elfogadását jelezte (Pearl, 1988). Mindez azt követően történt, hogy Peter Cheeseman *In Defense of Probability* cikkében összefoglalta az érdeklődés újraéledését (Cheeseman, 1985). A **Bayes-hálók** (**Bayesian networks**) formalizmusát a bizonytalan tények hatékony ábrázolására és a velük történő szabatos következtetés céljára találták ki. Ez a megközelítés a valószínűségi következtető rendszerek 1960-as és 1970-es években tapasztalt problémáit nagyrészt megoldotta, és ma uralja a bizonytalan következtetésre és a szakértőrendszerkre irányuló MI-kutatásokat. Ez a megközelítés teszi lehetővé a tapasztalatból való tanulást és ez kapcsolja össze a klasszikus MI és a neurális hálók legfontosabb eredményeit. Judea Pearl, továbbá Eric Horvitz és David Heckerman munkája támogatta a normatív szakértőrendszer gondolatát, azaz egy olyan rendszerét, amely a döntéselméleti törvényeknek megfelelően racionálisan cselekszik, és nem kíséri meg az emberi szakértőket imitálni (Pearl, 1982a; Horvitz és Heckerman, 1986; Horvitz és társai, 1986). A Windows™ operációs rendszer tartalmaz néhány normatív szakértőrendszert a felmerülő hibák javítására. Erről az elgondolásról szól a 13–16. fejezet.

Hasonló szelíd forradalom következett be a robotika, a gépi látás és a tudásreprezentációk területén. A problémák és bonyolultságuk jobb megértése, a növekvő matematikai háttérrel összefonódva, robusztusabb módszerekhez és megvalósítható kutatási menetrendekhez vezetett. Sok esetben a formalizálás és a speciálizálódás felaprózódást eredményezett. Az olyan témaik, mint a látás és a robotika az MI fő vonalától egyre jobban elszigetelődnek. Az MI-nek a racionális ágensben megtestesült egységesítő képe egy olyan megközelítés, amely e divergáló területeken újra egységet teremthet.

## Az intelligens ágensek kialakulása (1995-től napjainkig)

Valószínűleg az MI részproblémáiban elérte sikereken felbátorodva a kutatók elővettek a „teljes ágens” problémakörét. A teljes ágensarchitektúra legismertebb esete a SOAR, Allen Newell, John Laird és Paul Rosenbloom munkája (Newell, 1990; Laird és társai, 1987). Az ún. beágyazott mozgalom célul tűzte ki a valós környezetbe ágyazott, folytonos szenzorikus adatokat fogadó ágensek működésének a megértését. Az intelligens

ágensek szempontjából az egyik legfontosabb környezet az internet. A világhálós alkalmazásokban az MI-rendszerek annyira minden naposak lettek, hogy a „-bot” szóveződés már a minden nap nyelvbe is beépült. Ráadásul az MI-technológiák sok olyan internetes eszköznek képezik az alapját, amelyek a keresőgépek, az ajánló rendszerek és a weboldalszerkesztő rendszerek.

E könyv első kiadása (Russell és Norvig, 1995) mellett más kurrens könyv is átvette az ágensperspektívát (Poole és társai, 1998; Nilsson, 1998). A teljes ágenstervezés egyik következménye, hogy fel kell ismerni, az MI eddig elszigetelt területeit minden bizonnal valamelyest át kell szervezni, ha az eredményeket össze akarjuk kapcsolni. Ma már széles körben elfogadott, hogy az érzékelő rendszerek (látás, szonár, beszédfelismerés stb.) nem képesek a környezetről tökéletesen megbízható információt szolgáltatni. A következetettségnek és a tervkészítésnek így fel kell készülnie a bizonytalanság kezelésére. Az ágensperspektíva másik lényegi következménye, hogy az MI az ágensekkel foglalkozó más területekkel, például az irányításelmélettel és a gazdaságtannal, sokkal közelebbi kontaktusba került.

## **1.4. A MESTERSÉGES INTELLIGENCIA JELENLEGI HELYZETE**

Mit tehet az MI manapság? Néhánz erre tömör választ adni, mert annyi minden történt, és olyan sok a mielőtt részterület. Az alábbiakban bemutatunk néhány alkalmazást, másokról a könyv további részeiben szó lesz.

**Autonóm tervkészítés és ütemezés:** Több száz millió mérföldre a Földtől a NASA Remote Agent programja lett az első fedélzeti autonóm tervkészítő program, amely egy űrhajó műveleteinek ütemezését felügyelte (Jonsson és társai, 2000). A Remote Agent a terveit a Földről küldött magas szintű célokhoz generálta, és a tervek végrehajtása közben monitorozta az űrhajó működését, hibákat detektált, diagnosztizált, és visszaállította a helyes működést, ha problémák léptek fel.

**Kétszemélyes játékok:** Az IBM Deep Blue rendszere lett az első számítógépes sakkprogram, amely legyőzte a világbajnokot, amikor egy bemutató mérkőzésen 3,5 : 2,5 arányban győzedelmeskedett Garri Kaszparov felett (Goodman és Keene, 1997). Kaszparov nyilatkozata szerint egy „újfajta intelligenciát” érzett a sakktábla mögött. A Newsweek újság a mérkőzést az „agy utolsó védelmi vonalának” titulálta. Az IBM részvényei 18 milliárd dollárral emelkedtek.

**Autonóm szabályozás:** Az ALVINN számítógépes látórendszert arra tanították, hogy egy gépkocsit egy közlekedési sávot követve vezessen. A rendszert a CMU NAVLAB számítógép-vezérelt kis tehergépkocsijára helyezték, és arra használták, hogy az Egyesült Államokon keresztül elnavigáljon. A 2850 mérföldes távból a rendszer az idő 98%-ban vezetett. A maradék 2%-ért, főleg a sztrádalejáratokon, az ember vállalta felelősséget. A NAVLAB videokamerákkal rendelkezik, amelyek az ALVINN számára közvetítik az út képeit. A tanulóutakból szerzett tapasztalatok felhasználásával az ALVINN számítja ki a kormánykerék legjobb beállítását.

**Diagnózis:** Valószínűségi elemzésen alapuló orvosi diagnosztizáló rendszerek az orvosi tudományok több területén szakértő orvosok szintjén voltak képesek helytállni. Heckerman (Heckerman, 1991) leír egy esetet, amikor a nyirokcsomó-patológia egyik vezető szakembere gúnyolódó megjegyzést tesz egy pokolian nehéz esetet diagnosztizáló program javaslatára. A rendszer fejlesztője azt javasolja, hogy kérdezze meg a rendszertől a diagnózis magyarázatát.

A gép rámutatott a döntését befolyásoló fő tényezőkre, és megmagyarázta az adott esetben jelentkező tünetek bonyolult kölcsönhatását. A szakember végül egyetértett a programmal.

**Logisztikai tervkészítés:** 1991-ben, az Öböl-válság idején az amerikai haderő automatikus logisztikai tervkészítésre és a szállítás ütemezésére egy DART – Dynamic Analysis and Replanning Tool – nevű rendszert alkalmazott. A rendszer működése egyidejűleg 50 ezer (teher- és szernélyszállító-) járműre terjedt ki, figyelembe vétele a kiindulási és célállomásokat, útvonalakat és az összes paraméter közötti konfliktusfeloldást is. MI-technikák révén a terv órák alatt kész volt, szemben a heteket igénylő korábbi megoldásokkal. A Védelmi Kutatási Ügynökség (Defense Advanced Research Project Agency, DARPA) közleménye szerint, csupán ezen egyetlen alkalmazás kapcsán megtérült a DARPA által 30 éven keresztül az MI-re fordított befektetés.

**Robotika:** A mikrosebészeti manapság sok sebész robotsegédekre támaszkodik. A HipNav rendszer, miután számítógépes látási technikák segítségével létrehozta a páciens belső anatómiájának háromdimenziós modelljét, robotszabályozással irányítja a csípőprotézis behelyezését (DiGioia és társai, 1996).

**Nyelvmegértés és problémamegoldás:** A PROVERB rendszer a legtöbb embernél jobb keresztrejtvényfejtő (Littman és társai, 1999). Ehhez rendelkezik a lehetséges megfejtő szavakra vonatkozó korlátozásokkal, a régebbi keresztrejtvények nagy adatbázisával és sokféle információs forrással, szótárakat és az olyan online adatbázisokat is beleértve, mint mozcímlisták a bennük szereplő színészekkel. Így például képes megállapítani, hogy a „Nice Story” meghatározás megoldása „ETAGE”, mert az adatbázisban „Story in France/ETAGE” meghatározás/megoldás pár szerepel, és felismeri, hogy a „Nice X” és „X in France” mintáknak sokszor azonos a megoldása. A program természetesen nem tudta, hogy Nice egy város Franciaországban, a keresztrejtvényt mégis képes volt megoldani.<sup>18</sup>

Fentiek csupán példák a ma létező mesterséges intelligencia rendszerekre. Nem magia vagy sci-fi – inkább tudomány, technika és matematika, amihez bevezetőt nyújt ez a könyv.

<sup>18</sup> Az ilyen és hasonló helyeken, ahol a könyv szerzője igazán csak angolul értelmes szöfőrökkel illusztrálja a mondanivalóját, az üzenet érdekében meghagyjuk a szavak eredeti angol formáját. (A lektor)

## 1.5. ÖSSZEFOGLALÁS

Ebben a fejezetben definiáltuk az MI-t és áttekintettük az MI fejlődésének kulturális háttérét. A legfontosabb gondolatok közül néhány:

- Az egyes emberek különböző módon vélekednek az MI-ról. A két fontos felteendő kérdés az, hogy: a gondolkodás vagy a viselkedés az, ami Önt érdekli? Embereket akar modellezni, vagy egy idealizált megközelítést választ?
- Ebben a könyvben azt a nézetet fogadjuk el, hogy az intelligencia lényegében a racionális cselekvéssel kapcsolatos. Egy *intelligens ágens*, ideális esetben, az adott szituációban a legjobb cselekvéshez folyamodik. Az ilyen értelemben vett intelligens ágensek építési problémáit fogjuk tanulmányozni.
- A filozófusok (i. e. 400-ig visszamenőleg) tettek lehetővé az MI kialakulását azáltal, hogy felvetették: az elme bizonyos értelemben gépszerű, hogy valamelyen belső nyelvezetben kódolt tudásanyagon operál, és hogy a gondolat a helyes cselekvés megválasztásának eszköze.
- A matematikusok megadták a logikailag biztos, valamint a bizonytalan valószínűségi állítások manipulálásának eszközét. Megadták annak az alapjait is, hogy megértsük a számításokat és az algoritmusokról következtethessünk.
- A közigazdászok formalizálták a döntéshozatal folyamatát, hogy a döntéshozónak maximális várható hasznai biztosítson.
- A pszichológusok megerősítették azt a gondolatot, hogy az ember és az állatok információprocesszáló gépeketek tekinthetők. A nyelvészek azt mutatták ki, hogy a nyelvhasználat ezzel a modellel összhangban van.
- A számítógép-technika biztosította az MI alkalmazását lehetővé tevő „műterméket”. Az MI-programok nagyok, a működéstük lehetetlen lenne a memóriának és a sebességnek a számítógépes ipar biztosította nagyfokú fejlődése nélkül.
- Az MI történetében voltak sikeres időszakok, de megtalálhatók a téves optimizmus és a lelkesedés, valamint a finanszírozás elapadása következtében beálló hanyatlás ciklusai is. Voltak időszakok, amelyek során új kreatív megközelítések bevezetése és a legjobb ötletek szisztematikus finomítása történt meg.
- Az MI fejlődése az utolsó évtizedben a tudományos módszereknek a kísérletezésben és a megközelítések összehasonlításában való szélesebb körű alkalmazása következében felgyorsult.
- Az intelligencia elméleti alapjainak megértésében bekövetkezett jelenlegi fejlődés kéz a kézben együtt járt a valós rendszerek képességeinek javulásával. Az MI egyes részterületei jobban integrálódtak, és az MI és más tudományágak megtalálták a közös alapjukat.

## Irodalmi és történeti megjegyzések

Herb Simon a *The Science of the Artificial* c. műben tárgyalja az MI módszertani státuszát (Simon, 1981). A mű azokat a kutatási területeket tárgyalja, amelyek bonyolult műtermékekkel kapcsolatosak. Azt is bemutatja, hogy hogyan lehet az MI-t mind matematikának, mind tudománynak látni. Cohen (Cohen, 1995) áttekinti az MI kísérleti

technológiáját. Ford és Hayes (Ford és Hayes. 1995) pedig a Turing-teszt hasznosságáról fejtik ki a véleményüket.

Az MI filozófiai és gyakorlati problémáiról olvasmányos áttekintést ad John Haugeland *Artificial Intelligence: The Very Idea* c. műve (Haugeland, 1985). A kognitív tudományról néhány újkeletű forrás (Johnson-Laird, 1988; Stillings és társai, 1995; Thagard, 1996) és az *Encyclopedia of the Cognitive Sciences* (Wilson és Keil, 1999) adnak jó leírást. A modern nyelvészeti szintaktikai aspektusát (Baker, 1989), szemantikai aspektusát pedig (Chierchia és McConnell-Ginet, 1990) mutatja be jól. A számítógépes nyelvészettel Jurafsky és Martin (Jurafsky és Martin. 2000) foglalkoznak.

Az MI-kutatások kezdetétől a Feigenbaum és Feldman *Computers and Thought* (Feigenbaum és Feldman, 1963) és a Minsky *Semantic Information Processing* (Minsky, 1968) című művekben, továbbá a Donald Michie által szerkesztett *Machine Intelligence* sorozatban lehet olvasni. Webber és Nilsson (Webber és Nilsson, 1981) és Luger (Luger, 1995) nagy hatású cikkekből állítottak össze antológiákat. A *Neurocomputing* (Anderson és Rosenfeld, 1988) a neurális hálókkal foglalkozó korai cikkek gyűjteménye. Az *Encyclopedia of AI* (Shapiro, 1992) majdnem minden MI-témáról tartalmaz egy áttekintő cikket. Ezek a cikkek általában jó kiindulópontok az egyes témaörök kutatásával foglalkozó irodalomhoz.

A legfrissebb munkák a nagyobb MI-konferenciák – a kétévente megrendezett International Joint Conference on AI (IJCAI), az évenkénti European Conference on AI (ECAI) és a támogató szervezetről AAAI-ként ismert évenkénti National Conference on AI – kiadványaiban találhatók meg. Az MI általános kérdéseivel foglalkozó fő folyóiratok az *Artificial Intelligence*, a *Computational Intelligence*, az *IEEE Transactions on Pattern Analysis and Machine Intelligence*, az *IEEE Intelligent Systems* és az elektronikus formában megjelenő *Journal of Artificial Intelligence Research*. Sok, speciális területekkel foglalkozó folyóirat és konferencia is létezik, ezekről azonban a megfelelő fejezetekben lesz szó. Az MI fő szakmai szervezetei az American Association for Artificial Intelligence (AAAI), az ACM Special Interest Group in Artificial Intelligence (SIGART) és a Society for Artificial Intelligence and Simulation of Behaviour (AISB). Az AAAI *AI Magazine*-ja sok áttekintő és tematikus cikket publikál, az aaai.org weboldala pedig híreket és háttérinformációkat tartalmaz.

## Feladatok

Az alábbi gyakorlópéldáknak az a rendeltetése, hogy vitát stimuláljanak. Néhány közülük féléves házi feladatnak is alkalmas. E feladatokat már most megkísérelhetik megoldani és később, a könyv olvasását befejezve, e próbálkozásokat megismételhetik.

- 1.1.** Saját szavaival definiálja: (a) az intelligencia, (b) a mesterséges intelligencia, (c) az ágens fogalmakat.
  
- 1.2.** Olvassa el Turing eredeti cikkét az MI-ről (Turing, 1950). A cikkben Turing a célkitűzésével és az intelligenciatesztjével kapcsolatos potenciális kifogásokat tárgyalja. Melyik kifogásnak van még ma is súlya? Érvényesek-e a cáfolatai? El tud képzelni más kifogásokat is, amelyek a cikk megírását követő fejlődés-

ből adódnak? Turing a cikkében azt jósolja, hogy 2000-ben egy számítógépnek 30%-os esélye lesz, hogy teljesítsen egy 5 perces Turing-tesztet egy nem képzett kérdezővel szemben. Ésszerűnek gondolja ezt? És mondjuk 50 év múlva?

- 1.3.** Minden évben Loebner-díjjal jutalmazzák azt a programot, amely a Turing-teszt egy bizonyos verzióját legjobban képes teljesíteni. Kutassa fel a Loebner-díj legújabb győztesét és számoljon be róla. Milyen technikára támaszkodik ez a megoldás? Milyen hatással lehet ez az MI jelenlegi helyzetére?
- 1.4.** A problémáknak vannak olyan jól ismert osztályai, amelyek egy számítógép számára kezelhetetlenül nehezek, más osztályok viszont számítógépen bizonyíthatóan nem dönthetők el. Azt jelenti-e ez, hogy az MI lehetetlen?
- 1.5.** Tegyük fel, hogy Evans ANALOGY programját úgy fejlesztjük tovább, hogy a standard IQ-tesznél 200 pontot érjen el. Lesz-e akkor egy olyan programunk, ami intelligensebb az embernél? Magyarázza meg a választ!
- 1.6.** Hogyan lehetséges, hogy az önelemzés – valakinek a saját gondolatairól szóló beszámolója – pontatlan? Tévedhetek arról, amit gondolok? Vitassa meg!
- 1.7.** Tanulmányozza az MI-irodalmat, hogy eldönthesse, az alábbi feladatokat jelenleg meg lehet-e oldani számítógépen:
- (a) Egy színvonalas tenisz- (pingpong-) játszma.
  - (b) Autóvezetés Kairó központjában.
  - (c) Heti élelmiszer-vásárlás a piacon.
  - (d) Heti élelmiszer-vásárlás a vilaghálón.
  - (e) Elfogadható bridzsparti versenyszinten.
  - (f) Új matematikai tételek felfedezése és bizonyítása.
  - (g) Szándékosan humoros történet írása.
  - (h) Kompetens jogi tanácsadás egy specializált jogi területen.
  - (i) Beszélt angol nyelv valós időben történő fordítása beszélt svéd nyelvre.
  - (j) Egy bonyolult sebészeti beavatkozás levezetése.
- A ma még nem megvalósítható feladatok esetén próbálja azonosítani a nehézségeket, és megbecsülni, hogy mikorra tervezhető azok megoldása.
- 1.8.** Néhány szerző azt állította, hogy az érzékelés és a motorikus képességek az intelligencia legfontosabb komponensei, és hogy a „magasabb szintű” képességek szükségszerűen parazita jellegűek – adalékok az alattuk levő képességekhez. Nyilvánvaló, hogy az evolúció zöme és az agy nagy része az érzékeléssel és a motorikus képességekkel foglalkozik. Az MI pedig a játék vagy logikai következetet jellegű feladatokat a valós világban való érzékelésnél és cselekvésnél sok szempontból könnyebbnek találta. Nem gondolja-e, hogy az MI-nek a magasabb szintű kognitív képességekre irányuló hagyományos összpontosítása téves célkitűzés?
- 1.9.** Az evolúció miért eredményezett racionálisan cselekvő rendszereket? Milyen célokat hivatottak az ilyen rendszerek teljesíteni?

- 1.10. Racionálisak-e a reflexszerű cselekvések (mint például a forró tűzhelytől elrántani a kezet)? Intelligensek-e?
- 1.11. „Egy számítógép nem lehet intelligens – csak azt teszi, amit a programozó mond neki.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?
- 1.12. „Az állatok nem lehetnek intelligensek – csak azt teszik, amit a géneik mondanak nekik.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?
- 1.13. „Az állatok, az emberek és a számítógépek nem lehetnek intelligensek – csak azt teszik, amit a részüket alkotó atomoknak a fizika törvényei parancsolnak.” Igaz-e az állítás második része, és vajon implikálja-e az elsőt?

## 2. INTELLIGENS ÁGENSEK

Ebben a fejezetben az ágensek természetét, legyen az tökéletes vagy sem, a környezetek különbözőségét és az ágenstípusok ezekből származó sokféleségét tárgyaljuk.

Az 1. fejezet a **racionális ágens** (*rational agent*) fogalmát úgy vezette be, mint a mesterséges intelligencia tárgyalásának általunk választott központi módszerét. Ebben a fejezetben pontosítjuk ezt a koncepciót. Látni fogjuk, hogy a racionálitás konцепciója az ágensek széles körére alkalmazható mindenféle elköpzelhető környezetben. Az a tervünk a könyvben, hogy ezt a koncepciót néhány olyan tervezési elv megfogalmazásához használjuk, amelyekkel sikeres ágenseket építhetünk – olyan rendszereket, melyek joggal hívhatók **intelligensnek**.

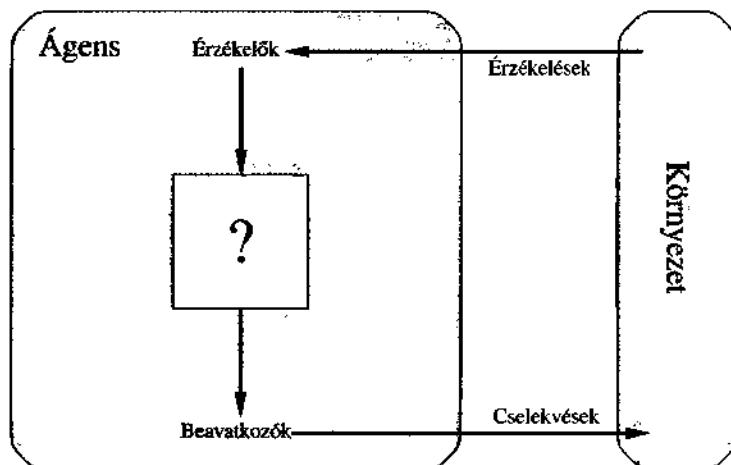
Ágensek, környezetek és a köztük lévő kapcsolatok vizsgálatával kezdünk. Annak megfigyelése, hogy egyes ágensek jobban viselkednek, mint mások, a racionális ágens ötletéhez vezet majd – olyan ágenshez, amelyik olyan jól viselkedik, amennyire csak lehet. Az, hogy mennyire viselkedhet jól egy ágens, a környezet természetén múlik; bizonyos környezetek sokkal nehezebbek, mint mások. A környezetek nagyjából csoportosításával megmutatjuk, hogy egy környezet tulajdonságai hogyan befolyásolják a környezetnek megfelelő ágensek tervezését. Bemutatunk számos egyszerű ágens-tervezési „csontvázat”, melyeket a könyv hátralevő fejezeteiben tovább részletezünk.

### 2.1. BEVEZETÉS

Egy **ágens** (*agent*) bármí lehet, amit úgy tekinthetünk, mint ami az **érzékelői** (*sensors*) segítségével érzékeli a **környezetét** (*environment*), és **beavatkozói** (*actuators*) segítségével megváltoztatja azt. A 2.1. ábra ezt az egyszerű elköpzelést szemlélteti. Az emberi ágensek van szeme, füle és egyéb szervei az érzékelésre, és keze, lába, szája és egyéb testrészei a beavatkozásra. A robotágens kamerákat és infravörös távolsági keresőket használ érzékelőként, és különféle motorokat beavatkozóként. A szoftverágens billentyűléjtéset, fájltartalmakat és hálózati adatszolgálatokat fogad érzékelőinek beemeneteként, és képernyőn történő kijelzéssel, fájlok írásával, hálózati csomagok küldésével avatkozik be a környezetébe. Azzal az általános feltételezéssel fogunk elni, hogy minden ágens képes saját akciójainak érzékelésére (de nem minden látja azok hatását).

Az **érzékelés** (*percept*) fogalmat használjuk az ágens érzékelő bemeneteinek leírására egy tetszőleges pillanatban. Egy ágens **érzékelési sorozata** (*percept sequence*) az ágens érzékeléseinek teljes története, minden, amit az ágens valaha is érzékelt. Általánosságban, *egy adott pillanatban egy ágens cselekvése az addig megfigyelt teljes érzékelési sorozatától függhet*. Ha az összes lehetséges érzékelési sorozathoz meg tudjuk





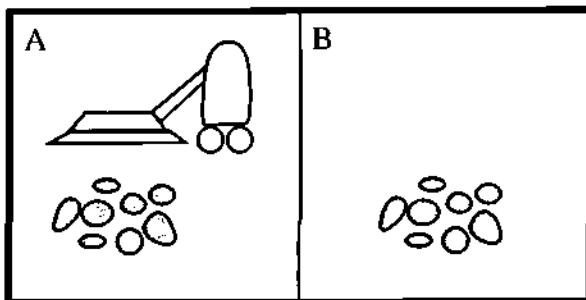
2.1. ábra. Az ágensek környezetükkel érzékelőkön és beavatkozókon keresztül állnak kapcsolatban

határozni az ágens lehetséges cselekvéseit, akkor lényegében minden elmondunk az ágensről. Matematikailag megfogalmazva azt mondhatjuk, hogy az ágens viselkedését az **ágensfüggvény (agent function)** írja le, ami az adott érzékelési sorozatot egy cselekvésre képezi le.

Elképzelhetjük **táblázatos formában** az ágensfüggvényt, amely forma mindenféle ágenst leír; az ágensek többségére ez igen nagy táblázatot jelentene – valójában végig nem nagyon, hacsak nem korlátozzuk a figyelembe veendő érzékelési sorozatok hosszát. Ha adott egy ágens, amivel kísérletezhetünk, akkor elméletileg megalkothatjuk ezt a táblázatot az összes lehetséges érzékelési sorozat kipróbálásával és az ágens válaszul végrehajtott cselekvéseinek feljegyzésével.<sup>1</sup> Természetesen a táblázat az ágens *külső* jellemzése. Egy mesterséges ágens *belsejében* az ágensfüggvényt egy **ágensprogram (agent program)** valósítja meg. Fontos, hogy megkülönböztessük e két dolgot. Az ágensfüggvény egy absztrakt matematikai leírás, az ágensprogram egy konkrét implementáció, amely az ágens architektúráján működik.

Ezen ötletek illusztrálására egy igen egyszerű példát fogunk használni – a 2.2. ábrán látható porszívóvilágot. A világ annyira egyszerű, hogy minden le tudunk írni, ami megtörténik. Továbbá ez egy kitalált világ, így sokféle variációt kitalálhatunk. A világban csak két helyszín van: az *A* és *B* négyzetek. A porszívóágens észleli, hogy melyik négyzetben van, valamint azt, hogy van-e ott piszok, vagy nincs. Lehetséges akciói: mozoghat balra vagy jobbra, felszívhatalja a port, valamint nem csinál semmit. Egy egyszerű ágensfüggvény a következő: ha az aktuális négyzet koszos, szívd fel a koszt, egyébként menj át a másik négyzetbe. Ezen ágensfüggvény táblázatának egy részlete a 2.3. ábrán látható. Egy, az ágensfüggvényt megvalósító egyszerű ágensprogramot a fejezet későbbi részében, a 2.8. ábrán mutatunk be.

<sup>1</sup> Amennyiben az ágens cselekvésének kiválasztásában a véletlennek is szerepe van, úgy minden sorozatot többször kellene kipróbálnunk, hogy az egyes cselekvések valószínűségeit meghatározhassuk. Látszólag úgy tűnhet, hogy véletlenszerűen cselekedni botorság, a későbbiekben azonban látni fogjuk, hogy ez lehet nagyon intelligens viselkedés is.



2.2. ábra. A porszívóvilág minden össze két helyszínnel

Érzékelési sorozat	Cselekvés
[A, Tiszt]	Jobbra
[A, Piszkos]	Felszívás
[B, Tiszt]	Balra
[B, Piszkos]	Felszívás
[A, Tiszt], [A, Tiszt]	Jobbra
[A, Tiszt], [A, Piszkos]	Felszívás
...	...
[A, Tiszt], [A, Tiszt], [A, Tiszt]	Jobbra
[A, Tiszt], [A, Tiszt], [A, Piszkos]	Felszívás
...	...

2.3. ábra. A 2.2. ábrán szereplő porszívóvilág egy egyszerű ágensfüggvénye részleges táblázatos formában

A 2.3. ábrára nézve láthatjuk, hogy sokféle porszívóvilág-beli ágens definiálható egyszerűen a jobb oldali oszlop különböző kitöltésével. Ezek után a nyilvánvaló kérdés a következő: *Mi a táblázat helyes kitöltésének módja?* Más szavakkal, mitől lesz egy ágens jó vagy rossz, intelligens vagy buta. Ezeket a kérdéseket a következő részben fogjuk megválaszolni.

Mielőtt lezárnánk ezt a részt, fontos megjegyeznünk, hogy az ágens fogalom bevezetésének a célja nem a világ ágensekre és nem ágensekre történő felosztása, hanem egy olyan eszköz megeremítése, amivel rendszereket elemezhetünk. Egy kézi számológép is tekinthető ágensnek, amely azt a cselekvést választja, hogy „4”-et ír a kijelzőjére, ha „2 + 2 =” érzékelési sorozatot kap, de egy ilyen felfogás aligha segít minket a számológép megértésében.

## 2.2. JÓ VISELKEDÉS: A RACIONALITÁS KONCEPCIÓJA

Egy racionális ágens (*rational agent*) olyan, amely helyesen cselekszik – elméletileg megfogalmazva az ágensfüggvény táblázatában minden bejegyzés helyesen van kitölve. Helyesen cselekedni nyilvánvalón jobb, mint helytelenül, de mit is jelent helyesen cselekedni? Első közelítésben azt mondjuk, hogy a helyes cselekedet az, amely az

ágenst a legsikeresebbé teszi. Tehát szükségünk lesz egy módszerre, amivel a sikeresség mérhető. A környezetnek, valamint az ágens érzékelőinek és beavatkozóinak leírásával együtt ez az ágens feladatát teljes mértékben specifikálja. Ezáltal precízebben definiálhatjuk, mit is jelent racionálisnak lenni.

## Teljesítménymértékek

A **teljesítménymérték** (*performance measure*) testesíti meg az ágens sikerességének kritériumát. Ha egy ágenst elhelyezünk a környezetében, akkor cselekvések sorozatával válaszol az érzékeléseire. A cselekvések sorozatának hatására a környezet állapotok sorozatán halad végig. Ha ez a sorozat kívánatos, akkor az ágens jól teljesített. Nyilvánvalóan nincs egyetlen olyan rögzített mérték, amely minden ágens számára megfelelő lenne. Megkérdezhetjük az ágens szubjektív véleményét, hogy mennyire érzi magát elégedettnek a saját teljesítményével, de egyes ágensek képtelenek lesznek válaszolni, míg mások becsapják magukat.<sup>2</sup> Ezért valamilyen objektív teljesítménymérékhez fogunk ragaszkodni, tipikusan egy olyanhoz, amit az ágens tervezője határozott meg.

Példaként tekintsünk az előző részben megismert porszívó ágenst. Teljesítménymértekként javasolhatjuk az egy nyolcórás műszakban feltakarított por mennyiséget. Egy racionális ágenssel, természetesen, az ember azt kapja, amit kér. Azaz a racionális ágens azzal maximalizálhatja ezt a teljesítménymértéket, hogy feltakarítás után kiönti a szemetet, majd újra feltakarítja és így tovább. Egy finomított mérték a tiszta padló elérését díjazná. Például minden tiszta négyzet egy pontot érhet minden időlépésben (talán valamilyen levonással figyelembe véve az elfogyasztott elektromos energiat, illetve a kellett zaj mennyiséget is). Általános szabályként megfogalmazható, hogy a teljesítményméréket jobb aszerint megállapítani, hogy mit akarunk elérni a környezetben, mint aszerint, hogy miképp kellene az ágensnek viselkednie.

A teljesítménymérték kiválasztása nem minden könnyű. Például a „tiszta padló” előző paragrafusbeli fogalma a tisztaság időátlagán alapul. Még ugyanaz az átlagos tisztaság is elérhető két különböző ágenssel, az egyik közepes aktivitással dolgozik állandóan, míg a másik energikusan rövid ideig, aztán hosszú szüneteket tart. Hogy melyik a javasolt módszer, az látszólag a házvezetés tudományának kérdése, valójában egy mély filozófiai kérdés, messzibe mutató következményekkel. Mi a jobb: a túlzások vakmerő világa vagy a biztonságos, de unalmas létezés? Mi a jobb: egy gazdaság, ahol mindenki egyformán mérsékelt szegény, vagy egy másik, ahol vannak nagyon gazdagok és nagyon szegények? Ezen kérdések megválaszolását házi feladatként bízzuk a szorgalmass olvasóra.

<sup>2</sup> Az emberi ágensekre különösen jellemző a „savanyú a szóló” effektus – azaz hogy nem is akarják igazán azt, amit nem sikerült megkapniuk, mint például: „Nos, sebaj, nem is akartam annyira azt a buta Nobel-díjat”.

## Racionalitás

Négy dolgon múlik az, hogy egy adott pillanatban mi racionális. Ezek:

- A siker fokát mérő teljesítménymérték.
- Az ágens eddigi tudása a környezetről.
- A cselekvések, amiket az ágens képes végrehajtani.
- Az ágens érzékelési sorozata az adott pillanatig.

 Ez elvezet a racionális ágens (*rational agent*) definíciójához:

*Az ideális racionális ágens minden egyes észlelési sorozathoz a benne található tények és a beépített tudása alapján minden elvárható dolgot megtesz a teljesítménymérték maximalizálásáért.*

Vizsgáljuk meg az egyszerű porszívóágenst, amelyik megtisztítja a négyzetet, ha az koszos, és átmegy a szomszédos négyzetbe, ha nem az! Az ágensfüggvényt táblázatosan a 2.3. ábra adja meg. Ez egy racionális ágens? Attól függ! Először is, meg kell mondanunk mi a teljesítménymérték, mit tudunk a környezetről, és milyen érzékelői és beavatkozói vannak az ágensnek. Tegyük fel a következőket:

- A teljesítménymérték egy ponttal jutalmaz minden tiszta négyzetet minden időpillanatban egy 1000 időpillanatból álló „élettartam” alatt.
- A környezet „geográfiája” *a priori* ismert (2.2. ábra), de a piszok eloszlása és az ágens kezdeti pozíciója nem. A tiszta négyzetek tiszták maradnak, a felszívás pedig megtisztítja az aktuális négyzetet. A *Balra* és *Jobbra* cselekvések balra és jobbra mozgatják az ágenst, kivéve ha ezzel kikerülne a környezetből, mely esetben az ágens nem mozdul.
- A következő cselekvések léteznek: *Balra*, *Jobbra*, *Szív* és *Semmittevés* (azaz nem tesz semmit sem).
- Az ágens helyesen észleli jelenlegi helyzetét és azt, hogy van-e ott kosz.

Azt állítjuk, hogy *ezen feltételek mellett* az ágens igenis racionális: a várható teljesítménye legalább annyira jó, mint bármely más ágensé. A 2.4. feladatban azt kérjük majd, hogy ezt az állítást bizonyítsa be.

Könnyen belátható, hogy ugyanez az ágens más körtülmények között irrationális lenne. Például ha minden koszt feltakarított, akkor szükségtelenül oszcillálni fog ide-oda; ha a teljesítménymérték tartalmaz egy büntetőpontot minden balra vagy jobbra történő mozgásért, akkor az ágens egészen gyengén teljesít. Egy jobb ágens abban az esetben, ha meggyőződött arról, hogy az összes négyzet tiszta, nem csinálna semmit. Ha a tiszta négyzetek ismét koszossá válhatnak, az ágensnek időnként ellenőriznie kellene őket, és kitakarítania, ha szükséges. Ha a környezet geográfiája ismeretlen, az ágensnek fel kell térképeznie ahelyett, hogy az *A* és *B* négyzetekhez köti magát. A 2.4. feladatban azt kérjük majd, hogy tervezzen ezekre az esetekre ágenseket.

## Mindentudás, tanulás és autonómia

Ügyelnünk kell a racionalitás és a **mindentudás (omniscience)** különválasztására. Egy minden tudó ágens tudja cselekedetei *valódi* kimenetelét, és ennek megfelelően cselekedhet, de a gyakorlatban a minden tudás lehetetlen. Vegyük a következő példát: egy nap a Nagykörúton sétálok és egy régi barátomat látom az út túloldalán közeledni. Az úton nincs forgalom, és ráérek, így – racionális lévén – elindulok a másik oldalra. Mindeközben 5000 kilométer magasan egy utasszállító repülőgép raktárajtaja leesik,<sup>3</sup> és mielőtt még átérnék az út túloldalára, rám esik és kilapít. Irracionális voltam, amikor át akartam kelni az úton? Valószínűtlen, hogy a gyászjelentésem így fog szólni: „egy idiota megpróbált keresztlümmenni az utcán”.

Ez a példa azt mutatja, hogy a racionalitás nem azonos a tökéletességgel. A racionalitás az *elvárt teljesítményt maximalizálja*, míg a tökéletesség a *tényleges teljesítményt*. Az, hogy elállunk az ágensekkel szemben támasztott tökéletesség követelményétől, nem csak a velük szembeni méltányosság kérdése. Az a helyzet, hogy ha azt a cselekvést várjuk el egy ágenstől, ami a megtörtént után a legjobbnak bizonyul, akkor lehetetlen lesz egy ilyen ágens megalkotása – hacsak nem javítjuk a kristálygömbök vagy az időgépek teljesítményét.

Ezek alapján a racionalitás definícióink nem követel minden tudást, hiszen a racionális választás csak az *adott pillanatig felépített érzékelési sorozattól* függ. Azt is biztosítunk kell, hogy tudtunk nélkül se engedjük meg az ágensnek, hogy tudatosan unintelligens cselekvésbe kezdjen. Például ha egy ágens nem néz körül minden irányban, mielőtt egy forgalmas utat keresztek, akkor az érzékelési sorozata nem fogja neki megmondani, hogy egy hatalmas kamion közeledik nagy sebességgel. Azt mondja-e a racionalitás definícióink, hogy ebben az esetben rendben átkelhet az úton? Egyáltalán nem! Először is, az útkeresztes nem lenne racionális ezen információs zegény észlelési sorozat mellett: az átkelés szétnézés nélkül túlságosan rizikós. Másodszor, egy ideális racionális ágensnek a „szétnézés” cselekvést kellene választania az útra lépés előtt, mivel ez segíti a teljesítmény maximalizálásában. A *hasznos információk beszerzése érdekében* véghezvitt – általában **információgyűjtésnek (information gathering)** hívott – cselekedetek a racionalitás fontos részét jelentik, részletesebben a 16. fejezetben lesz szó róluk. Az információgyűjtés másik példáját a **felfedezés (exploration)** szolgáltatja, amit egy porszívóágensnek egy ismeretlen környezetben meg kell tennie.

Definícióink nemcsak azt követeli meg a racionális ágenstől, hogy információt gyűjtsön, hanem azt is, hogy amennyit csak lehet **tanuljon (learn)** a megfigyeléseiből. Az ágens kezdeti konfigurációja valamilyen előzetes tudást tükrözhet a környezetről, de ahogy az ágens tapasztalatot szerez, ez a tudás módosulhat és átértékelődhet. Vannak szélsőséges esetek, amikor a környezet *a priori* teljesen ismert. Ilyen esetekben az ágensnek nem kell érzékelnie vagy tanulnia, egyszerűen csak helyesen cselekszik. Természetesen az ilyen ágensek nagyon sérülékenyek. Vegyük például az egyszerű ganajtúró bogarat. Miután kiásta a fészket és elhelyezte a tojásait, a közeli halomból egy ganajlabdát készít, hogy elzárja a fészek bejáratát. Ha útközben a labdát eltávolítjuk a fogói közül, a bogár folytatja útját, és a nem létező labdával is eljátsza a fészek lezárását,

<sup>3</sup> Lásd N. Henderson: Sürgősen új ajtózárakat a Boeing 747 jumbo jetek számára. *Washington Post*, 1989. augusztus 24.

miközben észre sem veszi, hogy a labda hiányzik. Az evolúció beépített egy feltételezést a bogár viselkedésébe, és amikor ez megsérül, sikertelen viselkedés lesz az eredménye. Kicsit intelligensebb a szöcskeölő darázs. A nőstény ás egy lyukat, keres és megcsíp egy hernyót, elcipeli a lyukhoz, bemegy a lyukba megnézni, hogy minden rendben van-e, majd bevonszsolja a hernyót és lerakja a tojásait. A hernyó szolgál a tojások kikelése után táplálékul. Eddig jó, de ha egy rovarkutató pár centiméterrel odébb rakja a hernyót, miközben a darázs a lyukat ellenőrzi, az visszalép a hernyó odacipelése fázishoz, és a ter-vét változtatás nélkül folytatja onnan, akár tucatnyi hernyóelmozdító közelbelépés után is. A darázs képtelen megtanulni, hogy öröklött terve hibás, így nem fogja megváltoztatni.

A sikeres ágensek három különböző periódusra bontják az ágensfüggvény kiszámításának feladatát: amikor az ágenst tervezik, a számítás egy részét tervezőik végez; amikor megfontolja a következő cselekvését, az ágens további számításokat végez; és amikor tanul a tapasztalataiból, még további számításokat végez annak előtöntésére, hogy hogyan módosítsa a viselkedését.

Addig a szintig, míg az ágens a tervezői által beépített tudásra épít és nem saját megfigyeléseire, azt mondjuk, hogy az ágens **nem autonóm (autonom)**. Egy racionális ágensnek autonómnak kell lennie – minden, amit csak megtanulhat, meg kell tanulnia ahhoz, hogy a hiányos vagy hibás előzetes tudását kompenzálna. Például az a porszívó-ágens, amelyik megtanulja előreláttni, hogy hol és mikor tűnik fel további szemét, jobban fog dolgozni, mint amelyik nem. Praktikus okból ritkán kell teljes autonómiát biztosítani már a kezdetektől: amikor egy ágensnek nincs tapasztala, vagy csak kevés van, akkor véletlenszerűen kellene cselekednie, hacsak a tervezője nem ad valamilyen segítséget. Így, mint ahogy az evolúció ellátja az állatokat elegendő beépített reflexsel a túléléshez addig, míg saját maguk is megtanulják minden, ami a túléléshez szükséges, a mesterséges intelligens ágensek kezdő tudással és tanulási képességgel való ellátása is ésszerű lenne. Miután elegendő tapasztalatot szerzett a környezetéről, a racionális ágens viselkedése gyakorlatilag *függetlené* válhat a beépített előzetes tudásától. Ily módon a tanulás alkalmazásával olyan ágens tervezhető, amely sokféle környezetben is sikeres lesz.

## 2.3. A KÖRNYEZETEK TERMÉSZETE

Most, hogy már rendelkezünk a racionálitás definíciójával, már majdnem készen állunk arra, hogy racionális ágensek építéséről gondolkodjunk. Először azonban meg kell vizsgálnunk a **feladatkörnyezeteket (task environments)**, amelyek lényegében a „problémák”, amelyekre a racionális ágensek jelentik a „megoldásokat”. Először azt mutatjuk meg, hogyan kell a feladatkörnyezetet meghatározni, több példával illusztrálva a folyamatot. Ezek után megmutatjuk, hogy többféle feladatkörnyezet létezik. A környezet típusa közvetlenül befolyásolja az ágensprogram megfelelő tervezését.

### A környezet meghatározása

Az egyszerű porszívóágens racionálitásának vizsgálatakor specifikálnunk kellett egy teljesítménymértéket, a környezetet és az ágens beavatkozót és érzékelőit. Mindezeket a **feladatkörnyezet (task environment)** címszó alatt fogjuk egyesíteni. A betűszavak-

Ágenstípus	Teljesítménymérték	Környezet	Beavatkozók	Érzékelők
Taxisofőr	Biztonságos, gyors, törvényes, kényelmes utazás, maximális haszon	Utak, egyéb forgalom, gyalogosok, ügyfelek	Kormány, gáz, fék, index, kiürítő, kijelző	Kamerák, hangradar, sebességmérő, GPS (műholdas pozicionáló), kilométeróra, motorérzékelők, billentyűzet

2.4. ábra. Egy automatizált taxi feladatkörnyezetének TKBÉ-leírása

ban gondolkodók számára ezt **TKBÉ – Teljesítmény, Környezet, Beavatkozók, Érzékelők (Performance, Environment, Actuators, Sensors)** leírásnak hívjuk. Egy ágens tervezése során az első lépésnek mindenkor a feladatkörnyezet lehető legteljesebb meghatározásának kell lennie.

A porszívóvilág egyszerű példa volt, vegyünk most egy bonyolultabbat: az automatizált taxisofőr példáját. Ezt a példát fogjuk a fejezet hátralevő részében használni. Ki kell emelnünk, mielőtt az olvasó megrémülné, hogy a teljesen automata taxisofőr jelenleg valamelyest túlmutat a létező technológia lehetőségein. (A 60. oldalon található egy létező sofőr robot leírása, de az „Intelligent Transportation Systems” konferencia legutóbbi kiadványait is érdemes megnézni.) A teljes autóvezetési feladat rendkívül nyitott végű. Nincs határa a lehetségesen felbukkanó körülmények újszerű kombinációinak – egy másik ok arra, hogy ezt válasszuk vizsgálatunk tárgyául. A 2.4. ábra összefoglalja a taxi feladatkörnyezetének TKBÉ-leírását. A következőkben részletesen tárgyaljuk az egyes elemeket.

Elsőként: mi az a **teljesítménymérték**, amit az automata taxisofőrünk elő állítunk? Elvárt jellemzők a helyes célállomás elérése; az üzemanyag-fogyasztás és az elhasználódás minimalizálása; az út költségének és/vagy idejének minimalizálása; a közlekedési szabályok megszegésének és más vezetők megzavarásának minimalizálása; a biztonság és utaskényeleml maximalizálása; a haszon maximalizálása. Nyilvánvalóan ezen célok egy része konfliktusban van egymással, kompromisszumokra lesz szükség.

A következő: mi a taxi vezetési **környezete**? minden taxisofőrnek több típusú út-tal kell szembenéznie, a vidéki és városi utaktól egészen a tizenkét sávos autópályáig. Az utak más forgalommal, gyalogosokkal, kóbor állatokkal, útkarbantartással, rendőrautókkal, tócsákkal és gödrökkel is rendelkeznek. A taxinak a lehetséges és a jelenlegi utasokkal is kapcsolatban kell állnia. Van továbbá néhány opcionális lehetőség. Lehet, hogy a taxinak Dél-Kaliforniában kell üzemelnie, ahol a hó ritkán jelent problémát, vagy Alaszkában, ahol ritkán nem. Közlekedhet mindenkor az út jobb oldalán, vagy szeretnénk, hogy elég rugalmat legyen ahoz, hogy a bal oldalon is vezesszen abban az esetben, ha Nagy-Britanniában vagy Japánban akarjuk működtetni. Nyilvánvalóan minél korlátozottabb a környezet, annál egyszerűbb a tervezési probléma.

Az automata taxisofőr rendelkezésére álló **beavatkozók** lényegében ugyanazok lesznek, amelyek egy emberi vezető számára adottak: a motor vezérlése a gázpedál segítségével,

Ágenstípus	Teljesítménymérték	Környezet	Beavatkozók	Érzékelők
Orvosi diagnosztikai rendszer	Egészséges páciens, költségek és perek minimalizálása,	Páciens, kórház, személyzet	Kérdezések kijelzése, vizsgálatok, diagnózisok, kezelések, beszámolók	Tünetek, leletek, páciensek válaszainak bevitele billentyűzeten
Műholdas képfeldolgozó rendszer	Helyes képkategorizálás	Orbitális műhold földi kapcsolata	A kép kategorizálásának kijelzése	Színes képpontmátrixok
Alkatrész-felvezető robot	A helyes tartályokban levő alkatrészek aránya	Futószalag alkatrészekkel, tartályok	Csuklós kar és fogó	Kamera, csuklószög-érzékelők
Olajfinomító-vezérlő	A tisztaság, hozam és biztonság maximálizálása	Finomító, operátorok	Szelepek, pumpák, fűtők és kijelzők	Hőmérséklet, nyomás, kémiai érzékelők
Interaktív angol tanár	A tanulók teszteredményeinek maximálizálása	Tanulók egy csoportja, tesztelő hivatal	Gyakorlatok, javaslatok és javítások kijelzése	Billentyűzet bevitel

2.5. ábra. Példák ágenstípusokra és TKBÉ-leírásuk

kormányzás és fékezés. Ezek mellett szükséges egy képernyő vagy beszédszintetizátor a kimenet megjelenítéséhez az utasok számára, és valószínűleg valamilyen mód arra, hogy más járművekkel kommunikálhasson, udvariasan vagy másképp.

A vezetési környezetben levő céljai eléréséhez a taxinak tudnia kell, hogy hol van, mi más van az úton, és hogy milyen gyorsan halad. Alapvető érzékelői között lenni kell ezért egy vagy több vezérelhető tv-kamerának, sebességmérőnek és kilométerórának. A jármű helyes irányításához – különösen a kanyarokban – szüksége lesz gyorsulásmérőre, ezenkívül ismernie kell a jármű mechanikai állapotát, így szüksége lesz a szokásos motor- és elektromosrendszer-érzékelők rendszerére is. Lehetnek olyan műszerei is, amelyek nem érhetők el az átlagos emberi sofőrk számára: egy műholdas globális pozicionáló rendszer (GPS), amely megadja a pontos pozícióját egy elektronikus térképen, vagy infravörös- és hangradar-érzékelők más autótól és tárgyaktól való távolságok érzékelésére. Végül szüksége lesz egy mikrofonra vagy billentyűzetre, aminek a segítségével az utasok az úti céljukat megadhatják.

A 2.5. ábrán felvázoltuk az alapvető TKBÉ-elemeket számos további ágenstípusra. További példák a 2.5. feladatban jelennek meg. Egyes olvasóink számára meglepő lehet, hogy az ágenstípusok listájában olyan programokat is szerepeltekünk, amelyek a billentyűzet bemenetének és a karakteres képernyő kimenetének teljesen mesterséges környezetében működnek. „Igen”, mondhatja valaki, „ez nem egy valódi környezet, ugye?”. A tény az, hogy nem a „valódi” és a „mesterséges” környezetek közötti különbségtétel számít, hanem az ágens viselkedése, a környezet keltette érzékelési sorozat, valamint a teljesítménymérték közötti viszony komplexitása. Egyes „valódi” környezetek valójában igen egyszerűek. Például egy robot, amit a futószalagon érkező alkatrészek vizsgálatára terveztek, sok egyszerűsítő feltevéssel élhet: a világítás állandó, a futószalagon csak általa ismert alkatrészek vannak, és csak két cselekvés létezik (elfogad vagy elutasít).

Ezzel ellentétben bizonyos **szoftverágensek** (vagy szoftverrobotok, azaz **szoftbotok**) részletgazdag, korlátok nélküli környezetekben léteznek. Képzeljünk el egy szoftbotot, amelyik egy nagy utasszállító gép szimulátorával repül. A szimulátor egy részletgazdag, komplex környezet, amelyben más repülőgépekkel kapcsolatos és földi műveletek is vannak, és a szoftverágensnek valós időben kell választania az akciók széles választékából. Vagy képzeljünk el egy szoftbotot, amelyet internetes hírforrások figyelésére terveztek, és az érdekes híreket megmutatja az ügyfeleinek. A helyes működéshez szüksége lesz valamilyen természetes nyelv feldolgozási képességre, meg kell tanulnia, hogy melyik ügyfelét mi érdekli, és szüksége lesz a tervez dinamikus megváltoztatására – például amikor az egyik hírforráshoz megszűnik a kapcsolata vagy amikor egy új forrás létesül. Az internet egy olyan környezet, amelynek komplexitása versenyez a fizikai világéval, és amelynek lakói között sok mesterséges ágens van.

## A környezetek tulajdonságai

Az MI-ben felmerülő feladatkörnyezetek választéka nyilvánvalóan hatalmas. Mindazonáltal meghatározhatunk viszonylag kevés számú dimenziót, amelyek mentén a feladatkörnyezeteket kategorizálhatjuk. Ezek a dimenziók nagymértékben meghatározzák a helyénvaló ágenstervezést és az ágensimplementációk alaptechnikáinak alkalmazhatóságát. Először felsoroljuk a dimenziókat, azután több feladatkörnyezetet elemzünk az elkövetések bennutasására. Az itt szereplő definíciók informálisak – a későbbi fejezetek sokkal pontosabban fogalmazzák meg a környezeteket, és mindegyik fajtára példákat mutatnak majd.

- **Teljesen megfigyelhető (fully observable)** vagy **részlegesen megfigyelhető (partially observable)**. Ha az ágens szenzorai minden pillanatban hozzáférést nyújtanak a környezet teljes állapotához, akkor azt mondjuk, hogy a környezet teljesen megfigyelhető.<sup>4</sup> Egy környezet részlegesen teljesen megfigyelhető, ha az érzékelők minden olyan aspektusát észlelik, amelyek a cselekvés kiválasztásához relevánsak – a relevancia pedig a teljesítménymértéktől függ. A teljesen megfigyelhető környezetek kényelmesek, mivel az ágensnek nem kell semmilyen belső állapotot nyilvántartania a környezet nyomon követéséhez. Egy környezet lehet részlegesen megfigyelhető a zajos és pontatlan szenzorok miatt, vagy mivel az állapot egyes részei egyszerűen nem szerepelnek a szenzorok adatai között – például egy helyi koszérzékelő szenzorral rendelkező porszívóágens nem tudja megmondani, van-e piszok más négyzetekben, és egy automata taxi nem láthatja, hogy más vezetők mit gondolnak.
- **Determinisztikus (deterministic)** vagy **sztochasztikus (stochastic)**. Amennyiben a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott csele-

<sup>4</sup> A könyv első kiadása a **hozzáférhető (accessible)** és a **nem hozzáférhető (inaccessible)** kifejezéseket használta a teljesen és a részlegesen megfigyelhető (fully és partially observable) helyett; a **nemdeterminisztikus (nondeterministic)** a sztochasztikus (stochastic) helyett, valamint a **nem eplzödszerű (nonepisodic)** a sorozatszerű (sequential) kifejezés helyett szerepelt. Az új terminológia jobban illeszkedik az elterjedt gyakorlatba.

vés teljesen meghatározza, akkor azt mondjuk, hogy a környezet determinisztikus, egyébként sztochasztikus. Teljesen megfigyelhető, determinisztikus környezetben az ágensnek elvben nem kell a bizonytalansággal töröndnie. Amennyiben azonban a környezet részlegesen megfigyelhető, úgy sztochasztikusnak *tűnhet*. Különösen ígaz ez, ha a környezet összetett, nehezen teszi lehetővé nem megfigyelhető aspektusainak követését. Így gyakran jobb az *ágens szemszögéből* determinisztikusnak vagy sztochasztikusnak tekinteni egy környezetet. A taxivezetés ebben az értelemben nyilvánvalóan sztochasztikus, mivel senki sem tudja megjósolni a forgalmat pontosan, továbbá egy motor minden figyelemzettel nélkül lerobbanhat, és a kerekek váratlanul kidurranhatnak. A porszívóvilág, ahogy leírtuk, determinisztikus, de egyes variációk tartalmazhatnak sztochasztikus elemeket, például véletlenszerűen megjelenő piszkot vagy megbízhatatlan szívási mechanizmust (2.12. feladat). Ha a környezet más ágensek cselekvéseit leszámítva determinisztikus, akkor azt **stratégiainak (strategic)** nevezzük.

- **Epizódszerű (episodic) vagy sorozatszerű<sup>5</sup> (sequential).** Epizódszerű környezetben az ágens tapasztala elemi „epizódokra” bontható. minden egyes epizód az ágens észleléseiből és egy cselekvésből áll. Nagyon fontos, hogy a következő epizód nem függ az előzőben végrehajtott cselekvéstől. Epizódszerű környezetekben az egyes epizódokban az akció kiválasztása csak az aktuális epizódtól függ. Sok osztályozási feladat epizódszerű. Például az összeszerelő soron levő hibás alkatrészeket észlelő ágens minden egyes döntését az aktuális alkatrész alapján hozza, függetlenül a korábbi döntésekétől, továbbá az aktuális döntés nem befolyásolja, hogy a következő alkatrész hibás lesz-e. Másrészt, sorozatszerű környezetekben az aktuális döntés befolyásolhat minden továbbit. A sakk és a taxivezetés sorozatszerű: a rövid távú akciók minden esetben hosszú távú következményekkel járhatnak. Az epizódszerű környezetek sokkal egyszerűbbek a sorozatszerűknél, hiszen az ágensnek nem kell előre gondolkodnia.
- **Statikus (static) vagy dinamikus (dynamic).** Ha a környezet megváltozhat, amíg az ágens gondolkodik, akkor azt mondjuk, hogy a környezet az ágens számára dinamikus; egyébként statikus. A statikus környezetekkel egyszerű bínni, mivel az ágensnek nem kell állandóan a világot figyelnie, miközben dönt a cselekvés felől, és nem kell az idő műlásával sem töröndnie. Másrészt, a dinamikus környezetek állandóan azt kérdezik az ágenstől, hogy mit akar tenni; ha még nem döntött el, az annak számít, hogy úgy döntött, hogy nem tesz semmit. Ha a környezet nem változik az idő előrehaladtával, de az ágens teljesítménymértéke igen, akkor azt mondjuk, hogy a környezet **szemidinamikus (semidynamic)**. A taxivezetés nyilvánvalóan dinamikus: a többi autó és a taxi továbbhalad, miközben a vezetési algoritmus azon bizonytalankodik, hogy mit is tegyen. Az órával játszott sakk szemidinamikus. A keresztrejtvények statikusak.
- **Diszkrét (discrete) vagy folytonos (continuous).** A diszkrét/folytonos felosztás alkalmazható a környezet *állapotára*, az *időkezelés módjára*, az ágens *észleléseire*, valamint *cselekvéseire*. Például egy diszkrét állapotú környezet, mint amilyen a sakkjáték, véges számú különálló állapottal rendelkezik. A sakkban szintén diszkrét

<sup>5</sup> A „sorozatszerű” (sequential) szót a számítástudományban a „párhuzamos” (parallel) ellentéteként is használják. A két jelentés alapvetően független egymástól.

az akciók és cselekvések halmaza. A taxivezetés folytonos állapotú és idejű probléma: a sebesség, a taxi és más járművek helye folytonos értékek egy tartományát járja végig a folytonos időben. A taxivezetés akciói szintén folytonosak (például kanyarodási szögek stb.). Szigorúan véve a digitális kameráktól érkező bemenet diszkrét, de tipikusan úgy kezeljük, mint amely folyamatosan változó mennyiségeket és helyeket reprezentál.

- **Egyágenses (single agent) vagy többágenses (multiagent).** Az egyágenses és többágenses környezetek közötti különbségtétel egyszerűnek tűnhet. Például a keresztrejtvényt megfejtő ágens önmagában nyilvánvalóan egyágenses környezetben van, míg egy sakkozó ágens egy kétágensesben. Vannak azonban kényes kérdések. Először is: leírtuk azt, hogy egy entitás hogyan tekinthető ágensnek, ugyanakkor nem magyaráztuk meg, mely entitások tekintendők ágensnek. Egy A ágensnek (például a taxisoförnek) egy B objektumot (egy másik járművet) ágensnek kell tekintenie, vagy egyszerűen egy sztochasztikusan viselkedő dolognak, a tengerparti hullámokhoz vagy a szélben szálló falevelekhez hasonlatosan? A választás kulcsa az, hogy vajon B viselkedése legjobban egy A viselkedésétől függő teljesítménymérték maximalizálásával írható-e le. Például a sakkban a B ellenfél saját teljesítménymértékét próbálja maximalizálni, amely – a sakk szabályainak következtében – A teljesítménymértékét minimalizálja. Így a sakk egy **versengő (competitive)** többágenses környezet. Másrészről, a taxi vezetési környezetben az ütközések elkerülése az összes ágens teljesítménymértékét maximálja, így az részben **kooperatív (cooperative)** többágenses környezet. Emellett részben versengő is, hiszen például csak egy autó tud egy parkolóhelyet elfoglalni. A többágenses környezetekben felmerülő ágenstervezési problémák gyakran egészen mások, mint egyágenses környezetekben. Többágenses környezetekben például a **kommunikáció (communication)** gyakran racionális viselkedésként bukkan fel; egyes részlegesen megfigyelhető versengő környezetekben a **sztochasztikus viselkedés racionális**, hiszen így elkerülhetők a megjósolhatóság csapdái.

Ahogy várható, a legnehezebb a részlegesen megfigyelhető, sztochasztikus, sorozatszerű, dinamikus, folytonos és többágenses eset. Az is kiderül, hogy a valós helyzetek legtöbbje olyan bonyolult, hogy valódi determinisztikusságuk vitatott kérdés; gyakorlati okokból sztochasztikusként kezelendők. A taxivezetés minden szempontok szerint nehéz.

A 2.6. ábra ismerős környezetek tulajdonságait sorolja fel. Vegyük észre, hogy a válaszok nem mindenkor egyértelműek. Például a sakkot teljesen megfigyelhetőnek tüntettük fel; szigorúan véve ez hibás, mivel a rosálásra, menet közbeni ütéstre és ismétléses döntetlenre vonatkozó egyes szabályok megkövetelik a játék menetével kapcsolatos adatok meggjegyzését, amelyek a táblázat állapotának részeként nem figyelhetők meg. A megfigyelhetőség ezen kivételei persze csekélyek egy taxivezető, egy angol nyelvtanár vagy egy orvosi diagnosztikai rendszerhez képest.

A táblázatban bizonyos további válaszok a feladatkörnyezet definiálásától függnek. Az orvosi diagnosztikai feladatot egyágensesnek tüntettük fel, mivel a betegben zajló betegségi folyamat eredményesen nem modellezhető ágensként, de egy orvosi diagnosztikai rendszernek szükség esetén törődnie kell ellenszegűl betegekkel és szkeptikus munkatársakkal, így a környezetnek lehet többágenses aspektusa. Továbbá az orvosi diagnosztika epizódszerű, ha valaki a feladatot úgy tekinti, mint a tünetek listája

Környezet	Megfigyelhető	Determinisztikus	Epizódszerű	Statikus	Diszkrét	Ágensek
Keresztrejtvény Sakkjátszma időmérésessel	Teljesen	Determinisztikus	Sorozat	Statikus	Diszkrét	Egy
	Teljesen	Stratégiai	Sorozat	Szemidinamikus	Diszkrét	Több
Póker Ostábla	Részben	Sztochasztikus	Sorozat	Statikus	Diszkrét	Több
	Teljesen	Sztochasztikus	Sorozat	Statikus	Diszkrét	Több
Taxivezetés Orvosi diagnosztika	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Több
	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Egy
Képfeldolgozó rendszer Alkatrészfelvezvő robot	Teljesen	Determinisztikus	Epizód	Szemidinamikus	Folytonos	Egy
	Részben	Sztochasztikus	Epizód	Dinamikus	Folytonos	Egy
Olajfinomító-vezérlő Interaktív angol tanár	Részben	Sztochasztikus	Sorozat	Dinamikus	Folytonos	Egy
	Részben	Sztochasztikus	Sorozat	Dinamikus	Diszkrét	Több

2.6. ábra. Példák feladatkörnyezetekre és jellemzőik

alapján történő diagnózis kiválasztását; a probléma sorozatszerű, ha a feladat tartalmazhat egy vizsgálatsorozat-javaslatot, az eredmények kiértékelését a kezelés folyamán és így tovább. Továbbá sok környezet epizódszerű az ágensek egyéni akciójánál magasabb szinteken. Például egy sakktorna játszmák sorozatát tartalmazza, minden játék egy epizód, mivel (nagyjából) az ágens adott játszmabeli lépéseihez hozzájárulása az ágens teljesítményéhez nem függ a korábbi játszmákban választott lépéseitől. Másrészről, az egy játszmán belüli döntéshozatal biztosan sorozatszerű.

A könyvhöz tartozó példaprogramtár ([aima.cs.berkeley.edu](http://aima.cs.berkeley.edu)) tartalmaz számos környezetimplementációt, egy általános célú környezetszimulátorral együtt, amely egy vagy több ágenst helyez el egy szimulált környezetben, megfigyeli a viselkedéstük az időben, és kiértékelőket egy adott teljesítménymérték szerint. Ilyen kísérleteket gyakran nem egy, hanem több, egy **környezetosztályból** (**environment class**) származó környezetre hajtanak végre. Például egy taxiszofor szimulált forgalomban történő értékelésére több szimulációt futtatnánk különböző forgalommal, világítással, és időjárási viszonyokkal. Ha az ágenst egyedi esetre terveztük, kihasználhatjuk az adott környezet speciális tulajdonságait, de nem találhatjuk meg a vezetés általában jó módját. Ezért a programtár tartalmaz egy **környezetgenerátort** (**environment generator**) minden környezetosztályra, amelyik kiválaszt bizonyos környezeteket (bizonyos valószínűséggel) az ágens futtatására. Például a porszívó környezetgenerátor véletlenszerűen állítja be a piszok mintázatát és az ágens helyét. Ezek után az ágens átlagos teljesítményére vagyunk kíváncsiak az adott környezetosztályra nézve. Egy racionális ágens az adott környezetosztályra maximalizálja ezt az átlagos teljesítményt. A 2.7. és 2.12. közötti feladatok végigvisznek egy környezetosztály kialakításán és abban többféle ágens kiértékelésén.

## 2.4. AZ INTELLIGENS ÁGENSEK STRUKTÚRÁJA

Az ágenseket eddig *viselkedésük* leírásán keresztül vizsgáltuk – azon cselekvés alapján, amelyet egy adott észlelési sorozat hatására végrehajtanak. Most neki kell esnünk a kemény diónak: hogyan működnek belül. A mesterséges intelligencia feladata az **ágensprogram** (*agent program*) megtervezése: egy függvényé, amely megvalósítja az észlelések és a cselekvések közötti leképezést. Feltételezzük, hogy ez a program valamiféle fizikai érzékelőkkel és beavatkozókkal ellátott számítóeszközön fog futni – ezt architektúrának (*architecture*) nevezzük:

$$\text{ágens} = \text{architektúra} + \text{program}$$

Nyilvánvalóan a kiválasztott programunknak olyannak kell lennie, ami megfelelő az architektúra számára. Ha a program például Sérülés jellegű akciókat fog javasolni, akkor jobb, ha az architektúrának vannak lábai. Az architektúra lehet egyszerű számítógép, vagy lehet egy robotautó számos fedélzeti számítógéppel, kamerával és más érzékelőkkel. Általánosságban az architektúra a szensoruktól érkező észleléseket elérhetővé teszi a program számára, futtatja a programot, és cselekvéseit létrejöttük pillanatában a beavatkozók felé továbbítja. E könyv túlnyomó része az ágensprogramok tervezéséről szól, bár a 24. és 25. fejezet kifejezetten az érzékelőkkel és a beavatkozókkal foglalkozik.

### Ágensprogramok

A könyvben tervezett ágensprogramok mindenkorán azonos vázzal rendelkeznek: bemenetként fogadják az aktuális észleléseket a szensoruktól, és visszaküldenek egy cselekvést a beavatkozókhoz.<sup>6</sup> Vegyük észre a különbözetet az ágensprogram, amely az aktuális észlelést veszi bemenetként, és az ágensfüggvény között, amely a teljes észlelési történetet fogadja. Az ágensprogram azért fogadja csak az aktuális észlelést bemenetként, mert csak ez érkezik a környezettől – ha az ágens cselekedetei a teljes észlelési sorozattól függnek, az ágensnek emlékeznie kell az észlelésekre.

Az ágensprogramot egy egyszerű pszeudokódú nyelv segítségével fogjuk leírni, amit a B) függelék definiál. (Az interneten elérhető kódtárvállóságos programozási nyelveken írt megvalósításokat tartalmaz.) Például a 2.7. ábra egy egészen egyszerű ágenst mutat, amely nyomon követi az észlelési sorozatot, és felhasználja egy akciót tartalmazó táblázat megcímzésére annak eldöntésére, hogy mit csináljon. A táblázat reprezentálja az ágensfüggvényt, amit az ágensprogram testesít meg. Ahhoz, hogy ily módon egy racionális ágenst építsünk, nekünk, mint tervezőknek, olyan táblázatot kell megalakítunk, amely minden lehetséges észlelési sorozathoz a megfelelő cselekvést tartalmazza.

Példaértékű annak megfontolása, hogy az ágensek tervezésének táblázatvezérelt megközelítése miért van kudarcra ítélezve. Legyen *P* a lehetséges észlelések halmaza és

<sup>6</sup> Vannak más lehetőségek is az ágensvállás számára, például leírhatjuk az ágensprogramot korutinok (*coroutines*) formájában, amelyek a környezetbe képest aszinkron módon futnak. minden ilyen korutin rendelkezik egy bemeneti és egy kimeneti porttal, és egy olyan ciklust tartalmaz, amelyik beolvassa a bemeneti portról az észleléseket, és kiírja a cselekvéseket a kimeneti portra.

```

function TÁBLÁZAT-VEZÉRLÉSÜ-ÁGENS(észlelés) returns egy cselekvés
  static: észlelések, egy sorozat, kezdetben üres
  táblázat, egy táblázat, az észlelési sorozat indexeli, kezdetben teljesen feltöltött
    csatold az észlelést az észlelések végére
    cselekvés  $\leftarrow$  KIKERESÉS(észlelések, táblázat)
    return cselekvés

```

**2.7. ábra.** A TÁBLÁZAT-VEZÉRLÉSÜ-ÁGENS program minden egyes új észlelésre meghívódik, és minden alkalommal visszaad egy cselekvést. Saját adatstruktúrájában követi nyomon az észlelési sorozatot.

T az ágens élettartama (az általa vett észlelések teljes száma). A táblázat  $\sum_{t=1}^T |P_t|$  elemet fog tartalmazni. Vegyük példaként az automata taxit: az egyetlen kamerától érkező vizuális bemenet körülbelül 27 megabajt/másodperc sebességgel érkezik (30 kocka másodpercenként,  $640 \times 480$  képpont 24 bit színinformációval). Ez alapján egyórányi vezetéshez olyan táblázatot kapunk, amely több mint  $10^{250,000,000,000}$  bejegyzést tartalmaz. Még a sakkhoz tartozó táblázat is – amely a való világ egy kicsiny, jól viselkedő részlete – legalább  $10^{150}$  bejegyzést tartalmazna. Ezen táblázatok ijesztő mérete (a megfigyelhető világégyetemben az atomok száma kevesebb mint  $10^{80}$ ) azt jelenti, hogy (a) ebben az univerzumban egyetlen fizikai ágensnek sem lesz elég helye a táblázat tárolására, (b) a tervezőnek nem lenne elég ideje a táblázat elkészítéséhez, (c) egyetlen ágens sem lenne képes a táblázat helyes bejegyzéseit megtanulni saját tapasztalatából, és (d) még ha a környezet elégéggé egyszerű is egy megvalósítható méretű táblázathoz, a tervezőnek még akkor sincs segítsége ahhoz, hogyan töltse ki a táblázat bejegyzéseit.

Mindezek ellenére, a TÁBLÁZAT-VEZÉRLÉSÜ-ÁGENS *megteszi* azt, amit akarunk: megvalósítja a kívánt ágensfüggvényt. Az MI alapvető kihívása, hogy hogyan írunk olyan programot, amely – a lehetőségek határain belül – nagyszámú táblázatbejegyzések helyett kisméretű programkóddal produkál racionális viselkedést. Sok példánk van arra, hogy ez más területeken megvalósítható: például az 1970-es évek előtt a mérnökök és az iskolás gyerekek által használt hatalmas négyzetgyöktáblázatokat ma már egy eletronikus kalkulátorokon futó ötsoros program váltotta fel, amely a Newton-módszert alkalmazza. A kérdés az, hogy megteheti-e az MI azt az általános intelligens viselkedésre, amit Newton tett meg a négyzetgyökökre? Abban hiszünk, hogy a válasz igen.

E fejezet hátralevő részében négy alapvető ágensprogramtípust vázolunk, amely megtestesíti a szinte minden intelligens rendszer mögött meghúzódó alapelveket:

- Egyszerű reflexszerű ágensek.
- Modellalapú reflexszerű ágensek.
- Célorientált ágensek.
- Hasznosságorientált ágensek.

Ezek után általánosságban elmagyarázzuk, hogyan alakíthatjuk át ezeket *tanuló ágensekké*.

## Egyszerű reflexszerű ágensek

A legegyszerűbb fajtájú ágens az **egyszerű reflexszerű ágens** (**simple reflex agent**). Ezek az ágensek az *aktuális* észlelés alapján választják ki a cselekvéseket, figyelmen kívül hagyva az észlelési történet többi részét. Például a porszívó ágens, amelynek ágensfüggvényét a 2.3. ábra táblázatos formában mutatja, egy egyszerű reflexszerű ágens, mivel döntései csak a jelenlegi helyszínen és azon alapulnak, hogy ott van-e piszok. A 2.8. ábra mutatja ezen ágens programját.

Vegyük észre, hogy az ágensprogram igen kicsi a megfelelő táblázathoz képest. A legnyilvánvalóbb méretcsökkenés az észlelési történet figyelmen kívül hagyásából következik, amely a lehetőségek számát  $4^T$ -ről 4-re csökkenti. Egy további, kisebb csökkenés abból a tényből fakad, hogy amikor az aktuális négyzet piszkos, az akció nem függ a helyszíntől.

```
function REFLEXSZEGRÜ-PORSZIVÓ-ÁGENS(helyszín, állapot) returns egy cselekvés
```

```
if állapot = Piszkos then return Felszívás
else if helyszín = A then return Jobbra
else if helyszín = B then return Balra
```

**2.8. ábra.** Egy egyszerű reflexszerű ágens programja a kétállapotú porszívókörnyezetben. A 2.3. ábrán megadott ágensfüggvényt valósítja meg.

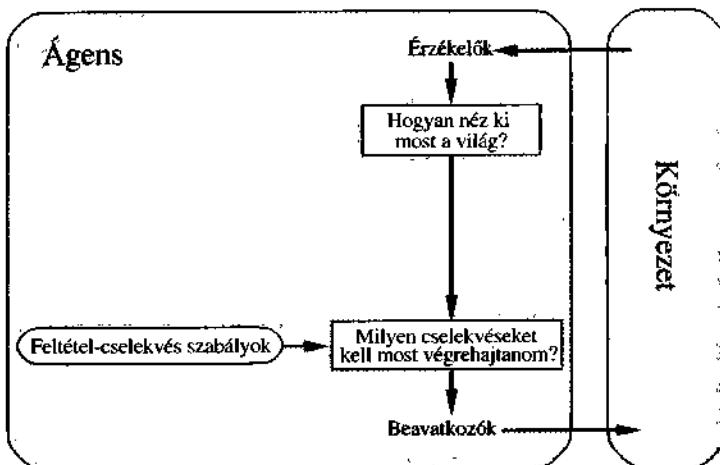
Képzeljük magunkat az automata taxisofőr helyébe! Ha az előttünk haladó autó félez, és a féklámpái kigyulladnak, akkor észre kell vennünk, és el kell kezdenünk félezni. Más szavakkal, valamilyen feldolgozás zajlik a képi bemeneten, hogy megállapítsunk egy feltételt, amit az „előző autó félez”-nek hívunk; azután ez kivált valamelyen kialakított kapcsolatot az ágens programjában a „kezdj félezni” cselekvéshez. Ezt a kapcsolatot **feltétel–cselekvés szabálynak** (**condition–action rule**)<sup>7</sup> hívjuk, és

**ha az-előző-autó-félez akkor kezdj-félezni**

formában írjuk. Az embereknek sok hasonló kapcsolatuk van, egy részük tanult válasz (ami a vezetést illeti), míg más részük feltétlen reflex (mint például a pislogás, amikor valami megközelíti a szemünket). A későbbiekben több különböző módot is látni fogunk arra, hogy ilyen kapcsolatok hogyan tanulhatók és valósíthatók meg.

A 2.8. ábra programja egy egyedi porszívókörnyezetre specifikus. Egy általánosabb és rugalmasabb megközelítés az, hogy először egy általános célú értelmezőt építünk feltétel–cselekvés szabályokra, majd szabályhalmazokat hozunk létre specifikus feladatkörnyezetek számára. A 2.9. ábra ezen általános program struktúráját adja meg sematikus formában, meghatározva, hogy a feltétel–cselekvés szabályok hogyan teszik lehetővé az ágens számára az észlelések és cselekvések összekötését. (Ne aggódjon, ha ez túl egyszerűnek tűnik: hamarosan sokkal érdekesebbé válik.) Az ágens döntési folyamatának aktuális belső állapotát négyzetekkel jelöljük, a folyamatban felhasznált háttérítudás reprezentálására pedig ováli-

<sup>7</sup> A kapcsolatot **situáció–cselekvés** (**situation–action rule**), **produkciós** (**production rule**) vagy **ha–akkor szabálynak** (**if–then rule**) is nevezik.



2.9. ábra. Egy egyszerű reflexszerű ágens sematikus diagramja

sokat használunk. Az ágensprogram, amely szintén igen egyszerű, a 2.10. ábrán látható. A BEMENET-FELDOLGOZÁS függvény a bemenetből állítja elő az aktuális állapot absztrakt leírását, a SZABÁLY-ILLESZTÉS függvény pedig az első olyan szabályt adja vissza a szabályok halmazából, amely illeszkedik az adott állapot leírására. Vegyük észre, hogy a „szabályok” és az „illeszkedés” fogalmakkal történő leírás tisztán koncepcionális – egyedi megvalósítások akár egy logikai áramkört megvalósító logikai kapu halmazból is állhatnak.

**➔** Az egyszerű reflexszerű ágenseknek megvan az az értékelendő tulajdonsága, hogy egyszerűek, ugyanakkor igen korlátozott intelligenciájúnak bizonyulnak. A 2.10. ábrán látható ágens *csak akkor* fog működni, ha a helyes döntés kizárolag az aktuális észlelés alapján meghozható – azaz akkor, ha a környezet teljesen megfigyelhető. Már a megfigyelhetőség legkisebb hiánya is komoly problémát okozhat. Például a korábban bemutatott fékezési szabály feltételezi, hogy az előző-autó-félező feltétel megállapítható az aktuális észlelésből – az aktuális videoképből – amennyiben az előző autónak középre szerelt féklámpája van. Sajnálatos módon, a régebbi modellekben különbözőképpen helyezkednek el a hátsó helyzetjelzők, a féklámpák és az indexlámpák, így egyetlen kép alapján nem mindig lehetséges azt megállapítani, hogy az autó félez-e. Egy egyszerű reflexszerű ágens, amely egy ilyen autó mögött halad, vagy folyamatosan szükségtelenül félezne, vagy ami még rosszabb, sosem félezne.

```

function EGYSZERÜ-REFLEXSZERÜ-ÁGENS(észlelés) returns egy cselekvés
    static: szabályok, feltétel-cselekvés szabályok halmaza

    állapot ← BEMENET-FELDOLGOZÁS(észlelés)
    szabály ← SZABÁLY-ILLESZTÉS(állapot, szabályok)
    cselekvés ← SZABÁLY-CSELEKVÉS[szabály]
    return cselekvés

```

2.10. ábra. Egy egyszerű reflexszerű ágens. A jelenlegi szituációra (amit az észlelés határoz meg) illeszkedő feltételű szabály megkeresésével, majd a hozzá tartozó cselekvés végrehajtásával működik.

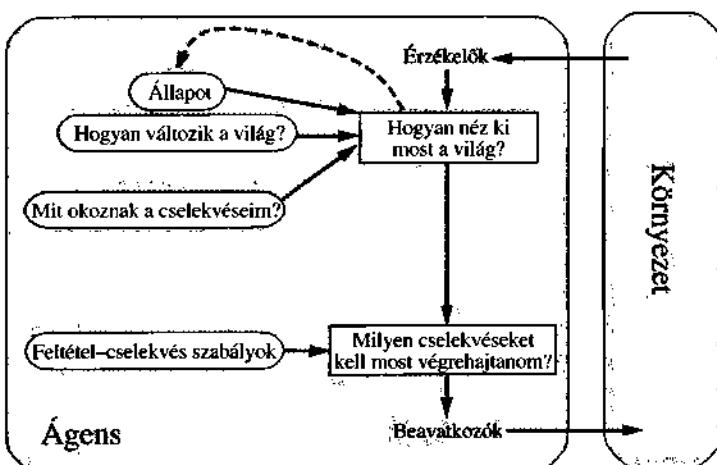
Hasonló problémát láthatunk felbukkanni a porszívóvilágban. Tegyük fel, hogy egy egyszerű reflexszerű ágenst megfosztunk a helyzetérzékelő szenzoráról, és csak piszokérzékelője van. Egy ilyen ágensnek csak két észlelése van: [Koszos] és [Tiszta]. Felszívhat válaszul a [Koszos]-ra; de mit tegyen, ha [Tiszta] az észlelés? A Balra mozgás (mindörökre) sikertelen lesz, ha például az A négyzetben kezd, a Jobbra mozgás pedig (mindörökre) sikertelen, ha például a B négyzetben kezd. A végtelen ciklusok gyakran elkerülhetetlenek részlegesen megfigyelhető környezetben működő egyszerű reflexszerű ágensek számára.

Elképzelhető menekvés a végtelen ciklusokból, ha az ágens **randomizálhatja (randomize)** cselekvéseit. Például ha a porszívó ágens [Tiszta]-t észlel, feldobhat egy érmét a Bal és Jobb közötti választáshoz. Könnyű megmutatni, hogy az ágens el fogja érni a másik négyzetet átlagosan két lépésen belül. Ezután, ha a négyzet piszkos, feltakarítja, és a tisztítási feladat befejeződik. Ily módon egy randomizált egyszerű reflexszerű ágens túlteljesíthet egy determinisztikus egyszerű reflexszerű ágenst.

A 2.3. alfejezetben említettük, hogy a helyes formájú randomizált viselkedés lehet racionális bizonyos többágenses környezetekben. Egyágenses környezetekben a randomizáció általában *nem* racionális. Egy hasznos trükk, amely segítheti az egyszerű reflexszerű ágenst bizonyos szituációkban, de a legtöbb esetben ennél sokkal jobbat éhetünk el kifinomultabb determinisztikus ágensekkel.

## Modellalapú reflexszerű ágensek

A részleges megfigyelhetőség kezelésének leghatékonyabb módja, ha az ágens *nyomon követi a világ jelenleg nem látható részét*. Azaz, az ágensnek nyilván kell tartania valamiféle **belső állapotot (internal state)**, amely az észlelési történeten alapul, és így a jelenlegi állapot nem megfigyelt aspektusainak legalább egy részét tükrözi. A fékezési probléma esetében a belső állapot nem túlságosan terjedelmes – csak a kamerától származó előző képkocka, amely lehetővé teszi az ágens számára annak megállapítását,



2.11. ábra. Belső állapottal rendelkező reflexszerű ágens

```

function REFLEXSZERŰ-ÁGENS-ÁLLAPOT(észlelés) returns egy cselekvés
  static: állapot, a világ jelenlegi állapotának leírása
    szabályok, feltétel-cselekvés szabályok halmaza
    cselekvés, a legutolsó cselekvés, kezdetben semmi

  állapot ← ÁLLAPOT-FRISÍTÉS(állapot, cselekvés, észlelés)
  szabály ← SZABÁLY-ILLESZTÉS(állapot, szabályok)
  cselekvés ← SZABÁLY-CSELEKVÉS[szabály]
  return cselekvés

```

**2.12. ábra.** A modellalapú reflexszerű ágens. Nyomon követi a világ jelenlegi állapotát egy belső modellben. Ezek után a reflexszerű ágenshez hasonlóan választ egy cselekvést.

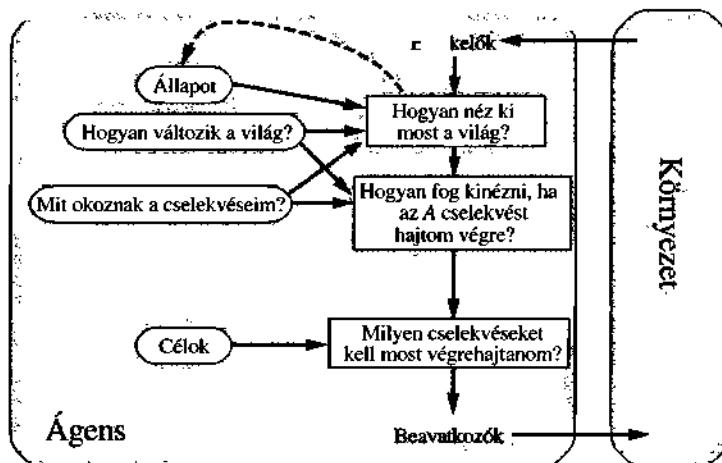
hogy a jármű két szélén levő két piros lámpa egyszerre gyullad és alszik-e ki. Más vezetési feladatokhoz, mint például a sávváltáshoz, az ágensnek nyomon kell követnie hol tartózkodik a többi autó, amennyiben nem látja mindegyiket egyszerre.

Ezen belső állapot frissítése az idő előrehaladtával kétféle tudás beépítését igényli az ágensprogramba. Először is szükségünk van olyan információra, hogy a világ hogyan alakul az ágenstől függetlenül – például egy előző autó általában közelebb lesz most, mint egy pillanattal azelőtt. Másodsor, szükségünk van valamilyen információra arról, hogy az ágens saját cselekvései hogyan változtatják meg a világot – például amikor az ágens a kormányt az óramutató járásának megfelelően tekeri, az autó jobbra fordul, vagy öt percig északnak tartó vezetés után egy autópályán általában körülbelül öt mérfölddel lesz északabbra, mint ahol öt perce volt. Ezt, a „világ működésének mikéntjéről” szóló tudást – akár egyszerű logikai áramkörökkel, akár teljes tudományos elméletekkel valósítjuk meg – a világ **modelljének** (**model**) hívjuk. Egy ilyen modellt használó ágenst pedig **modellalapú ágensnek** (**model-based agent**).

A 2.11. ábra szemlélteti a belső állapottal rendelkező reflexszerű ágens struktúráját, megmutatva azt, ahogyan a jelenlegi észlelés a régi belső állappal kombinálódva létrehozza a jelenlegi állapot frissített leírását. Az ágensprogram a 2.12. ábrán látható. Az érdekes rész az ÁLLAPOT-FRISÍTÉS eljárás, amely a belső állapot leírásának létrehozásáért felelős. Az új észleléseknek az állapotról szóló tudás tükrében történő értelmezése mellett az ágensprogram nyomon követi a világ láthatatlan részeit a világ fejlődését leíró információ segítségével, az ágensprogramnak pedig azt is tudnia kell, hogy az ágens cselekvései milyen hatással lesznek a világ állapotára. Részletesebb példák a 10. és 17. fejezetben találhatók.

## Célorientált ágensek

A környezet jelenlegi állapotának ismerete nem minden elég annak eldöntéséhez, hogy mit tegyünk. Például egy kereszteződésben a taxi mehet egyenesen, de fordulhat jobbra vagy balra is. A helyes döntés attól függ, hogy a taxi hova szeretne eljutni. Más szavakkal, a jelenlegi állapot leírása mellett az ágensnek valamiféle **cél- (goal)** információval is rendelkeznie kell, amely leírja a kívánatos helyzeteket – például azt, hogy az utas jusson el a célállomásig. Az ágensprogram ezt összevetheti a lehetséges cselekvések eredményeiről szóló információkkal (ugyanazokkal, amelyeket a reflexszerű



**2.13. ábra.** Egy modellalapú, célorientált ágens. Nyomon követi a világ állapotát és az előrendő célok halmazát is, és kiválaszt egy cselekvést, amely (végső soron) céljainak eléréséhez vezet.

ágens esetében a belső állapot frissítésekor használt) annak érdekében, hogy a céljához vezető cselekvést meghatározza. A 2.13. ábra bemutatja a célorientált ágens felépítését.

Néha a célorientált cselekvés választás egyszerű, amikor a cél teljesülése azonnal egyetlen cselekvéssel elérhető. Máskor sokkal trükkösebb, amikor az ágensnek befordulások és visszakanyarodások sorozatát kell megfontolnia a cél eléréséhez. A keresés (search) (lásd 3–6. fejezet) és a tervkészítés (planning) (lásd 11–12. fejezet) a mesterséges intelligencia azon területei, amelyek az ágens céljait elérő cselekvéssorozat megtervezésével foglalkoznak.

Vegyük észre, hogy ez a fajta döntéshozatal alapjaiban különbözik az előzőben ismertetett feltétel-cselekvés szabályuktól, ezek magukban foglalják a jövő figyelembevételét – mind a „Mi fog történni, ha ezt és ezt teszem?”, mind a „Boldoggá tesz ez majd engem?” kérdéseket. Reflexszerű ágensek tervezésében ezeket explicit módon nem használjuk, mivel a tervező különböző esetekre előre meghatározta a helyes cselekvést. A reflexszerű ágens félez, ha féklámpát lát kigyulladni. Célorientált elméletben gondolkodhat úgy, hogyha az előző autó féklámpái égnek, akkor az le fog lassulni. Az alapján, ahogy a világ általában működik, az egyetlen cselekvés, amely eléri azt a célt, hogy ne koccsanjon neki más autóknak, a félezés.

Bár a célorientált ágens kevésbé tűnik hatékonynak, sokkal rugalmasabb, mivel a döntéseit alátámasztó tudás explicit módon megjelenik, és így módosítható. Ha elkezdni az eső, frissítheti a tudását arról, hogy a fékek mennyire lesznek hatékonyak – ez automatikusan az összes releváns viselkedés olyan megváltozását eredményezi, hogy azok az új körülményhez igazodjanak. Reflexszerű ágensek esetében ugyanakkor, a feltétel-cselekvés szabályok nagy számát kellene átfirunk. Természetesen a célorientált ágens különböző úti célok elérésében is rugalmasabb. Egyszerűen egy új cél megadásával a célorientált ágenstől egy új viselkedés megijelenését kaphatjuk. A reflexszerű ágens „mikor fordulj” és „mikor menj egyenesen” szabályai csak egy cél esetében fognak működni: ki kell őket cserélni ahhoz, hogy egy új helyre menjen.

## Hasznosságorientált ágensek

A környezetek többségében a célok önmagukban nem elegendők jó minőségű viselkedés létrehozásához. Például többféle cselekvéssorozat vezet a taxi végállomásának – és így a céljának – eléréséhez, de némelyik gyorsabb, biztonságosabb, megbízhatóbb vagy olcsóbb, mint mások. A célok csak durva különbséget tesznek „boldog” és „boldogtalan” állapotok között, egy általánosabb teljesítménymérték ezzel szemben a világ állapotainak (vagy állapotSOROZATAinak) megfelelő összehasonlítását tehetné lehetővé, azt megadva, hogy pontosan mennyire tennék boldoggá az ágenst, ha elérné azokat. Mivel a „boldogság” nem hangzik túl tudományosan, ezért a szokásos terminológia azt mondja, hogy a világ egyik állapota előnyösebb egy másikhoz képest, ha nagyobb a hasznossága (utility)<sup>8</sup> az ágens számára.

A hasznosságfüggvény (utility function) egy állapotot (vagy állapotok egy sorozatát) egy olyan valós számra képezi le, amelyik a hozzá rendelt boldogság fokát írja le. A hasznosságfüggvény teljes meghatározása kétféle olyan helyzetben tesz lehetővé racionális döntéseket, amikor a célok erre alkalmatlanok. Egyrészt, amikor egymásnak ellentmondó célok vannak, amelyeknek csak egy része érhető el (mint például a biztonságosság és a sebesség), akkor a hasznosságfüggvény meghatározza a helyes kompromisszumot. Másrészt, amikor több cél van, amelyek elérésére az ágens törekedhet, és egyikük sem érhető el teljes bizonyossággal, a hasznosság olyan módszert ad, amivel a sikeres valószínűsége a célok fontosságához mérhető.

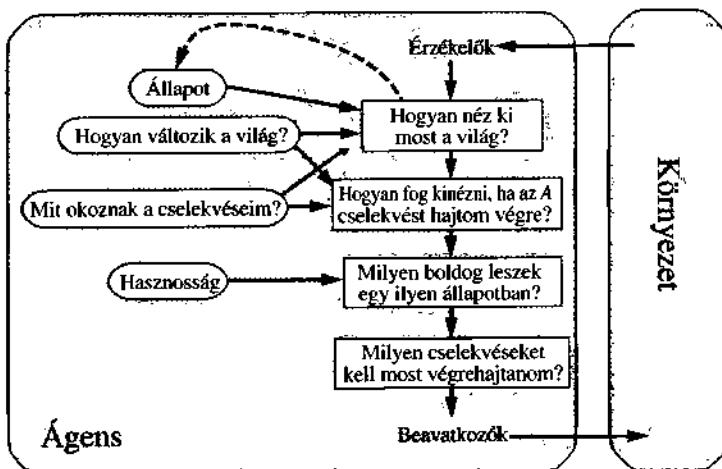
A 16. fejezetben megmutatjuk, hogy minden racionális ágensnek úgy kell működnie, mintha rendelkezne hasznosságfüggvénnyel, amelynek várható értékét maximalizálni akarja. Egy ágens, amelynek van egy *explicit* hasznosságfüggvénye, ily módon racionális döntéseket hozhat, és ezt egy általános algoritmussal teheti meg, amely nem függ a maximalizálandó specifikus hasznosságfüggvénytől. Ezáltal a racionálitás „globális” definíciója – amely azon ágensfüggvényeket tekinti racionálisnak, amelyek a legnagyobb teljesítményt hozzák – átváltozik a racionáliságens-tervezés egy „helyi” kényszerévé, amely kifejezhető egy egyszerű programban.

A hasznossággalapú ágens struktúrája a 2.14. ábrán látható. Valódi hasznossággalapú ágens programok az V. részben szerepelnek, ahol olyan döntéshozó ágenseket tervezünk, amelyeknek a részlegesen megfigyelhető környezetekben elkerülhetetlen bizonytalanságot is kezelniük kell.

## Tanuló ágensek

Leírtunk többféle módszert arra, hogy ágensprogramok hogyan választják ki cselekvéseiket. Nem magyaráztuk meg azonban eddig, hogy *hogyan jön létre* az ágensprogram. Híres korai cikkében Turing (Turing, 1950) azzal az ötlettel foglalkozik, hogy intelligens gépeit ténylegesen kézzel programozza. Megbecsüli, mennyi munkát vehet ez igénybe, és megállapítja, hogy „valamilyen eredményesebb módszer kívánatosnak látszik”. Az általa javasolt módszer az, hogy építünk tanuló gépeket, majd tanítsuk

<sup>8</sup> A „hasznosság” szó itt a „hasznosság minősége” kifejezésre utal, nem az áram- vagy vízszolgáltatóra. (Az angol utility szó többes jelentésű, a szerző ezért magyarázza meg helyes értelmezését a lábjegyzetben. –A ford.)



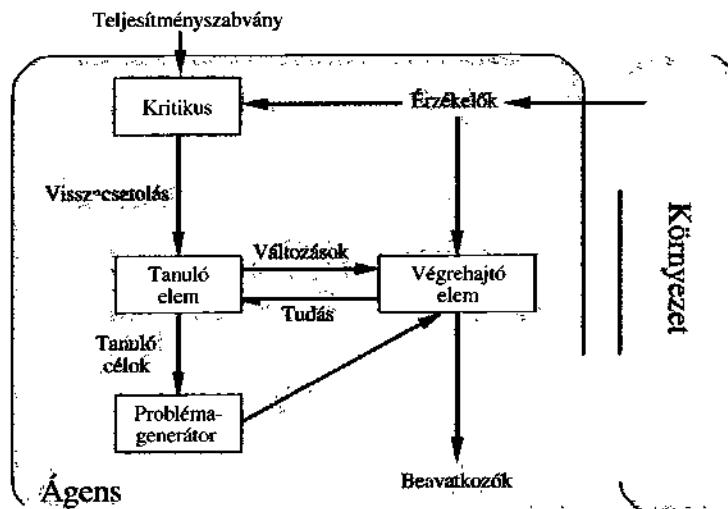
**2.14. ábra.** Egy modellalapú, hasznossággalapú ágens. A világ modellje mellett egy hasznosságfüggvényt is alkalmaz, amely a világ állapotaihoz rendelt preferenciáit méri. Ezek után olyan cselekvést választ, amely a legjobb várható hasznossághoz vezet, amit az összes lehetséges végállapot előfordulási valószínűségevel súlyozott átlagolásával számít ki.

ezeket. Az MI sok területén ez ma a javasolt módszer korszerű rendszerek építésére. A tanulásnak van egy másik előnye is, ahogy megállapítottuk korábban: lehetővé teszi az ágens számára, hogy kezdetben ismeretlen környezetben működjön, és kompetenssébbé váljon, mint ahogy azt kezdeti tudása lehetővé tette volna. Ebben a fejezetben röviden bemutatjuk a tanuló ágensek főbb elveit. A könyv majdnem minden fejezetében rámutatunk a tanulás lehetőségeire és módszereire az egyes ágensfajtákban. A VI. rész merül el mélyebben a különféle tanulási algoritmusokban.

Egy tanuló ágens négy koncepcionális komponensre bontható fel, ahogy azt a 2.15. ábra mutatja. A legfontosabb különbösségtétel a javításokért felelős **tanuló elem** (*learning element*) és a külső cselekvések kiválasztásáért felelős **végrehajtó elem** (*performance element*) között van. A végrehajtó elem az, amit eddig a teljes ágensnek tekintettünk: ez végzi az észleléseket, és ez dönt a cselekvésekről. A tanuló elem a **kritikustól** (*critic*) kapott, az ágens működéséről szóló visszajelzést használja annak megállapítására, hogy a végrehajtó elemet hogyan kell módosítani annak érdekében, hogy a jövőben jobban működjön az ágens.

A tanuló elem tervezése nagyrészt a végrehajtó elem tervezésétől függ. Amikor egy olyan ágenst próbálunk tervezni, amely megtanul egy bizonyos képességet, az első kérdés nem a „hogyan fogom én ezt megtanulni?”, hanem a „milyen teljesítményelemre lesz az ágensemnek szüksége ennek végrehajtásához, ha már megtanulta, hogyan kell?”. Ha adott egy meghozzájárult ágens, tanuló eljárások az ágens minden részének javításához készíthetők.

A kritikus megmondja a tanuló elemnek, hogy az ágens milyen jól működik egy rögzített teljesítményszabványhoz viszonyítva. A kritikus szükséges, hiszen az észlelések önmagukban nem jelzik az ágens sikerességét. Például egy sakk program fogadhat egy olyan észlelést, hogy mattot adott az ellenfelének, de szüksége van egy teljesítményszabványra, hogy tudja, ez egy jó dolog. Az észlelés önmagában nem mondja meg ezt. Fontos, hogy a teljesítményszabvány rögzített legyen. Koncepcionálisan tekinthető



2.15. ábra. Tanuló ágens egy általános modellje

úgy, mintha az ágensen teljesen kívül lenne, mivel az ágens nem módosíthatja azt saját viselkedésének megjavítása érdekében.

A tanuló ágens utolsó komponense a **problémagenerátor (problem generator)**. Ennek feladata, hogy olyan cselekvéseket javasoljon, amelyek új és informatív tapasztalatokhoz vezetnek. Az a helyzet, hogy ha a végrehajtó elem a saját feje után menne, akkor adott tudása alapján mindenleges a legjobb cselekvéseket választaná. De ha az ágens hajlandó egy kis felfedezésre, és rövid távon talán néhány szuboptimális cselekvésre, hosszú távon sokkal jobb cselekvéseket fedezhet fel. A problémagenerátor feladata, hogy ilyen felfedező cselekvéseket javasoljon. Ezt teszik a tudósok is, amikor kísérleteket végeznek. Galilei nem gondolta, hogy a pisai torony tetejéből köveket leejteni önmagában értékes dolog. Sem a köveket nem próbálta összetörni, sem a szerencsétlen arra járók agyát nem akarta „megváltoztatni”. Célja saját agyának megváltoztatása volt azáltal, hogy az objektumok mozgásának egy jobb elmeletét ismerje fel.

Hogy az általános tervet konkrétabbá tegyük, térdünk vissza az automata taxi példájához. A végrehajtó elem tartalmazza minden tudást és eljárások halmozát, amelyek a taxivezetés cselekvéseinek kiválasztásához szükségesek. A taxi ezen végrehajtó elem segítségével megy ki az utakra és közlekedik. A kritikus megfigyeli a világot, és továbbadja az információt a tanuló elemeknek. Például miután a taxi három forgalmi sávon keresztül egy gyors balkanyart vett, a kritikus megfigyeli a többi vezető által használt sokkoló nyelvezetet. Ebből a kísérletről a tanuló elem képes egy szabályt formálni, miszerint ez egy rossz cselekvés volt, és a végrehajtó elem módosul az új szabály beillesztésével. A problémagenerátor megfigyelheti a viselkedés bizonyos területeit, melyek javításra szorulnak, és kísérleteket javasol, mint például a fékek kipróbálását különböző útfelszíneken és különböző viszonyok között.

A tanuló elem módosíthatja az ágensdiagramokon (2.9., 2.11., 2.13. és 2.14. ábrák) „tudás” komponensek bármelyikét. A legegyszerűbb esetek magukban foglalják az eszlelési szekvenciából történő direkt tanulást. Az egymást követő környezeti állapot

párok megfigyelése lehetővé teszi az ágens számára annak megtanulását, hogy a „világ hogyan változik”, és cselekvései eredményeinek megfigyelése lehetővé teszi a „mit okoznak a cselekvésem” megtanulását. Például ha a taxi vezetője adott erővel fékez, mikor a taxi nedves úton halad, akkor hamarosan tapasztalni fogja, ténylegesen mekkora lassulást ért el. Világos, hogy ez a két tanulási feladat sokkal nehezebb, ha a környezet csak részlegesen figyelhető meg.

Az előző bekezdésben szereplő tanulási formák nem igényelnek hozzáférést külső teljesítményszabványhoz – bizonyos értelemben a mérték univerzális: a kísérletekkel összhangban levő előrejelzések készítése. A helyzet némiképp bonyolultabb egy hasznosságalapú ágens számára, amelyik a hasznosságinformációt szeretné megtanulni. Például tegyük fel, hogy a taxisofőr ágens nem kap borrávalót az utasoktól, akiket az út folyamán teljesen összerázott. A külső teljesítményszabványnak informálnia kell az ágenst, hogy a borrávaló elvesztése egy negatív hozzájárulás az általános teljesítményéhez – ezek után az ágens megtanulhatja, hogy a hirtelen manöverek nem járulnak hozzá a hasznosságához. Bizonyos értelemben a teljesítményszabvány a beérkező észlelés egy részét elkülönítő **jutalomként (reward)** (vagy büntetésként – **penalty**), amely az ágens viselkedési minőségének direkt visszacsatolása. A rögzített teljesítményszabványok, mint például állatokban a fájdalom vagy éhség, ily módon értelmezhetők. Ezt a kérdést a 21. fejezetben tárgyaljuk.

Összegezve, az ágenseknek sokféle komponensük van, és ezen komponensek sokféleképpen reprezentálhatók az ágensprogramban, így úgy tűnik, sokféle tanulási módszer létezik. Van ugyanakkor egy egyesítő motívum. A tanulás az intelligens ágensekben összefoglalható úgy, mint egy folyamat, amely az ágens minden komponensét úgy módosítja, hogy az összhangba kerüljön a rendelkezésre álló visszacsatolt információval, ily módon növelve az ágens általános teljesítményét.

## 2.5. ÖSSZEFOGLALÁS

Ebben a fejezetben szélesben végigszaladtunk a mesterséges intelligencián, amelyet úgy tekintettünk, mint az ágenstervezés tudományát. A következő fontos pontokra kell emlékeznünk:

- **Egy ágens (agent)** olyan valami, amely észlel és cselekszik egy környezetben. Az ágensfüggvény (**agent function**) meghatározza, hogy egy tetszőleges észlelési sorozathoz milyen cselekvést hajtson végre az ágens.
- A **teljesítménymérték (performance measure)** értékeli az ágens viselkedését egy környezetben. Egy **racionális ágens (rational agent)** úgy cselekszik, hogy maximálizálja a teljesítménymérték várható értékét az addigi észlelési sorozat alapján.
- A **feladatkörnyezet (task environment)** leírása tartalmazza a teljesítménymérték, a külső környezet, a beavatkozók és a szenzorok meghatározását. Egy ágens tervezése során az első lépés minden feladatkörnyezet minél teljesebb leírása.
- A feladatkörnyezetek számos lényeges dimenzió mentén különbözhetnek. Lehetnek teljesen vagy részben megfigyelhetők, determinisztikusak vagy sztochasztikusak, epizódsszerűek vagy sorozatszerűek, statikusak vagy dinamikusak, diszkrétek vagy folytonosak, valamint egyágensesek vagy többágensesek.

- Az **ágensprogram (agent program)** implementálja az ágensfüggvényt. Többféle alapvető ágensfelépítés létezik attól függően, hogy milyen információ jelenik meg és használódik fel a döntési folyamatban. A konstrukció különbözhet hatékonyságában, kompaktságában és rugalmasságában. Az ágensprogram megfelelő konstrukciója a környezet természetétől függ.
- Az **egyszerű reflexszerű ágensek (simple reflex agents)** azonnal válaszolnak az észlelésekre, míg a **modellalapú reflexszerű ágensek (model-based reflex agents)** a világ aktuális észlelésekből nem nyilvánvaló aspektusait belső állapotukban tartják nyilván. A **célorientált ágensek (goal-based agents)** céljaik elérése érdekében cselekszenek, a **hasznosságorientált ágensek (utility-based agents)** pedig saját „boldogságukat” próbálják meg maximalizálni.
- minden ágens javíthatja a teljesítményét **tanulás (learning)** segítségével.

## Irodalmi és történeti megjegyzések

A cselekvés központi szerepe az intelligenciában – a praktikus következtetés fogalma – egészen Arisztotelész *Nikomakhoszi Etikájáig* vezethető vissza. A praktikus következtetés McCarthy (McCarthy, 1958) nagy hatású cikkének is tárgya volt: *Programs with Common Sense* (Józan ésszel bíró programok). A robotika és szabályozáselmélet területei természetükönél fogva alapvetően fizikai ágensek tervezésével foglalkoznak. A **vezérlő (controller)** koncepciója a szabályozáselméletben megegyezik az MI ágens fogalmával. Talán meglepő, hogy az MI történelménél nagy részében az ágensek elközlönlített komponenseire – kérdés-felelet rendszerek, tételelbizonyítók, látórendszerök és így tovább – koncentrált inkább, mint a teljes ágensekre. Az ágensek leírása Genesereth és Nilsson (Genesereth és Nilsson, 1987) cikkében egy nagy hatású kivétel volt. A teljes ágenskép ma már széles körben elfogadott a szakmában, és ez lett az újabb írások központi témája (Pool és társai 1998; Nilsson 1998).

Az 1. fejezet a racionalitás koncepciójának gyökereit a filozófiára és közigazdaságtanra vezette vissza. Az MI-ben a nyolcvanas évek közepéig ez a koncepció csak periódikus érdeklődést kapott, amikor elkezdte elborítani a terület megfelelő technikai megalapozásának tárgyalásait. Jon Doyle cikke azt járult, hogy a racionális ágensek tervezése képezi majd az MI fő küldetését, míg más népszerű területek leválnak új tudományágakat formálva (Doyle, 1983).

A környezetek tulajdonságainak gondos figyelembevétele és hatásuk a racionális ágensek tervezésére leginkább a szabályozáselméleti tradícióban látható – például a klasszikus szabályozó rendszerek (Dorf és Bishop, 1999) teljesen megfigyelhető, determinisztikus környezeteket kezelnek; a sztochasztikus optimális vezérlés (Kumar és Varaiya, 1986) részlegesen megfigyelhető, sztochasztikus környezeteket kezel; és a hibrid szabályozó (Henzinger és Sastry, 1998) olyan környezetekkel foglalkozik, amelyek mind diszkrét, mind folytonos elemeket tartalmaznak. A teljesen és a részlegesen megfigyelhető környezetek közötti különbségtétel az operációkutatás területén kifejlődött **dinamikus programozás (dynamic programming)** irodalmában szintén központi kérdés (Puterman, 1994), amit a 17. fejezetben fogunk tárgyalni.

A reflexszerű ágensek voltak az elsődleges modellek a pszichológiai behavioristák, mint például Skinner (Skinner, 1953) számára, aki megpróbálta az organizmusok pszi-

chológiáját kizárolag bemenet/kimenet, vagy ínger/válasz leképezésekre visszavezetni. A pszichológiában a behaviorizmustól a funkcionálizmusig jutó fejlődés, amelyet legalább részben a számítógép-metafora ágensekre történő alkalmazása idézett elő (Putnam, 1960; Lewis, 1966), az ágens belső állapotának a megjelenését eredményezte. Az MI munkáinak nagy része az állapottal rendelkező tiszta reflexszerű ágensek ötletét túl egyszerűnek tartja ahhoz, hogy előrelépést jelentsen, de Rosenschein (Rosenschein, 1985) és Brooks (Brooks, 1986) munkája megkérdezte ezt a feltételezetést (25. fejezet). Az elmúlt években jelentős munka zajlott annak érdekében, hogy hatékony algoritmusokat találjanak komplex környezetek nyomon követésére (Hanscher és társai, 1992). A Remote Agent program, amely a Deep Space One űrhajót vezérelte (leírása a 60. oldalon található) egy különösen meggyőző példa (Muscettola és társai, 1998; Jonsson és társai, 2000).

A célorientált ágenseket mindenhol feltételezték, kezdve Arisztotelész gyakorlati gondolkodásáról alkotott nézetétől McCarthy logikai MI-ról írt korai cikkeig. A Shakey the Robot (Fikes és Nilsson, 1971; Nilsson, 1984) volt egy logikai, célorientált ágens első robot formájú megtesztelése. A célorientált ágensek teljes logikai elemzése Genesereth és Nilsson (Genesereth és Nilsson, 1987) cikkében jelent meg, és egy célorientált programozási módszertant, az úgynevezett ágensorientált programozást Shoham (Shoham, 1993) dolgozott ki.

A célorientált nézőpont a kognitív pszichológia hagyományában is domináns a problémamegoldás területén, kezdve a jelentős mértékben meghatározó *Human Problem Solving* (Emberi problémamegoldás) (Newell és Simon, 1972) művel, és folytatódva Newell későbbi munkájában (Newell, 1990). A célok, tovább bontva vágyakra (általában) és szándékokra (jelenleg követett), központi téma a Bratman (Bratman, 1987) által kifejlesztett ágenselméletnek is. Ez az elmélet hatással volt mind a természetes nyelv megértésére, mind a többágenses rendszerekre.

Horvitz kifejezetten azt javasolja, hogy az MI alapja a racionalitás mint az elvárt hasznosság maximalizálása legyen (Horvitz és társai, 1988). Pearl írása (Pearl, 1988) volt az első az MI-ben, amely részletesen tárgyalta a valószínűséget és a hasznosság-elméletet; az írás a bizonytalanság melletti következetés és döntéshozatal praktikus módszereinek kifejtésével valószínűleg a legnagyobb önálló tényező volt a hasznosság-orientált ágensek felé történő elmozdulásban a kilencvenes években (V. rész).

A 2.15. ábrán bemutatott általános tanuló ágens tervezés klasszikus a gépi tanulás irodalmában (Buchanan és társai, 1978; Mitchell, 1997). Ezen felépítés programokban megtettesülvő példái legalább olyan messze nyúlnak vissza, mint Arthur Samuel dáma-játékot játszó tanuló programja (Samuel, 1959, 1967). A tanuló ágenseket a VI. rész tárgyalja részletesen.

Az ágensek és ágenstervezés iránti érdeklődés gyorsan növekedett az elmúlt években, részben az internet, valamint az automatizált és mobil szoftbotok iránti igény növekedésével (Etzioni és Weld, 1994). A releváns cikkeket a *Readings in Agentsben* (Huhns és Singh, 1998), valamint a *Foundations of Rational Agencyben* (Wooldridge és Rao, 1999) gyűjtötték össze. A *Multiagent Systems* (Weiss, 1999) nyújtja az ágenstervezés sok aspektusának egységes alapját. Az ágenseknek szentelt konferenciák között van az International Conference on Autonomous Agents, az International Workshop on Agent Theories, Architectures, and Languages, és az International Conference on Multiagent Systems. Végezetül, a *Dung Beetle Ecology* (Ganajtúró bogár ökológia) (Hanski és Cambefort, 1991) érdekes információk gazdag választékát kínálja a ganajtúró bogarak viselkedéséről.

## Feladatok

- 2.1.** Definiálja saját világait a következő fogalmakkal: ágens, ágensfüggvény, ágensprogram, racionalitás, autonómia, reflexszerű ágens, modellalapú ágens, célorientált ágens, hasznosságorientált ágens, tanuló ágens!
- 2.2.** Mind a teljesítménymérték, mind a hasznosságfüggvény azt méri, hogy mennyire dolgozik jól egy ágens. Magyarázza el a köztük levő különbséget!
- 2.3.** Ez a gyakorlat felfedi az ágensfüggvények és a programok közötti különbségeket.
- Lehetséges több mint egy ágensprogram, amely egy adott ágensfüggvényt valósít meg? Adjon meg egy példát vagy mutassa meg, miért nem lehetséges!
  - Vannak olyan ágensfüggvények, amelyeket nem lehet programmal megvalósítani?
  - Rögzítve egy géparchitektúrát, igaz-e, hogy minden egyes ágensprogram pontosan egy ágensfüggvényt valósít meg?
  - Lehetséges több mint egy ágensprogram, amely egy adott ágensfüggvényt valósít meg?
- 2.4.** Vizsgáljuk meg különböző porszívóvilág-beli ágensfüggvények racionalitását!
- Mutassa meg, hogy a 2.3. ábrán látható egyszerű porszívó ágensfüggvény igenis racionális a 70. oldalon felsorolt feltételezésekkel!
  - Írjon le egy racionális ágensfüggvényt arra a módosított teljesítménymértékre, amely levon egy pontot minden elmozdulásért! Szüksége van-e a függvénynek megfelelő ágensprogramnak belső állapotra?
  - Mutasson lehetséges ágensterveket olyan esetekre, amikor tiszta négyzetek piszkossá válhatnak, és a környezet geografiája ismeretlen! Értelmes dolog-e, hogy ezekben az esetekben az ágens saját tapasztalataiból tanuljon? Ha igen, akkor mit kellene megtanulnia?
- 2.5.** Az alábbi ágensek mindegyikére dolgozza ki a feladatkörnyezetek TKBÉ-leírását!
- Robot focijátékos;
  - Internetes könyvvásárló ágens;
  - Autonóm Mars terepjáró;
  - Matematikusok tételelbizonyító segítője.
- 2.6.** A 2.5. példa minden egyes ágenstípusára jellemesse a környezeteket a 2.3. fejezetben megadott tulajdonságok alapján, majd válasszon ki megfelelő ágensterveket!
- A következő feladatok mindegyike egy környezetnek és egy sor ágensnek a megvalósítására vonatkozik a porszívóvilágban.
- 2.7.** Valósítson meg egy teljesítménymérő környezetszimulátort a 2.2. ábrán mutatott és 70. oldalon specifikált porszívóvilágra! Implementációja legyen moduláris, hogy könnyen megváltoztathassa az érzékelőket, a beavatkozókat és a környezeti jellemzőket (méret, alak, piszok elhelyezkedése stb.). (Megjegyzés: Bizonyos ope-

rációs rendszerekre és programozási nyelvekre vannak kész implementációk az internetes kódtárban.)

- 2.8.** Tervezzen és valósítson meg egy tiszta reflexszerű ágenst a 2.7. feladat környezetére! minden lehetséges kezdeti koszeloszlásra és ágenselhelyezkedésre futtassa le a környezetsimulárt ezzel az ágessel! Jegyezze fel az ágens teljesítménymértekét minden egyes konfigurációra és az átlagos pontszámát is!
- 2.9.** Vizsgálja meg a 2.7. gyakorlat egy módosított környezetét, amelyben az ágenst egy ponttal büntetik minden egyes elmozdulásért.
- Lehet-e tökéletesen racionális egy egyszerű reflexszerű ágens ebben a környezetben? Magyarázza meg, miért!
  - Mi a helyzet az állapottal rendelkező reflexszerű ágensekkel? Tervezzen egy ilyen ágenst!
  - Hogyan változna meg az (a) és (b) pontokra adott válasza, ha az ágens észlelései a környezet összes négyzetére megmondják neki, hogy azok tiszták vagy piszkosak?
- 2.10.** Vizsgálja meg a 2.7. feladat porszívókörnyezetének egy módosított változatát, amelyben a környezet alakja – mérete, határai és akadályai – ismeretlen, mint ahogy a kosz kezdeti elhelyezése is az! (Az ágens mehet *Fel* és *Le*, valamint *Balra* és *Jobbra* is.)
- Lehet-e tökéletesen racionális egy egyszerű reflexszerű ágens ebben a környezetben? Magyarázza meg, miért!
  - Egy *randomizált* ágensfüggvénnyel rendelkező reflexszerű ágens jobban teljesíthet egy egyszerűnél? Tervezzen egy ilyen ágenst, és mérje meg a teljesítményét különböző környezetekben!
  - Tud olyan környezetet tervezni, ahol az Ön randomizált ágense igen rosszul fog teljesíteni? Mutassa meg az eredményeit!
  - Egy állapottal rendelkező reflexszerű ágens jobban teljesíthet egy egyszerűnél? Tervezzen egy ilyen ágenst, és mérje meg a teljesítményét különböző környezetekben! Tud ilyen típusú racionális ágenst tervezni?
- 2.11.** Ismételje meg a 2.10. feladatot egy olyan esetre, amikor az elhelyezkedés érzékelőt egy „ütközés” érzékelő váltja fel, ami érzékeli az ágens olyan elmozdulási kísérleteit, amikor nekiütközik egy akadálynak vagy átlépi a környezet határát! Tegye fel, hogy az ütközésérzékelő működése leáll! Hogyan kellene az ágensnek viselkednie?
- 2.12.** Az eddigi feladatok porszívókörnyezetei determinisztikusak voltak. Vizsgáljon meg lehetséges ágensprogramokat a következő sztochasztikus variációkra:
- Murphy törvénye: az idő 25%-ában a *Felszívás* művelet nem tudja feltartani a padlót, ha az piszkos, illetve kiönti rá a koszt, ha az tiszta. Hogyan érinti az ágens programját az, ha a piszokérzékelő az idő 10%-ában rossz választ ad?
  - Kisgyerekek: minden időlétében minden négyzetnek 10% esélye van bekoszolódni. Tud javasolni egy racionális ágenstervet erre az esetre?

## **II. RÉSZ**

# **PROBLÉMA- MEGOLDÁS**

---

# 3. PROBLÉMAMEGOLDÁS KERESÉssel

*Ebben a fejezetben megnézzük, hogy egy ágens hogyan tudja megtalálni azt a cselekvéssorozatot, amely elvezet a célhoz, ha erre egyetlen egyedi cselekvés sem elegendő.*

A 2. fejezetben említett legegyszerűbb ágensek a reflexszerű ágensek voltak, amelyek cselekvései az állapotok és a cselekvések közötti közvetlen leképzésen alapulnak. Az ilyen ágensek nem képesek olyan környezetben tevékenykedni, ahol a leképzés tárolási igénye túl nagy lenne, vagy ahol a leképezés megtanulása túl sok időt venne igénybe. A célorientált ágensek azonban sikerrel járhatnak a jövőbeli cselekvéseknek és annak a számbavételével, hogy ezen cselekvések kimenetelei mennyire kívánatosak.

Ebben a fejezetben a célorientált ágensek egyik típusát, a **problémamegoldó ágenseket** (**problem-solving agent**) ismertetjük. A problémamegoldó ágensek úgy határozzák meg, mit is kell tenniük, hogy olyan cselekvéssorozatokat keresnek, amelyek a kívánt állapotokba vezetnek. Azzal kezdjük, hogy pontosan megfogalmazzuk a „problémát” és a „megoldását” felépítő alkotóelemeket, és számos példával illusztráljuk ezen definíciókat. Ezek után néhány általános rendeltetésű keresési algoritmust mutatunk be, amelyek e problémák megoldására alkalmasak, majd az egyes algoritmusok előnyeit összehasonlítjuk. Az algoritmusok **nem informáltak** (**uninformed**) abban az értelemben, hogy a probléma definícióján túlmenően más információval a problémáról nem rendelkeznek. A 4. fejezet **informált** (**informed**) keresési stratégiákat tárgyal, amelyek rendelkeznek valamilyen elképzeléssel arról, hogy a megoldást merrefelé is kell keresni.

Ebben a fejezetben az algoritmuselemet területén használt fogalmakat alkalmazzuk. Az aszimptotikus komplexitás (azaz az  $O()$  jelölésmód) és az NP-teljesség fogalom-körében nem járatós olvasó számára az A) függelék nyújt bővebb információkat.

## 3.1. PROBLÉMAMEGOLDÓ ÁGENSEK

Az intelligens ágensekről feltessziük, hogy a teljesítménymértéküket maximalizálják. Mint azt a 2. fejezetben láttuk, a feladat valamelyest leegyszerűsödik, ha az ágens egy cél (**goal**) tud maga elé tűzni, és megpróbálja azt elérni. Először vizsgáljuk meg, hogy az ágens hogyan és miért teheti ezt meg.

Tegyük fel, hogy az ágensünk a romániai Arad városában tartózkodik, és országjáró tűrája vége felé közeledik. Az ágens teljesítménymértéke számos tényezőt tartalmaz: például szépen le szeretné barnulni, tökéletesíteni szeretné a román nyelvtudását, látni szeretné az összes nevezetességet, élvezni az éjszakai életet (amilyen van), kerülni a másnaposságot stb. A döntési probléma igen bonyolult, sok kompromisszummal és

útikönyvek gondos tanulmányozásával jár. Tételezzük most fel, hogy ágensünk egy Bukarestből érvényes, vissza nem váltható repülőjeggyel rendelkezik másnapra. Ebben a helyzetben értelmes célnak (*goal*) tűnik, hogy eljusson Bukarestbe. Mindazon cselekvéseket további mérlegelés nélkül el lehet dobni, amelyekkel nem lehet időben Bukarestbe érni, amivel az ágens döntési problémája nagymértékben egyszerűsödik. A célok segítik megszervezni a viselkedést, mivel korlátozzák az ágens által elérendő dolgok számát. A pillanatnyi helyzetén és az ágens hasznosságmértékén alapuló **cílmegfogalmazás (goal formulation)** a problémamegoldás első lépése.

Célnak a világ állapotainak egy halmazát tekintjük, pontosabban azon állapotok halma-zát, amelyekben a cél teljesül. Az ágens feladata megkeresni, hogy amely cselekvések sorozata juttatja el őt egy célállapotba. Mielőtt azonban ezt megtehetné, el kell döntenie, hogy milyen cselekvéseket és állapotokat vizsgáljon meg. Amennyiben olyan szintű cselekvésekkel kellene dolgozna, mint például „a bal lábadat tudd előrebb fél méterrel” vagy „a kormánykereket hat fokkal tékerd balra”, az ágens még a parkolóból sem lenne képes kihajtani, nemhogy időben eljutni Bukarestbe, mivel az ilyen részletek szintjén a világban túl sok a bizonytalanság, és a megoldás túl sok lépésből állna. A **problémamegfogalmazás (problem formulation)** az a folyamat, amely során eldöntjük, hogy amely cselekvéseket és állapotokat vegyük figyelembe, ha egy cél adott. Ezt a folyamatot részletesebben is tárgyalni fogjuk. Egyelőre tételezzük fel, hogy az ágens az egyik nagyobb városból egy másik nagyobb városba való autózás szintjén szemléli a cselekvéseket. Ebből adódóan a figyelembe vett állapotok annak felelnek meg, hogy az ágens éppen egy adott városban van.<sup>1</sup>

Az ágensünk azt tüzte ki célul, hogy Bukarestbe megy, és azt fontolatja, hogy Aradról hová menjen. Három út vezet ki Aradról: egy Nagyszében felé, egy Temesvár felé és egy Nagyzerénd felé. Ezek egyike sem éri el közvetlenül a célt, így amennyiben ágensünk nem ismeri ki magát tökéletesen Romániában, nem fogja tudni, hogy melyik úton induljon el.<sup>2</sup> Más szóval az ágens nem fogja tudni eldönteni, hogy a lehetséges cselekvések közül melyik a legjobb, mert nincs elegendő információja a cselekvések hatására előálló állapotokról. Amennyiben az ágensnek nem áll rendelkezésére további tudás, akkor itt megakad. A legjobb amit tehet, hogy a cselekvésekkel véletlenszerűen kiválaszt egyet.

Tegyük fel azonban, hogy ágensünknek akár papíron, akár a fejében van egy Románia-térképe. A térkép jelentősége, hogy információt nyújthat az ágensnek azokról az állapotokról, ahova eljuthat, illetve a végrehajtható cselekvésekről. Az ágens ezen információt felhasználhatja egy hipotetikus utazás megtervezésére ezeken a városokon keresztül, hogy megtalálja a legjobb utat, amely végül elvezet Bukarestbe. Miután a térképen megtalált egy utat Aradról Bukarestbe, a célját el tudja érni, ha az utazás egyes állomásaihoz tartozó vezetési szakaszokat végrehajtja. Általánosságban: *amennyiben egy ágens előtt közvetlenül több ismeretlen értékű lehetőség áll, ki tudja választani,*

<sup>1</sup> Vegyük észre, hogy ezek az állapotok valójában a világ állapotaiból képezett nagy halmazoknak felelnek meg, mivel a világ állapotai a valóság minden aspektusát meghatározzák. Fontos észben tartani a problémamegoldás állapotai és a világ állapotai közötti különbséget.

<sup>2</sup> Feltételezzük, hogy legtöbb olvasónk ugyanebben a cípőben jár, és könnyen át tudja érezni ágensünk tanácsatlanságát. Ezután elnázést kérünk a romániai olvasóktól, akik számára ezen pedagógiai eszköz nem jelent segítséget.

*hogy mit tegyen, ha először megvizsgálja a különböző lehetséges cselekvéssorozatokat, amelyek ismert értékű állapotokba vezetnek, majd ezután kiválasztja a legjobbat.*

Az ilyen sorozat előállítási folyamatát **keresésnek (search)** nevezzük. A keresési algoritmus bemenete egy probléma, kimenete pedig egy cselekvéssorozat formájában előálló **megoldás (solution)**. Miután előállt egy megoldás, az abban szereplő cselekvésekkel végre lehet hajtani. Ezt **végrehajtási (execution)** fázisnak szokás nevezni. Így előállt egy egyszerű „fogalmazd meg, keresd meg, hajtsd végre” ágenstervezési séma. Az előállt sémát a 3.1. ábra mutatja. A cél és a megoldandó probléma megfogalmazása után az ágens egy keresési eljárást hív meg. Ezek után a megoldást használja cselekvései vezérlésére, azzal teszi, amit a megoldás a következő lépések javasol – ez tipikusan a cselekvéssorozat első lépése –, és a végrehajtott lépést törli a cselekvéssorozatból. Miután a megoldást végrehajtotta, az ágens új célt keres.

```

function EGYSZERŰ-PROBLÉMA-MEGOLDÓ-ÁGENS(érzékelés) returns egy cselekvés
  inputs: érzékelés, egy érzékelés
  static: sorozat, egy cselekvéssorozat, kezdetben üres
            állapot, a világ pillanatnyi állapotának valamelyen leírása
            cél, egy cél, kezdetben üres
            probléma, egy probléma megfogalmazása

  állapot  $\leftarrow$  ÁLLAPOT-FRISSÍTÉS(állapot, érzékelés)
  if sorozat üres then
    cél  $\leftarrow$  CÉL-MEGFOGALMAZÁS(állapot)
    probléma  $\leftarrow$  PROBLÉMA-MEGFOGALMAZÁS(állapot, érzékelés)
    sorozat  $\leftarrow$  KERESÉS(probléma)
    cselekvés  $\leftarrow$  AJÁNLÁS(sorozat, állapot)
    sorozat  $\leftarrow$  MARADÉK(sorozat, állapot)
  return cselekvés

```

**3.1. ábra.** Egy egyszerű problémamegoldó ágens. Először a célt és a problémát fogalmazza meg, majd a problémát megoldó cselekvéssorozatot keres, végül a cselekvésekkel egyenként végrehajtja. Amikor kész vele, egy másik célt fogalmaz meg, és az egészet újrakezdi. Jegyezzük meg, hogy a cselekvéssorozata végrehajtása alatt az ágens az érzékeléseivel nem foglalkozik. Feltételezi, hogy az általa megtalált megoldás minden működőképes.

Először a problémamegfogalmazás folyamatát írjuk le, majd a fejezet további részeit a KERESÉS függvény különböző változatainak ismertetésére fordítjuk. Ebben a fejezetben nem tárgyalunk részletesen az ÁLLAPOT-FRISSÍTÉS és a CÉL-MEGFOGALMAZÁS függvényeket.

Mielőtt a részletekbe belemerülnék, egy pillanatra állunk meg és nézzük meg, hogy a problémamegoldó ágens hogyan illik bele a 2. fejezetben tárgyalt ágensek és környezetek sokaságába. A 3.1. ábrán látható ágenstervezet feltételezi, hogy a környezet **statisztikus (static)**, mivel a probléma megfogalmazása és megoldása semmilyen változásról nem vesz tudomást, amely esetleg a környezetben beáll. Az ágenstervezet azt is feltételezi, hogy a kezdeti állapot ismert. Ennek ismerete akkor a legkönnyebb, amikor a környezet **megfigyelhető (observable)**. Az „alternatív cselekvések számortartása” tulajdonképpen azt tételezi fel, hogy a környezet **diszkrét (discrete)**. Végül a legfontosabb:

az ágenstervezet azt is feltételezi, hogy a környezet **determinisztikus** (**deterministic**). A problémamegoldások egyedi cselekvéssorozatok, így a váratlan eseményeket nem is tudják figyelembe venni. A megoldások végrehajtása ráadásul az érzékelésről nem is vesz tudomást! Annak az ágensnek, amely mondhatni csukott szemmel hajtja végre a terveit, elégé biztosnak kell lennie a dolgában (szabályozáselméletben az ilyen rendszereket **nyílt hurkúnak** – **open-loop** – hívják, mert az érzékelések figyelmen kívül hagyása az ágens és a környezete közötti hurkot felbontja). E feltételezések azt jelentik, hogy a környezetek lehető legegyszerűbbikével foglalkozunk, és ez egyben indokolja azt is, hogy e fejezet miért a könyv elején található. A 3.6. alfejezet rövid bepillantást ad abba, hogy mi történik, amikor a megfigyelhetőség és a determinizmus feltételezéséről lemondunk. Ezt a téma sokkal mélyebben a 12. és a 17. fejezet tárgyalja.

## Jól definiált problémák és megoldások

Egy **probléma (problem)** formális megragadásához az alábbi négy komponensre van szükség:

- A **kiinduló állapot (initial state)**, amiből az ágens kezdi a cselekvéseit. A romániai ágensünk kezdeti állapotát például *Benn(Arad)*-ként lehetne leírni.
- Az ágens rendelkezésére álló lehetséges **cselekvések (actions)** halmaza. A leginkább használatos leírás<sup>3</sup> az **állapotátmenet-függvényt (successor function)** alkalmazza. Egy adott  $x$  állapot esetén az ÁLLAPOTÁTMENET-Fv( $x$ ) visszaadja a rendezett (*cselekvés, utódállapot*) párok halmazát, ahol minden cselekvés az  $x$  állapotban legális cselekvések egyike, és minden utódállapotot egy cselekvésnek az  $x$  állapotra való alkalmazásával nyerünk. A *Benn(Arad)* állapotban az állapotátmenet-függvény a romániai problémára a

$\{\langle Menj(Nagyszében), Benn(Nagyszében) \rangle, \langle Menj(Temesvár), Benn(Temesvár) \rangle, \langle Menj(Nagyzerénd), Benn(Nagyzerénd) \rangle\}$

párokat adná vissza.

A kezdeti állapot és az állapotátmenet-függvény együttesen implicit módon definíálják a probléma **állapotterét (state space)**: azon állapotok halmazát, amelyek a kiinduló állapotból elérhetők. Az állapotter egy gráfot alkot, amelynek csomópontjai az állapotok és a csomópontok közötti élek a cselekvések. (Románia 3.2. ábrán látható térképe állapotterként értelmezhető, ha minden útját kétirányú gépkocsivezetési cselekvésként fogjuk fel.) Az állapotter egy **útja (path)** az állapotok egy sorozata, amely állapotokat a cselekvések egy sorozata köt össze.

- A **célteszt (goal test)**, amely meghatározza, hogy egy adott állapot célállapot-e. Néha létezik a lehetséges célállapotok egy explicit halmaza, és a teszt egyszerűen megnézi, hogy az ágens elérte-e ezek egyikét. Romániában az ágens célja a  $\{Benn(Bukarest)\}$  szingleton.

<sup>3</sup> Egy alternatív megfogalmazás az **operátorok (operators)** egy halmaza, amelyeket egy állapotra alkalmazva lehet az utódállapotokat generálni.

Néha a cél valamilyen absztrakt tulajdonsággal van definiálva, nem pedig explicit módon felsorolt állapothalmazzal. A sakkban például az úgynevet „sakk-matt” állapot elérése a cél, amelyben az ellenfél királya az őt ért támadás elől nem tud elmenekülni.

- Egy **útköltség-** (**path cost**) függvény, amely minden úthoz hozzárendel egy költséget. A problémamegoldó ágens azt a költségfüggvényt fogja választani, amely a saját hatékonysági mértékének felel meg. A Bukarestbe siető ágens számára az idő a lényeg, így az útköltség lehetne például az út a km-ben kifejezett hossza. Ebben a fejezetben az út költségének az utat alkotó egyes cselekvések költségének összegét fogjuk tekinteni az út mentén. Az  $x$  állapotból az  $y$  állapotba vezető cs cselekvés **lépésköltsége** (**step cost**) legyen  $lk(x, cs, y)$ . Romániában a lépésköltségeket a 3.2. ábra mutatja úttávolságok formájában. Feltételezzük, hogy a lépésköltségek nemnegatívak.<sup>4</sup>

Az előbbi elemek definiálják a problémát és egy közös adatstruktúrába foghatók, amit a problémamegoldó algoritmus bemenetének tekintünk. A probléma **megoldása** (**solution**) nem más, mint a kiinduló állapotból a célállapotba vezető út. A megoldás kvalitását az útköltségfüggvény méri, és egy **optimális megoldásnak** (**optimal solution**) a megoldások között a legkisebb lesz az útköltsége.

## A problémák megfogalmazása

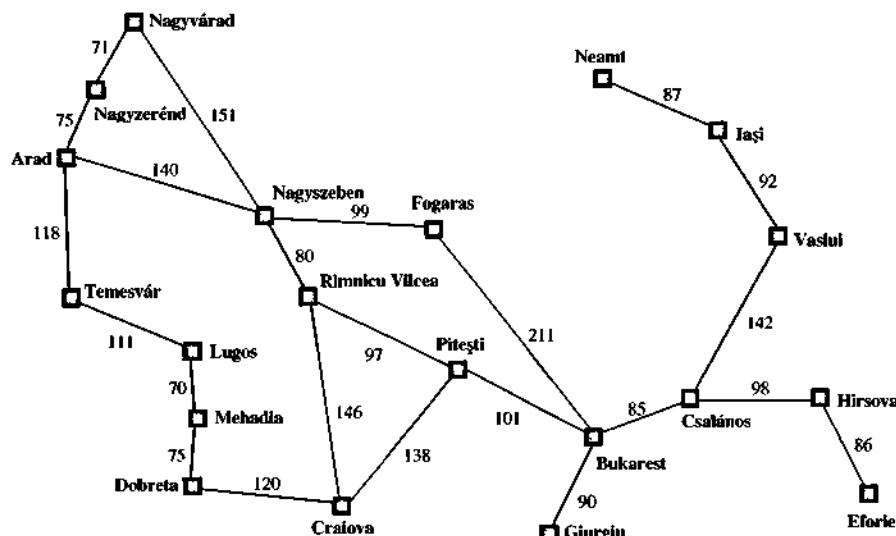
Az előbbi részben a „hogyan jutunk el Bukarestbe” probléma olyan megfogalmazását javasoltuk, amely a kezdeti állapotból, az állapotámenet-függvényből, a célállapottesztből és az útköltségből áll. Ez a megfogalmazás értelmesnek tűnik, a valós világ számos aspektusát mégis figyelmen kívül hagyja. Hasonlítsuk csak össze az általunk választott egyszerű állapotleírást, *Benn(Arad)*, egy tényleges országjáró kirándulással, ahol a világ állapota rengeteg mindenzt tartalmazhat: kivel utazunk, mit közvetít a rádió, milyen tájat látunk az ablakon át, van-e közelben rendőr, milyen messze van a következő pihenőhely, milyen az út állapota, milyen az időjárás és sok-sok más.

Ezeket a részleteket kihagytuk az állapotleírásokból, mert nem lényegesek a Bukarestbe vezető út megkeresésében. Egy reprezentációból a részletek eltávolítását **absztraktiának** (**abstraction**) nevezzük.

Az állapotleírás absztrahálása mellett magukat a cselekvéseket is absztrahálni kell. Egy vezetési cselekvésnek számos hatása van. Amellett, hogy megváltoztatja a gépkocsi és utasai helyét, időbe telik, üzemanyagot fogyaszt, szennyezi a levegőt és megváltoztatja az ágenst (azt mondják, hogy az utazás kiszélesíti a látókört). A mi megfogalmazásunkban csak a helyváltoztatást vesszük figyelembe. Van számos olyan cselekvés, amit teljesen elhagyunk: például a rádió bekapsolását, az ablakon való kinézést, a lassítást, mert rendőr van a közelben stb. És persze a „fordítsd a kormányt balra 3 fokkal” szintű cselekvésekkel sem foglalkozunk.

Lehetünk-e pontosabbak a megfelelő absztraktiós szint meghatározásában? A megválasztott absztrakt állapotokra és cselekvésekre úgy gondolunk, hogy azok a részletes

<sup>4</sup> A negatív költségek következményeivel a 3.17. feladatban foglalkozunk.

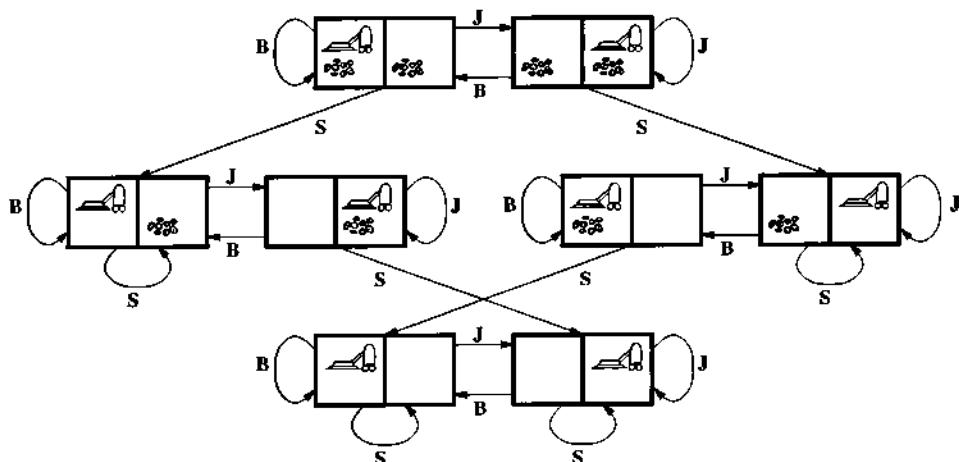


3.2. ábra. Románia egy részének sematikus országúti térképe

valós állapotok és cselekvések egész halmazaihoz tartoznak. Most tekintsük az absztrakt probléma egy megoldását: például az Aradról Nagyszebenbe, majd Rimnicu Vilceába, Piteştibe és Bukarestbe vezető utat. Ez az absztrakt megoldás rengeteg részletesebb útnak felel meg. Például Nagyszeben és Rimnicu Vilcea között vezethettünk bekapcsolt rádióval, majd az utazás további részére kikapcsolhattuk a rádiót. Az absztrakció érvényes, ha az absztrakt megoldást megoldássá fejthetjük ki egy részletesebb világban is. Elégséges feltétel az, hogy minden olyan részletes állapothoz, mint az „Aradon van”, létezik egy részletes út valamelyik olyan állapothoz, mint a „Nagyszebenben van” stb. Az absztrakció hasznos, ha a megoldásbeli cselekvések végrehajtása az eredeti problémánál egyszerűbb. Ebben az esetben ezek elégége egyszerűek ahhoz, hogy egy átlagos gépkocsivezető ágens végre tudja azokat hajtani minden további keresés vagy tervkészítés nélkül. Így egy jó absztrakció megválasztása magában foglalja az érvényesség megőrzése mellett a lehető legtöbb részlet törlesztést, és annak biztosítását, hogy az absztrakt cselekvéseket könnyű legyen véghezvinni. Ha nem lenne meg a hasznos absztrakciók megalkotásának képessége, akkor az intelligens ágensek a valós világban teljesen használatlanak lennének.

## 3.2. PÉLDAPROBLÉMÁK

A problémamegoldó megközelítést rengeteg feladatkörben alkalmazták. Néhányat az alábbiakban sorolunk fel, megkülönböztetve az úgynevezett játék- és a valósvilág-beli problémákat. A **játékproblémák** (*toy problems*) rendeltetése, hogy segítségükkel a különféle problémamegoldó módszereket illusztrálni tudjuk vagy ki tudjuk próbálni. Az ilyen problémák egzakt tömör leírása megadható. Ez azt jelenti, hogy különböző kutatók így könnyűszerrel használhatják az algoritmusok hatékonysági összehasonlításához



3.3. ábra. A porszívóvilág állapotterre. Az élek cselekvéseket jelentenek: B = Balra, J = Jobbra, S = Szív.

A valós világ-beli problémák (**real-world problems**) azok, amelyek megoldása tényleg érdekes az emberek számára. Ezeknek nincs egyetlen általánosan elfogadott megfogalmazásuk, mi azonban megpróbáltuk érzékeltetni az általános megfogalmazásuk jellegét.

## Játékproblémák

Az első vizsgált példa a 2. fejezetben bevezetett porszívóvilág (**vacuum world**) (lásd 2.2. ábra). Ezt problémaként az alábbi módon definiálhatjuk:

- **Állapotok:** az ágens két hely egyikében lehet. Mindegyik lehet piszkos, de lehet tiszta is. Így  $2 \times 2^2 = 8$  lehetséges állapotról beszélhetünk.
- **Kezdeti állapot:** akármelyik állapot lehet kezdeti állapot.
- **Állapotátmenet-függvény:** a három (*Balra*, *Jobbra*, *Szív*) cselekvés alkalmazásából adódó legális állapotokat generálja. A teljes állapotter a 3.3. ábrán látható.
- **Célteszt:** ellenőrzi, hogy minden négyzet tiszta-e.
- **Útköltség:** minden lépés költsége 1. így az út költsége megegyezik az út lépéseinak a számával.

A valós világgal összevetve e probléma jellemzői a diszkrét lokációk, a diszkrét kosz, a megbízható takarítás, valamint az, hogy takarítás után piszok soha nem keletkezik újra (a 3.6. alfejezetben ezeket a feltételezésekkel feladjuk). Fontos megjegyezni, hogy az állapotot az ágens és a kosz helyzete együttesen határozza meg. Egy nagyobb,  $n$  helyű környezetben  $n \cdot 2^n$  állapot van.

A 3.4. ábrán bemutatott **8-as kirakójáték** (**8-puzzle**) egy  $3 \times 3$ -as táblából, 8 számozott kockából és egy üres helyből áll. Az üres hely melletti kockát be lehet csúsztatni az üres helyre. A cél egy meghatározott állás, mint például az ábra jobb oldalán látható állás elérése. A probléma szokásos megfogalmazása a következő:

7	2	4
5		6
8	3	1

Kiinduló állapot

	1	2
3	4	5
6	7	8

Célállapot

3.4. ábra. A 8-as kirakójáték egy tipikus feladvánnya

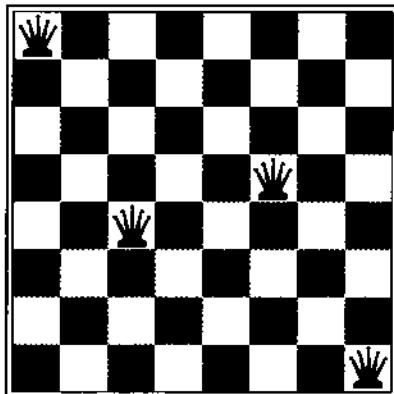
- **Állapotok:** az állapotleírás meghatározza mind a nyolc kocka és az üres hely pozícióját a kilenc lehetséges pozíció egyikében.
- **Kezdeti állapot:** akármelyik állás lehet kezdeti állapot. Figyeljük meg, hogy minden egyes cél pontosan az állapotok feléből lehet elérni (3.4. feladat).
- **Állapotátmenet-függvény:** a négy cselekvés (üres hely meg Balra, Jobbra, Fel, Le) alkalmazásából adódó legális állapotokat generálja.
- **Célteszt:** ellenőrzi, hogy az állapot megegyezik-e a 3.4. ábrán mutatott célállapottal (más célkonfiguráció is lehetséges).
- **Útköltség:** minden lépés költsége 1, így az út költsége megegyezik az út lépéseinak a számával.

Milyen absztraktiókhöz folyamodtunk itt? A cselekvéseket a kezdeti és végállapotuk erejéig absztraháltuk, elmozdítás közben egy kocka közbülső helyzetét figyelmen kívül hagytuk. Az olyan cselekvésekkel is elvonatkoztattunk, mint a tábla megrázása, ha a kockák megakadtak, vagy a kockák késsel való kiszedése és visszahelyezése. Amit kaptunk, az a játék szabálykészlete, a manipulációk fizikai részleteitől eltekintve.

A kirakójáték a csúsztatós kirakójátékok (*sliding-block puzzles*) családjába tartozik, amelyeket gyakran új kereső algoritmusok tesztelésére alkalmaznak az MI-ben. Ez az általános osztály NP-teljes, így nem várható, hogy lényegesen jobb módszert lehet találni az itt és a következő bekezdésben leírt keresési algoritmusoknál. A 8-as kirakójátéknak  $9!/2 = 181\,440$  elérhető állapota van, és így e játék könnyen megoldható. A 15-ös kirakójátéknak (a  $4 \times 4$ -es táblán) kb. 10 billió állapota van, és a legjobb algoritmusok a véletlenül generált eseteket optimálisan néhány milliszekundum alatt oldják meg. A 24-es játéknak (az  $5 \times 5$ -ös táblán) kb.  $10^{25}$  állapota van, és a véletlenszerűen kiválasztott esetek optimális megoldása még mindig kemény dió a jelenlegi gépek és algoritmusok számára.

A 8-királynő probléma (**8-queens problem**) célja, hogy 8 királynőt úgy helyezzük el egy sakktáblán, hogy egyáltalán ne támadják egymást. (Egy királynő egy vele azonos sorban, oszlopban vagy átlóban lévő bábut támad.) A 3.5. ábra egy sikertelen megoldási kísérletet mutat: a jobb szélső oszlopban lévő királynőt támadja a bal felső sarokban lévő királynő.

Bár hatékony célgalgoritmusok léteznek ezen probléma és a teljes  $n$ -királynő probléma megoldására, a királynőfeladat mindenkorral érdekes tesztproblémája marad a keresési algoritmusoknak. Két fő megfogalmazása létezik. Az **inkrementális megfogalmazás-**



**3.5. ábra.** Egy majdnem jó megoldás a 8-királynő problémára. (A megoldás megkeresését az olvasóra bízzuk.)

ban (**incremental formulation**) az operátorok az állapotleírást *bővíti*, az üres állapot-tól kezdve. A 8-királynő probléma esetén ez azt jelenti, hogy a királynőket egyenként helyezzük el a sakktáblán. A **teljes állapot leírásban** (**complete-state formulation**) először felhelyezzük minden a 8 királynőt, majd mozgatjuk őket. Az útköltség minden két esetben érdektelen számunkra, mert csak a végső állapot számít. A probléma első inkrementális megfogalmazása lehet a következő:

- **Állapotok:** egy állapot a 0 ... 8 királynő, tetszőleges elrendezése a táblán.
- **Kezdeti állapot:** a táblán nincs egy királynő sem.
- **Állapotátmenet-függvény:** helyezz egy új királynőt egy üres mezőre.
- **Célteszt:** 8 királynő a táblán és egyik sincs támadás alatt.

Ebben a megfogalmazásban  $64 \times 63 \times \dots \times 57 \approx 1.8 \times 10^{14}$  lehetséges vizsgálandó sorozatunk van. Sokkal ésszerűbb választás lenne, ha figyelembe vennénk azt a tényt, hogy egy már eleve támadt mezőre nincs értelme letenni egy királynőt:

- **Állapotok:**  $n$  ( $0 \leq n \leq 8$ ) királynő olyan elrendezése a táblán, hogy az  $n$  bal oldali oszlopban oszloponként egy található úgy, hogy nem támadják egymást.
- **Állapotátmenet-függvény:** helyezz egy királynőt a bal szélső, még üres oszlopba úgy, hogy azt ne támadja egyetlen királynő sem.

Ez a megfogalmazás a 8-királynő probléma állapotterét  $1.8 \times 10^{14}$ -ről 2057 méretű térré csökkenti, minekutána megoldást könnyű megtalálni. Másrészt 100 királynő a korábbi megfogalmazás  $10^{400}$  állapothoz vezet, míg a javítottnál csak kb.  $10^{52}$  állapotunk lesz (3.5. feladat). Ez óriási redukció, a javított eset azonban még mindig túl nagy az ebben a fejezetben ismertetett algoritmusok számára. A teljes állapot leírásról a 4. fejezetben olvashatunk, az 5. fejezet viszont egy olyan egyszerű algoritmust közölt, amellyel a millió királynőt számláló problémát is könnyen meg lehet oldani.

## Valósvilág-beli problémák

Már korábban láttuk, hogy az útkeresési problémát (**route-finding problem**) hogyan definiálhatjuk a megadott helyek és a közöttük lévő összeköttetések segítségével. Az útkereső algoritmusokat számos területen alkalmazzák. Használják például számítógép-hálózatokban útvonalkeresésre, katonai műveletek tervezésénél és légi útvonaltervező rendszerekben is. Általában nehéz az ilyen problémákat specifikálni. Nézzük meg a légi útvonaltervezési probléma egy egyszerűsített példáját:

- **Állapotok:** minden állapotot egy hely (például egy repülőtér) és az aktuális időpont azonosít.
- **Kezdeti állapot:** a probléma specifikálja.
- **Állapotátmenet-függvény:** azokat az állapotokat adja vissza, amelyek az aktuális repülőtérről valamely másikra olyan menetrendbeli járatok igénybevételével keletkeznek (esetleg tovább pontosítva, hogy mely osztályra és melyik konkrét ülésre vonatkozik az utazás), amelyek később indulnak, mint az aktuális időpont plusz a repülőtéri tranzit ideje.
- **Céltesz:** elértek-e a célállomást?
- **Útköltség:** függ a pénzügyi költségektől, várakozási időtől, repülési időtől, vámkezelési és útlevél-vizsgálati procedúráktól, az ülés minőségétől, a nap időszakától, a repülőgép típusától, az utaskedvezményektől stb.

A kereskedelmi útvonaljavaslati rendszerek a probléma valami hasonló megfogalmazását használják, sok további bonyolítással, hogy a repülőtársaságok bonyolult viteldíj-struktúráit kezelni tudják. minden tapasztalt utas tudja azonban, hogy nem minden repülőtől zajlik a tervezet szerint. Egy igazán jó rendszernek eshetőségi tervezéssel – például az alternatív járatokra történő tartalék helyfoglalásokkal – is tudnia kell foglalkozni, ameddig persze ezeket a költség és az eredeti terv összeomlásának a valószínűsége igazolja.

A **körutazási problémák** (**touring problems**) az útkeresési problémák rokonai egy fontos különbséggel. Tekintsük például azt a problémát, hogy „A 3.2. ábra minden egyes városába látogass el legalább egyszer úgy, hogy Bukarestből indulj, és az utat ott is fejezd be.” Az útkeresési problémához hasonlóan a cselekvés itt is két szomszédos város közötti utazásnak felel meg. De ebben a problémában az állapotokról alapvetően más. Az ágens helye mellett minden egyes állapotban tárolni kell az ágens által korábban meglátogatott városokat is. Így a kiinduló állapot ebben az esetben a „Bukarestben vagyok, eddig meglátogattam {Bukarestet}” állapot lenne, míg egy tipikus közbenső állapot a „Vasluban vagyok, eddig meglátogattam {Bukarestet, Csalánost, Vasluit}” állapot. A céltesz ellenőrizné, hogy az ágens Bukarestben van-e, és végiglátogatta-e minden a 20 várost.

Az **utazó ügynök probléma** (**Travelling Salesperson Problem – TSP**) egy körutazási probléma, amelyben minden várost pontosan egyszer kell meglátogatni. A cél a *legrövidebb* út megkeresése. A probléma NP-nehéz, de hatalmas erőfeszítések történtek a TSP-algoritmusok képességeinek javítására. Ezen algoritmusokat az utazó ügynökök útjainak megtervezése mellett nyomtatott áramkörök furatainak készítésénél a fűrő automata mozgatásának megtervezésére, továbbá áruházak rakodógépeinek útvonaltervezésére is alkalmazták.

A **VLSI elhelyezési (VLSI layout)** problémában komponensek és összeköttetések millióit egy chipen kell elhelyezni, a terület, az áramköri késleltetés és a szort kapacitások minimalizálása és a gyártási kihozatal maximalizálása mellett. Az elhelyezési probléma a logikai tervezés fázisát követi, és általában két részből áll: **cellaelrendezésből (cell layout)** és **huzalozásból (channel routing)**. A cellaelrendezés során az áramkör primitív komponenseit cellákba rendezik, amelyek mindegyike valamilyen jól definiált funkciót lát el. minden egyes cellának rögzített nyomtatási képe (méret és alak) van, és adott számú összeköttetést igényel a többi cellához. A cél az, hogy az egyes cellákat úgy helyezzük el a chipen, hogy azok ne lapolódjanak át, és a cellák között legyen megfelelő hely az összekötő vezetékek számára. A huzalozás során minden egyes vezetéknek megfelelő vezetési útvonalat keresünk a cellák között. Ezek a keresési problémák hihetetlenül összetettek, de mindenkiéppen érdemes a megoldásukat megtalálni. A 4. fejezetben látni fogunk néhány algoritmust, amelyek képesek megoldani ezeket a problémákat.

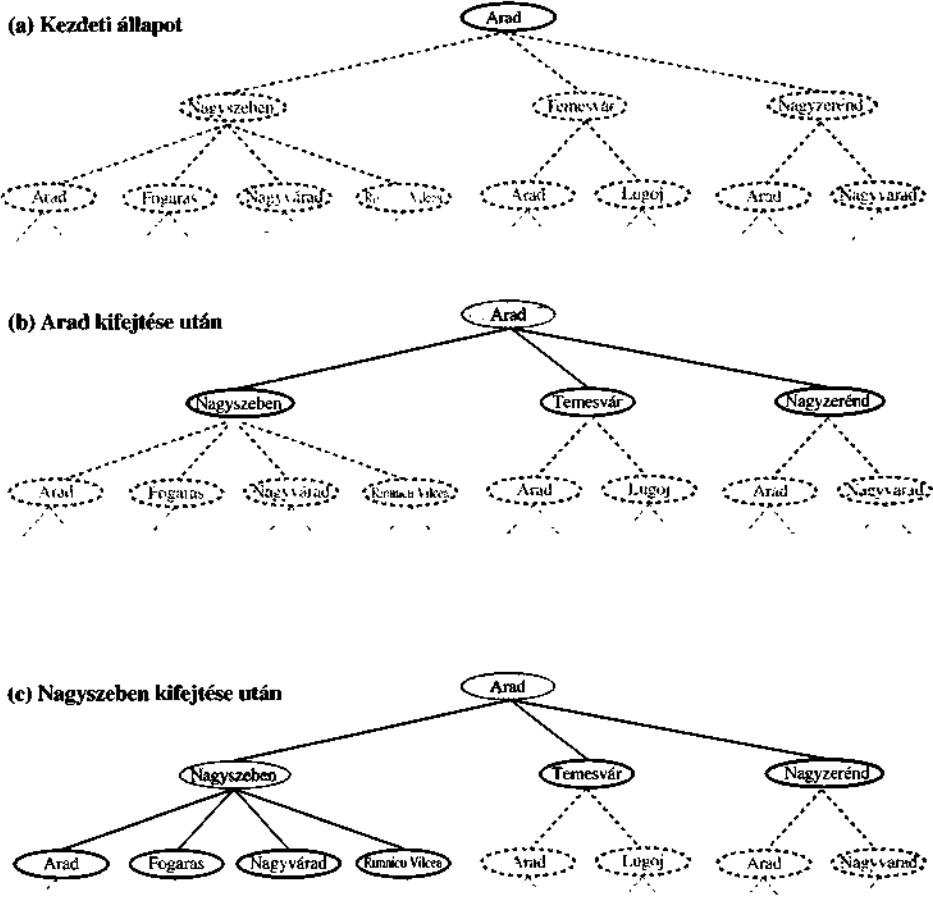
A **robotnavigáció (robot navigation)** a korábban leírt útkeresési probléma általánosítása. Az utak diszkrét halmaza helyett a robot egy folytonos térben mozog (elvben) végiglen cselekvés- és állapothalmazzal. Egy egyszerű, sík felületen mozgó kör alakú robot esetén az állapot lényegében kétdimenziós. Amennyiben a robotnak karja és lába is van, akkor azokat is vezérelni kell, így a keresési tér sokdimenzióssá válik. Már a keresési tér végesessé tételehez is kifinomult technikákra van szükség. A 25. fejezetben megvizsgálunk néhány ilyen módszert. A probléma összetettsége mellett a valós robotoknak foglalkozniuk kell az érzékelők által szolgáltatott jelek és a motorvezérlők hibáival is.

Összetett objektumok robotok általi **automatikus összeszerelését (automatic assembly sequencing)** először a FREDDY robot mutatta be (Michie, 1972). Azóta lassú, de biztos előrehaladás tapasztalható ezen a területen. Ma már például villanymotorok automatikus összeszerelése gazdaságossági szempontból is kivitelezhető. Az összeszerelési feladatoknál a probléma az alkatrészek összeszerelési sorrendjének megkeresése. Amennyiben rossz szerelési sorrendet választunk, valamikor a későbbiek folyamán egy adott alkatrész nem tudunk egy másik alkatrész kiszerelelse nélkül beszerelni. Annak eldöntése, hogy az összeszerelési sorrend egy lépése megvalósítható-e vagy sem, a robotnavigációhoz hasonló összetett geometriai keresési probléma. Így az összeszerelési sorrend megtervezésében a költséges feladat egy kivitelezhető következő lépés megtervezése. minden gyakorlati fontosságú algoritmusnak kerülnie kell az állapottér egy kis töredékén túlmenő feltárasát. Egy másik fontos összeállítási probléma a **fehérjetervezés (protein design)**, ahol a célkitűzés olyan aminosav-szekvencia megkeresése, amely valamilyen betegség gyógyítása szempontjából lényeges tulajdonságokkal rendelkezik és háromdimenziós fehérjévé formálható.

Az utóbbi időben egyre nagyobb a kereslet az **interneten kereső (Internet searching)** szoftverrobotok iránt, amelyek kérdésekre választ, kapcsolódó információkat, vagy kereskedelmi üzletek lehetőségét kutatják fel. A keresési technikák szempontjából ez igazán jó alkalmazás, mert elégnyi nyilvánvaló, hogy az internethez egy gráfszerű kép rendelhető, ahol a csomópontok (weboldalak) linkekkel össze vannak kapcsolva. Az internetkeresés teljes leírását a 10. fejezetre hagyjuk.

### 3.3. MEGOLDÁSOK KERESÉSE

Az előzőkben láttuk, hogyan kell egy problémát definiálni. A hátralévő lépés – a megoldás megkeresése – az állapottérben végrehajtott kereséssel történik. Ebben a részben olyan keresési technikákkal foglalkozunk, amelyek egy **explicit keresési fát** (search tree) használnak, amelyet az állapotteret együttesen definíáló kezdeti állapotból és az állapotátmenet-függvényből generálnak. Általánosságban, ha egy állapotot több úton is elérhetünk, inkább keresési *gráfról*, mint keresési *fáról* beszélünk. Ezzel az igen fontos bonyolítással később a 3.5. alfejezetben foglalkozunk.



**3.6. ábra.** Az Arad és Bukarest közötti útkeresési probléma részleges keresési fái. A kifejtett csomópontokat árnyékoltuk, a legenerált, de még ki nem fejtett csomópontok vastagon keretezettek, a még le nem generált csomópontokat pedig halvány szaggatott vonalak jelzik.

A 3.6. ábrán láthatjuk az Arad és Bukarest közötti útkereső probléma keresési fájának egy kezdeti kifejtését. A keresési fa gyökere az a **keresési csomópont (search node)**, amely a *Benn(Arad)* kezdeti állapotnak felel meg. Az első lépés annak ellenőrzése, hogy vajon ez célállapot-e. Nyilvánvalóan nem az, de fontos ellenőrizni, hogy meg tudjuk oldani az olyan beugrató problémákat is, mint amilyen például az „Aradról indulva jussunk el Aradra”. Mivel ez nem célállapot, egyéb állapotokat is meg kell vizsgálnunk. Ezt az aktuális állapot kifejtésével (**expanding**) tesszük, azaz az állapotátmenet-függvénynek az aktuális állapotra történő alkalmazásával, amivel az állapotok egy új halmazát **generáljuk (generating)**. Ebben az esetben három új állapotot kapunk: *Benn(Nagyszeben)*, *Benn(Temesvár)* és *Benn(Nagyzerénd)*. Most el kell döntenünk, hogy a három lehetőség közül melyik utat kövessük.

A keresés lényege a következő: egy lehetőséget kiválasztani, és a többet későbbre halasztani arra az eshetőségre, ha az első választás nem vezetne megoldásra. Tételezzük fel, hogy elsőnek Nagyszebent választjuk. Ellenőrizzük, hogy ez célállapot-e (nem az), majd kifejtjük, aminek hatására a *Benn(Arad)*, *Benn(Fogaras)*, *Benn(Nagyvárad)* és *Benn(Rimnicu Vilcea)* állapotokat kapjuk. Ezek után e négy közül bármelyiket választhatjuk, vagy akár vissza is mehetünk és választhatjuk Nagyzeréndet vagy Temesvárt. Folytatjuk a kiválasztást, a célállapot-ellenőrzést és a kifejtést, míg egy megoldást nem találunk, vagy amíg el nem fogynak a kifejtendő állapotok. A kifejtendő állapot kiválasztását a **keresési stratégia (search strategy)** határozza meg. Az általános fakereső algoritmus informális megfogalmazását a 3.7. ábra mutatja.

```

function FA-KERESÉS(probléma, stratégia) returns egy megoldás vagy kudarc
  a probléma kezdeti állapotából kiindulva inicializál a keresési fát
  loop do
    if nincs kifejtendő csomópont then return kudarc
    a stratégiának megfelelően válassz ki kifejtésre egy levélcsomópontot
    if a csomópont célállapotot tartalmaz then return a hozzá tartozó megoldás
    else fejtsd ki a csomópontot és az eredményül kapott csomópontokat, és add a keresési fához
  end
```

3.7. ábra. Az általános fakeresési algoritmus informális leírása

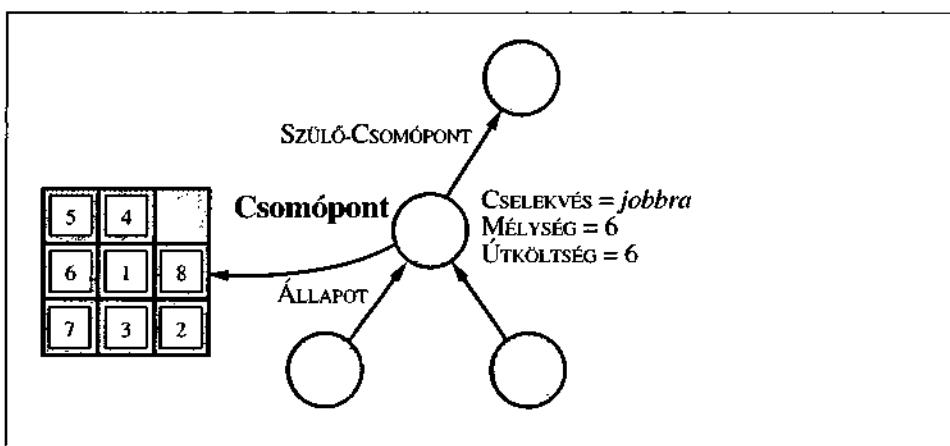
Fontos megkülönböztetni az állapoteret és a keresési fát. Az útkeresési probléma esetében az állapottér csak 20 állapotból áll, minden egyes városhoz tartozik egy állapot. Ebben az állapottérben azonban végtelen sok út vezet, így a keresési fa végtelen sok csomópontból áll. Például az Arad–Nagyszeben, Arad–Nagyszeben–Arad, Arad–Nagyszeben–Arad–Nagyszeben három út az első három a végtelen számú útszekvenciából. (Egy jó keresési algoritmus nyilvánvalóan elkerüli az ilyen utak követését. A 3.5. alfejezetben megmutatjuk hogyan.)

A csomópontokat sokféle módon lehet reprezentálni, de mi feltételezzük, hogy egy csomópont egy öt komponensből álló adatszerkezet:

- **ÁLLAPOT:** az állapottérnek a csomóponthoz tartozó állapota;
- **SZÜLŐ-CSOMÓPONT:** a keresési fa azon csomópontja, amely a kérdéses csomópontot generálta;

- **CSELEKVÉS:** a csomópont szülő-csomópontjára alkalmazott cselekvés;
- **ÚT-KÖLTSÉG:** a kezdeti állapotból a kérdéses csomópontig vezető út általában  $g(n)$ -nel jelölt költsége, ahogy ezt a szülőmutatók jelzik;
- **MÉLYSÉG:** a kezdeti állapotból vezető út lépéseinak a száma.

Fontos visszaidéznünk a csomópontok és az állapotok közti különbséget. A csomópont egy adatnyilvántartásra használt adatszerkezet, amit egy keresési fa leírására használunk. Egy állapot a világ egy konfigurációja. Így a csomópontok a Szülő-Csomópont mutatók által definiált meghatározott úton találhatók, míg az állapotok nem. Továbbá könnyen előfordulhat, hogy két különböző csomópont egyazon állapotot tartalmaz, ha ezt az állapotot két különböző cselekvéssorozattal generálták le. A csomópont-adatstruktúrát a 3.8. ábra mutatja.



**3.8. ábra.** A keresési fa alapvető építő adatstruktúrái a csomópontok. minden csomópontnak van szülője, állapota és számos adminisztráló adatmezeje. A nyílik a gyerektől a szülőig mutatnak.

Valahogy nyilván kell tartanunk a legenerált, kifejtésre váró csomópontokat is – ezt a gyűjteményt, listát peremnek (fringe) nevezik. A perem minden eleme egy **levélcsomópont (leaf node)**, azaz egy olyan csomópont, amelynek a fában nincsenek követői. A 3.6. ábrán a fák pereme a vastagon bekeretezett csomópontokból áll. A perem legszerűbb reprezentációja egy csomópontalmaz lenne. A keresési stratégia ekkor olyan függvény lenne, amely a következő lépésben kifejtendő csomópontot ebből a halmazból választaná ki. Bár elvi szempontból ez nyilvánvaló megoldás, számításigény szempontjából drága lenne, mert a keresési stratégia függvénynek a halmaz minden egyes elemét végig kellene néznie, hogy ki tudja választani a legjobb csomópontot. Ezért a továbbiakban feltesszük, hogy a csomópontgyűjtemény egy **várakozási sorként (queue)** van megvalósítva. A soron végezhető műveletek az alábbiak:

- **SORT-LÉTREHOZ( $elem, \dots$ )** létrehoz egy az adott elemeket tartalmazó sort.
- **ÜRES?( $sor$ )** csak akkor ad vissza igaz értéket, ha a sor üres.
- **ELOSÓ-ELEM( $sor$ )** visszaadja a sor első elemét.

- **TÁVOLÍTSD-EL-AZ-ELSŐ-ELEMET(*sor*)** visszaadja az **ELSŐ-ELEM(*sor*)**-t és ezt eltávolítja a sorból.
- **BESZÚR(*elem, sor*)** egy elemet szűr be a sorba.
- **BESZÚR-MIND(*elemek, sor*)** egy elemhalmazt beszűr a sorba.

Ezen definíciók felhasználásával az általános fakeresési algoritmus egy formálisabb definícióját adhatjuk meg. Ezt a 3.9. ábra mutatja.

```

function FA-KERESÉS(probléma, perem) returns egy megoldás vagy kudarc
  perem  $\leftarrow$  BESZÚR(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), perem)
  loop do
    if ÜRES?(perem) then return kudarc
    csomópont  $\leftarrow$  TÁVOLÍTSD-EL-AZ-ELSŐ-ELEMET(perem)
    if CÉL-TESZT[probléma] alkalmazva az ÁLLAPOT(csomópont) állapotra igazat ad vissza
      then return MEGOLDÁS(csomópont)
    perem  $\leftarrow$  BESZÚR-MIND(KIFEJT(csomópont, probléma), perem)
function KIFEJT(csomópont, probléma) returns a csomópontok halmazát
  követők  $\leftarrow$  üres halmaz
  for each (cselekvés, eredmény) in ÁLLAPOTÁTMENET-Fv[probléma](ÁLLAPOT[csomópont]) do
    s  $\leftarrow$  az új CSOMÓPONT
    ÁLLAPOT[s]  $\leftarrow$  eredmény
    SZÜLŐ-CSOMÓPONT[s]  $\leftarrow$  csomópont
    CSELEKVÉS[s]  $\leftarrow$  cselekvés
    ÚT-KÖLTSÉG[s]  $\leftarrow$  ÚT-KÖLTSÉG[csomópont] + LÉPES-KÖLTSÉG(csomópont, cselekvés, s)
    MÉLYSÉG[s]  $\leftarrow$  MÉLYSÉG[csomópont] + 1
    s hozzáadása a követők-höz
  return követők
```

**3.9. ábra.** Az általános fakeresési algoritmus. (Vegyük észre, hogy a *perem* argumentumnak egy üres sornak kell lennie, és a sor típusa befolyással lesz a keresés sorrendjére.) A MEGOLDÁS függvény a szílőmutatók gyökérig való követésével kinyert cselekvéssorozatot adja vissza.

## A problémamegoldó hatékonyúság mérése

A problémamegoldó algoritmus kimenete vagy *kudarc*, vagy egy megoldás (egyes algoritmusok végtelen hurokba kerülhetnek és soha nem térnek vissza válasszal). Mi az algoritmusok hatékonyását négyféle módon fogjuk értékelni:

- **Teljesség (completeness):** az algoritmus garantáltan megtalál egy megoldást, amennyiben létezik megoldás?
- **Optimalitás (optimality):** a stratégia megtalálja az optimális megoldást, ahogy azt a 100–101. oldalon definiáltuk?
- **Időigény (time complexity):** mennyi ideig tart egy megoldás megtalálása?
- **Tárigény (space complexity):** a keresés elvégzéséhez mennyi memóriára van szükség?

Az idő- és tárigényről beszélve minden a probléma nehézségének valamelyen mértékét tartjuk szem előtt. Az elméleti számítástudományban egy tipikus ilyen mérték az állapotér gráf nagysága, mivel a gráf a kereső program bemenetére adott explicit adatstruktúrának tekinthető (erre egy példa Románia térképe). Az MI-ben, ahol a gráfot implicit formában a kezdeti állapottal és az állapotátmenet-függvénnyel reprezentáljuk, és ahol a gráf sokszor végtelen, a komplexitást három tényezővel fejezzük ki. Ezek:  $b$  – az elágazási tényező (**branching factor**), vagyis a követők maximális száma minden csomópontban,  $d$  – a leg sekélyebb célállapot mélysége és  $m$  – az állappötörben található utak maximális hossza.

Az időt gyakran a keresés közben generált<sup>5</sup> csomópontok számával, a tárat pedig a memoriában maximálisan tárolt csomópontok számával méri.

A keresés hatékonyágának becslésénél gondolhatunk a keresési költségre (**search cost**), amely tipikusan az időigénytől függ, de tartalmazhat egy tárigény jelző komponenst is; vagy használhatjuk a keresés összköltségét (**total cost**), amely a megoldás útköltségét és a keresési költséget kapcsolja össze. Az Aradról Bukarestbe vezető útkeresési probléma esetén a keresési költség a keresés ideje, a megoldási költség pedig a teljes út hossza km-ben. Így amikor a teljes költséget akarjuk kiszámítani, kilométert és milliszekundumot kellene összeadnunk. Ez nem minden egyszerű, mert nem létezik semmilyen „hivatalos váltószám” a kettő között, ennél a problémánál azonban értelmesnek tűnhet a kilométereket milliszekundumokra átszámítani a gépkocsi átlagsebességét használva (mert az ágensnek az idő a fontos). Ez lehetővé teszi, hogy az ágens megtalálja azt a kompromisszumot, amikor a legrövidebb út keresését célzó minden további számítás improdiktívvá válik. A különböző javak közötti kompromisszum általános problémájára a 16. fejezetben még visszatérünk.

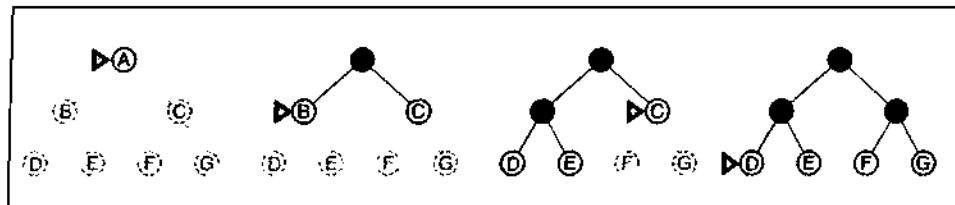
### 3.4. NEM INFORMÁLT KERESÉS

Ez az alfejezet öt keresési stratégiát tárgyal, amelyek a **nem informált** (vaknak is nevezett) keresés (**noninformed (blind) search**) cím alá sorolhatók. A kifejezés azt jelenti, hogy ezen stratégiáknak semmilyen információjuk nincs az állapotokról a probléma definíciójában megadott információin kívül. Működésük során más nem lehetnek, mint a következő állapotok generálása és a célállapot megkülönböztetése a nem célállapottól. Azokat a stratégiákat, amelyek tudják, hogy az egyik közbülső állapot „ígéretesebb”, mint egy másik közbülső állapot, **informált keresési** (**informed search**) vagy **heurisztikus keresési** (**heuristic search**) stratégiának nevezzük. Ezeket majd a 4. fejezet tárgyalja. A keresési stratégiákat a csomópontok kifejtési *sorrendje* különbözteti meg egymástól.

#### Szélességi keresés

A szélességi keresés (**breadth-first search**) egy egyszerű keresési stratégia, ahol először a gyökércsomópontot fejtjük ki, majd a következő lépésekben az összes a gyökércsomópontból generált csomópontot, majd azok követőit stb. Általánosságban a keresési stratégia

<sup>5</sup> Egyes források helyette az időt a kifejtett csomópontok számával méri. A két mérték legfeljebb egy  $b$  tényezőben tér el. Nekünk úgy tűnik, hogy a csomópontkifejtés végrehajtási ideje nő az adott kifejtésben generált csomópontok számával.



3.10. ábra. Szélességi keresés egy egyszerű bináris fában. minden lépésnél a következő kifejtendő csomópontot egy marker jelzi.

minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik, egy szinttel lejebb csomópontot kifejténe.

A szélességi keresést meg lehet valósítani a FA-KERESÉS algoritmussal egy olyan üres peremmel, amely egy először-be-először-ki (first-in-first-out – FIFO) sor, biztosítva ezzel, hogy a korábban meglátogatott csomópontokat az algoritmus korábban fejt ki. Más szóval a FA-KERESÉS(*probléma*, FIFO-SOR()) meghívása szélességi keresést eredményez. A FIFO sor az összes újonnan legenerált követőt a sor végére teszi, magyarán a sekélyebb csomópontok korábban kerülnek kifejtésre, mint a mélyebben fekvők. A 3.10. ábra illusztrálja a keresés előrehaladását egy egyszerű bináris fa esetén.

A szélességi keresés elemzéséhez az előbbi részben tárgyalt négy jellemzőt fogjuk használni. Könnyű belátni, hogy ez a keresés *teljes*. Ha a legsekelyebb célcsomópont valamilyen véges  $d$  mélységen fekszik, a szélességi keresés eljut hozzá az összes nála sekélyebben fekvő csomópontot kifejtve (feltéve persze, hogy a  $b$  elágazási tényező véges). A legsekelyebb célcsomópont nem szükségképpen *optimális*. Pontosabban a szélességi keresés optimális, ha az útköltség a csomópont mélységének nem csökkenő függvénye (például ha minden cselekvésnek ugyanannyi a költsége).

Eddig a szélességi keresésnek csak a jó tulajdonságait láttuk. Ahhoz, hogy megértsük miért nem mindig ezt a stratégiát választjuk, meg kell vizsgálnunk a keresés végrehajtáshoz szükséges idő és memória mennyiségett. Ehhez egy olyan hipotetikus állapotteret veszünk alapul, amelyben minden egyes állapotot kifejtve  $b$  új állapot keletkezik. A keresési fa gyökércsomópontja  $b$  csomópontot generál az első szinten, amelyek mindegyike újabb  $b$  csomópontot, összesen  $b^2$  csomópontot generál a második szinten. Ezek mindegyike újabb  $b$  csomópontot generál, összesen  $b^3$  csomópontot a harmadik szinten és így tovább. Tételezzük fel, hogy ezen probléma megoldása  $d$  mélységen található. Ekkor a legrosszabb esetben a  $d$ -edik szinten az utolsót kivéve (mert a célt magát nem fejtjük ki) a csomópontok mindegyikét ki kell fejtenünk, a  $(d+1)$ -edik szinten  $b^{d+1} - b$  csomópontot generálva. A generált csomópontok összszáma így:

$$b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

Minden legenerált csomópontot a memoriában el kell tárolni, mert vagy a perem eleme, vagy egy perembeli csomópont őse. A tárigény így az időigénytelenséget azonos (meg egy további csomópont a gyökér számára).

Azok, akik járatosak a komplexitáselemzésben, kezdenek aggódni (vagy izgatottak lesznek, ha szeretik a kihívásokat), amikor exponenciális komplexitást látnak, mint amilyen például az  $O(b^{d+1})$ . A 3.11. ábra megmutatja, hogy miért. Az ábra egy  $b = 10$  elágazási tényezővel rendelkező szélességi keresést mutat a  $d$  megoldás több

Mélység	Csomópontok	Időigény	Tárigény
2	1100	0,11 másodperc	1 Mbájt
4	111100	11 másodperc	106 Mbájt
6	$10^7$	19 perc	10 Gbájt
8	$10^9$	31 óra	1 Tbájt
10	$10^{11}$	129 nap	101 Tbájt
12	$10^{13}$	35 év	10 Pbájt
14	$10^{15}$	3523 év	1 Ebájt

**3.11. ábra.** A szélességi keresés idő- és tárigénye. Az ábra adatai  $b = 10$ -es elágazási tényezőt, 10 000 csomópont/percet és 1000 bájt/csomópontot feltételeznek.

értekére. A táblában feltételezzük, hogy másodpercenként 10 000 csomópontot generálunk, illetve egy csomópont tárolásához 1000 bájtra van szükség. Számos feladvány jellegű probléma felel meg ezeknek a feltételezéseknek (egy 100-as tényezővel ide vagy oda), ha azokat modern személyi számítógépeken futtatjuk.

A 3.11. ábra alapján két tanulságot vonhatunk le. Először is a *szélességi keresés esetén a tárigény nagyobb problémát jelent az időigénynél*. A legtöbb ember, amennyiben érdekli a válasz egy fontos problémára, türelmesen ki tud várni 31 órát, hogy egy 8 mélységű keresés lefusson, de csak kevés számítógépnek van a kereséshez szükséges Tbájtnyi memóriája. Szerencsére léteznak ennél kevesebb memóriát igénylő keresési algoritmusok is.

A második tanulság, hogy az időigény még mindig fontos tényező. Ha a problémánk 12 mélységű, akkor (a feltételezések mellett) a szélességi keresésnek (vagy akármilyik nem informált keresési algoritmusnak) 35 évbe telne a megoldás megtalálása. Általábanosságban az exponenciális komplexitású keresési problémák közül csak a legkisebb problémapéldányok oldhatók meg.

## Egyenletes költségű keresés

A szélességi keresés optimális, ha minden lépés költsége azonos, mert minden a legkélyebben fejtett csomópontot fejti ki. Egyszerű általánosítással egy olyan algoritmust találhatunk ki, amely tetszőleges lépésköltség mellett optimális. Az **egyenletes költségű keresés (uniform cost search)** minden a legkisebb útköltségű  $n$  csomópontot fejti ki először, nem pedig a legkisebb mélységű csomópontot. Egyszerűen belátható, hogy a szélességi keresés is egyenletes költségű keresés. Amennyiben minden lépésköltség azonos.

Az egyenletes költségű keresés nem foglalkozik azzal, hogy hány lépésből áll egy bizonyos út, hanem csak az összköltségükkel tömördik. Emiatt minden végtelen hurokba kerül, ha egy csomópont kifejtése zérus költségű cselekvéshez és ugyanahhoz az állapothoz való visszatérést eredményez (például a *NoOp* cselekvés). A teljességet csak úgy garantálhatjuk, hogy minden lépés költsége egy kis pozitív és konstansnál nagyobb, vagy azzal egyenlő. Ez a feltétel egyben az *optimalitás* elégésges feltétele is. Ez azt jelenti, hogy egy út költsége az út mentén mindenkor növekszik. Ebből a tulajdonsságból látszik, hogy az algoritmus a csomópontokat mindenkor növekvő útköltség függ-

vényében fejt ki. Azaz az első kifejtésre kiválasztott célcsomópont egyben az optimális megoldás is (emlékezzünk arra, hogy a FA-KERESÉS a célállapottesztet csak a kifejtésre megválasztott csomópontokra alkalmazza). Javasoljuk, hogy próbálják ki az algoritmust, hogy a Bukarestbe vezető legrövidebb utat megtalálják.

Az egyenletes költségű keresést nem a mélység, hanem az útköltség vezérli, így komplexitását a  $b$  és a  $d$  függvényében nehéz jellemezni. Helyette legyen  $C^*$  az optimális megoldás költsége, és tételezzük fel, hogy minden cselekvés költsége legalább  $\epsilon$ . Az algoritmus idő- és tárígénye legrosszabb esetben  $O(b^{1+|C^*|/\epsilon})$ , ami sokkal több lehet, mint  $b^d$ . Ez azért lehetséges, mert az egyenletes költségű keresés képes (és sokszor ezt meg is teszi) a kis lépésekkel álló nagy fákat felkutatni a nagy és feltehetően hasznos lépésekkel tartalmazó utak előtt. Amikor minden lépés költsége ugyanannyi, az  $(b^{1+|C^*|/\epsilon})$  persze azonos  $b^d$ -nel.

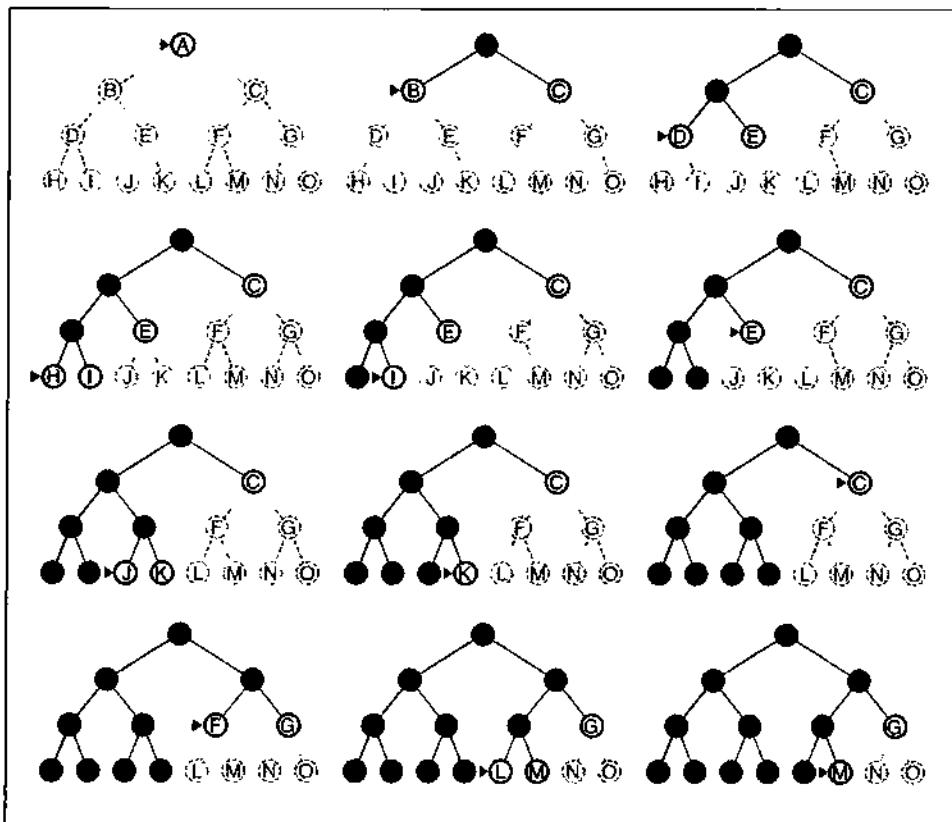
## Mélyiségi keresés

A mélyiségi keresés (*depth-first search*) minden a keresési fa aktuális peremében a *legmélyebben* fekvő csomópontot fejt ki. A keresés lefolyását a 3.12. ábra illusztrálja. A keresés azonnal a fa legmélyebb szintjére jut el, ahol a csomópontoknak már nincsenek követői. Kifejtésüket követően kikerülnek a peremből és a keresés „visszalép” ahhoz a következő legmélyebben fekvő csomóponthoz, amelynek vannak még ki nem fejtett követői.

Ez a stratégia egy olyan FA-KERESÉS függvényteljes implementálható, amelynek a sorbaállító függvénye az utolsónak-be-elsőnek-ki (last-in-first-out, LIFO), más néven verem. A mélyiségi keresést szokás a FA-KERESÉS függvény alternatívájaként egy rekurzív függvényteljes implementálni, amely a gyermekcsomópontokkal meghívja önmagát (mélyésgörbával dolgozó rekurzív mélyiségi keresés algoritmusát a 3.13. ábra mutatja).

A mélyiségi keresés nagyon szerény tárígényű. Csak egyetlen, a gyökér csomóponttól egy levélcsomópontig vezető utat kell tárolnia, kiegészítve az út minden egyes csomópontja mellett a kifejtetlen csomópontokkal. Egy kifejtett csomópont el is hagyható a memoriából, feltéve, hogy az összes leszármazottja meg lett vizsgálva. Egy  $b$  elágazási tényezőjű és  $m$  maximális mélyiségi állapottér esetén a mélyiségi keresés tárígénye  $bm + 1$ . A 3.11. ábra feltételezéseivel élve és feltételezve, hogy a célcsomópont mélyiségi csomópontoknak nincsenek követői, azt találjuk, hogy például  $d = 12$  mélyég esetén a mélyiségi keresés 118 kbájtot igényelne a 10 Pbájtal szemben, ami tízmilliárdos redukciót jelent a tárígényben.

A mélyiségi keresés **visszalépéses keresésnek** (*backtracking search*) nevezett változata még kevesebb memóriát használ. A visszalépéses keresés az összes követő helyett egyidejűleg csak egy követőt generál. minden részben kifejtett csomópont emlékszik, melyik követője jön a legközelebb. Ily módon csak  $O(m)$  memóriára van szükség,  $O(bm)$  helyett. A visszalépéses keresés még egy memória- (és idő-) spóroló trükkhöz folyamodik. Az ötlet a követő csomópont generálása az aktuális állapot *módosításával*, anélkül hogy az állapotot átmásolnánk. Ezzel a memóriaszükséglet egy állapotra és  $O(m)$  cselekvésre redukálódik. Ahhoz, hogy az ötlet működjön, amikor visszalépünk, hogy a következő követőt generáljuk, mindegyik módosítást vissza kell tudnunk csinálni.



**3.12. ábra.** Mélységi keresés egy bináris keresési fában. A kifejtett csomópontok, amelyeknek a peremben nincsenek követőik, el is hagyhatók a memoriából. Ezeket feketével jelöltük meg. A 3-as mélységű csomópontokról feltételezzük, hogy nincsenek követőik, valamint azt is feltehetjük, hogy M az egyetlen célcsomópont.

Nagy állapotterrel rendelkező problémák esetén, mint például robot-összeszerelés esetén, az ilyen módszerek lényegesek a sikerességhöz.

A mélységi keresés hátrányos tulajdonsága, hogy egy rossz választással egy hosszú (akár végtelen) út mentén lefelé elakadhat, miközben például egy más döntés elvezetné a gyökérhez közeli megoldáshoz. A 3.12. ábrán például a mélységi keresés kifejti az egész bal oldali részfát, annak ellenére, hogy a C csomópont a megoldás. Ha a J csomópont szintén megoldás lenne, a mélységi keresés azt adná vissza megoldásul, következetesképpen a mélységi keresés nem optimális. Ha a bal oldali részfa korlátlanul mély lenne és nem tartalmazna megoldást, a mélységi keresés soha nem állna meg, következetesképpen a mélységi keresés nem teljes. A legrosszabb esetben a mélységi keresés a keresési fában az összes  $O(b^m)$  csomópontot generálni fogja, ahol  $m$  a csomópontok maximális mélysége. Jegyezzük meg, hogy  $m$  sokkal nagyobb lehet, mint  $d$  (a leg sekélyebb megoldás mélysége), és korlátlan fák esetén értéke végtelen.

## Mélységekorlátozott keresés

A végtelen fák problémáját a mélységi keresés azáltal küszöböli ki, hogy az utak maximális mélységére egy  $\ell$  korlátot ad. Az  $\ell$  mélységen levő csomópontokat úgy kezeli, mintha nem is lennének követőik. A módszer neve a **mélységekorlátozott keresés, MKK (depth-limited search, DLS)**. A mélységekorlát a végtelen út problémáját ugyan megoldja, de a nemteljesség egy újabb forrását hozza be, ha  $\ell < d$ -t választunk, azaz, ha a legnagyobb célcsomópont a mélységekorláton túl van (ez nem is esélytelen, ha  $d$  eleve ismeretlen). A mélységekorlátozott keresés  $\ell > d$  választással sem lesz optimális. A keresés időigénye  $O(b^\ell)$ , tárigénye  $O(b\ell)$ . A mélységi keresés egy olyan speciális mélységekorlátozott keresésének tekinthető, amelynek mélységekorlátja  $\ell = \infty$ .

A mélységi korlátot néha a probléma ismeretére lehet alapozni. Például Románia térképén 20 város található, így tudjuk, hogy ha létezik egy megoldás, az maximálisan 19 lépés hosszú lehet, így az  $\ell = 19$  egy lehetséges választás. Ha azonban a térképet tüzetesebben tanulmányoznánk, felfedeznénk, hogy minden város bármelyik másik városból legfeljebb 9 lépében elérhető. Ez a szám, amit az állapottér átmérőjének (**diameter**) nevezünk, jobb mélységekorlátot ad, ami hatékonyabb mélységekorlátozott keresést eredményez. A legtöbb probléma esetén azonban mindenkor megoldást adni, amíg meg nem oldottuk a problémát.

A mélységekorlátozott keresést az általános fakeresési vagy a rekurzív mélységi keresési algoritmus egyszerű módosításával lehet implementálni. A 3.13. ábra mutatja a rekurzív mélységekorlátozott keresés pszeudokódját. Jegyezzük meg, hogy a mélységekorlátozott keresés kudarcjal kétféle módon állhat le: a standard *kudarc* csomópont jelzi a megoldás hiányát, a *vágás* érték viszont jelzi a megoldás mélységekorláton belüli hiányát.

```

function MÉLYSÉG-KORLÁTOZOTT-KERESÉS(probléma, korlát) returns egy megoldás vagy kudarc/vágás
return REKURZÍV-MKK(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), probléma, korlát)

function REKURZÍV-MKK(csomópont, probléma, korlát) returns egy megoldás vagy kudarc/vágás
vágás-megírtént? ← hamis
if CÉL-TESZT[probléma](ÁLLAPOT[csomópont]) then return MEGOLDÁS(csomópont)
else if MÉLYSÉG[csomópont] = korlát then return vágás
else for each követő in KIFEJT(csomópont, probléma) do
    eredmény ← REKURZÍV-MKK(követő, probléma, korlát)
    if eredmény = vágás then vágás-megírtént? ← igaz
    else if eredmény ≠ kudarc then return eredmény
    if vágás-megírtént? then return vágás else return kudarc

```

3.13. ábra. A mélységekorlátozott keresés rekurzív implementációja

## Iteratívan mélyülő mélységi keresés

Az **iteratívan mélyülő keresés (iterative deepening search)** – vagy iteratívan mélyülő mélységi keresés – egy általános stratégia, amit sokszor a mélységi kereséssel együtt alkalmaznak a legjobb mélységekorlát megtalálására. Az algoritmus képes erre, mert fokozatosan növeli a mélységekorlátot – legyen az először 0, majd 1, majd 2 stb. – amíg

a célt meg nem találja. Ez akkor következik be, ha a mélységek korlát eléri a  $d$ -t, a legsejélyebben fekvő célcsoportot mélységét. Az algoritmust a 3.14. ábra mutatja. Az iteratívan mélyülő keresés ötvözi a szélességi és a mélységi keresés előnyös tulajdonságait. A mélységi kereséshez hasonlóan szerény, pontosabban  $O(bd)$  memóriaigénytelivel rendelkezik. A szélességi kereséshez hasonlóan teljes, ha elágazási tényezője véges, és optimális, ha az útköltség a csomópontok mélységének nem csökkenő függvénye. A 3.15. ábra az ITERATÍVAN-MÉLYÜLŐ-KERESÉS első négy iterációját mutatja egy bináris fán, ahol az algoritmus a megoldást a negyedik iterációban találja meg.

Az iteratívan mélyülő keresés tékozlónak tűnhet, mert felettesebb sok állapotot többször is kifejt. Kiderül azonban, hogy a költségtöbblet nem lényeges. Ennek az az oka, hogy egy olyan keresési fában, ahol minden szinten ugyanaz (vagy közel ugyanaz) az elágazási tényező, majdnem az összes csomópont a legmélyebb szinten található, így nem túl sokat számít, hogy a magasabb szinteket többször is kifejtjük. Az iteratívan mélyülő keresésben a legmélyebb szinten ( $d$  mélység) található csomópontokat csak egyszer fejtjük ki, egy szinttel feljebb kétszer stb. egészen a gyökér gyerekeiig, amelyeket  $d$ -szer fejtünk ki. Így a kifejtett csomópontok összszáma:

$$Cs(IMK) = (d)b + (d-1)b^2 + \dots + (1)b^d$$

amely  $O(b^d)$  időkomplexitást eredményez. Összehasonlításul nézzük meg a szélességi keresés által generált csomópontok számát:

$$Cs(SZK) = b + b^2 + \dots + b^d + (b^{d+1} - b)$$

Vegyük észre, hogy a szélességi keresés  $d+1$  mélységen is generál csomópontokat, az iteratívan mélyülő kereséssel ellentétben. Ennek eredménye, hogy az iteratívan mélyülő keresés a szélességi keresésnél gyorsabb, annak ellenére, hogy a csomópontokat többször fejt ki. Konkretizálva, például  $b = 10$  és  $d = 5$  esetén ezek a számok:

$$Cs(IMK) = 50 + 400 + 3000 + 20\,000 + 100\,000 = 123\,450$$

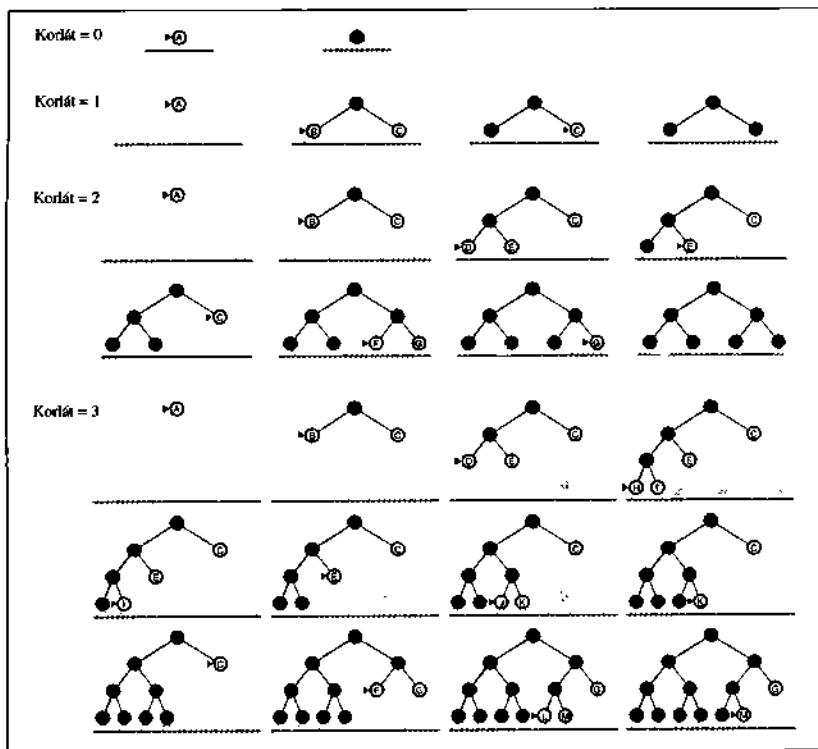
$$Cs(SZK) = 10 + 100 + 1000 + 10\,000 + 100\,000 + 999\,990 = 1\,111\,100$$

 Általánosságban nagy keresési térrrel rendelkező problémák esetén és ha a megoldás mélysége nem ismert, a nem informált módszerek köréből az iteratívan mélyülő keresés a javasolt.

```
function ITERATÍVAN-MÉLYÜLŐ-KERESÉS(probléma) returns egy megoldás vagy kudarc
  inputs: probléma, egy probléma
```

```
  for mélység ← 0 to ∞ do
    eredmény ← MÉLYSÉGKORLÁTOZOTT-KERESÉS(probléma, mélység)
    if eredmény ≠ vágás then return eredmény
```

**3.14. ábra.** Az iteratívan mélyülő keresési algoritmus mélységek korlátozott keresést alkalmaz ismételten, növekvő mélységekkel. Az algoritmus megáll, ha a megoldást megtalálja, vagy ha a mélységek korlátozott keresés kudarccal tér vissza, jelezve, hogy megoldás nem létezik.



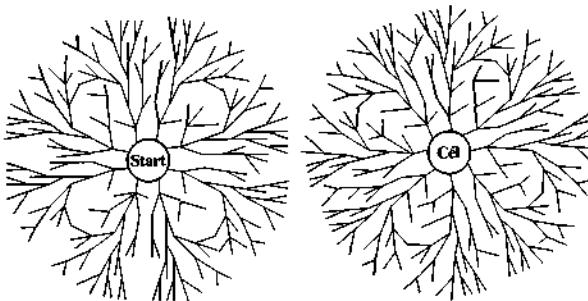
3.15. ábra. Az iteratívan mélyülő keresési algoritmus négy iterációja egy bináris fán

Az iteratívan mélyülő keresés a szélességi kereséssel abban rokon, hogy minden iterációban a csomópontok teljes rétegét megvizsgálja, mielőtt a következő rétegre térne rá. Hasznosnak látszik az egyenletes költségű keresés iteratív változatának kifejlesztése, amely örökölné ez utóbbi optimalitását, mellőzve annak a tárkötetelményeit. Az ötlet a növekvő útköltségek korlát használata a növekvő mélységek korlát helyett. Az eredményül kapott algoritmussal, amelynek neve **iteratívan megnyúló keresés** (**iterative lengthening search**) a 3.11. feladat foglalkozik. Sajnos az derül ki, hogy az iteratívan megnyúló keresés overheadje tekintélyes az egyenletes költségű kereséshez képest.

## Kétirányú keresés

A kétirányú keresést mögött az az ötlet húzódik, hogy egyszerre el lehet indítani egy keresést előrefelé a kiinduló állapotból, illetve hátrafelé a célállapotból, és a keresés akkor fejeződik be, ha a két keresés valahol találkozik (lásd 3.16. ábra). Az érv az, hogy  $b^{d/2} + b^{d/2}$  sokkal kisebb, mint  $b^d$ , illetve az ábrán szemléltve, hogy a két kisebb kör összterülete kisebb, mint annak a nagy körnek a területe, amelynek középpontja a kiinduló állapot, és amely a peremével a célállapotot eléri.

A kétirányú keresést úgy implementálják, hogy az egyik vagy mindkét keresés egy csomópont kifejtése előtt megvizsgálja, hogy az nem része-e a másik keresési fa pere-



**3.16. ábra.** A kétirányú szélességi keresés sematikus ábrája. Az ábrán a két keresési irány majdnem találkozik, amikor a kiinduló csomópontból kinyúl egyik ág összeér egy a célcsomópontból kinyúl másik ággal.

ménék. Ha igen, megvan a cél. Ha a probléma például  $d = 6$  megoldás mélységű, és mindegyik irányban a szélességi keresést futtatjuk csomóponthonként, a két keresés a legrosszabb esetben akkor találkozik, ha mindegyik algoritmus a 3-as mélységben egy csomópont kivételével minden csomópontot kifejtett. A  $b = 10$  esetén ez 22 200 csomópont generálását jelenti a standard szélességi keresés által generált 1 111 111 csomóponthoz képest. Annak ellenőrzését, hogy egy csomópont a másik keresési fához tartozik-e, egy hash-táblával konstans időben meg lehet oldani. A kétirányú keresés időkomplexitása így  $O(b^{d/2})$ . Legalább az egyik keresési fát a memoriában kell tartani, hogy a tartozás ellenőrzése kivitelezhető legyen, a tárkomplexitás tehát szintén  $O(b^{d/2})$ . Ez a tárkomplexitás a kétirányú keresés legnagyobb gyengéje. Ha minden két keresés szélességi keresés, az algoritmus teljes és optimális (egyenletes költség esetén). Más módszerek kombinációja vagy a teljesség, vagy az optimalitás, vagy mindenkor elvészítéséhez vezethet.

Az időkomplexitás mérséklése a kétirányú keresést igen vonzóvá teszi, de mit is jelent a célállapotból hátrafelé keresni? Ez nem is olyan egyszerű, mint amilyennek hangzik. Legyenek  $x$  csomópont elődcsomópontjai (*predecessors*), az  $Előd(x)$ -ek azon csomópontok, amelyek mindegyikének  $x$  a követő csomópontja. A kétirányú keresés feltételezi, hogy egy  $Előd(x)$  hatékonyan számítható. Legegyszerűbb az az eset, amikor az összes cselekvés az állapottérben reverzibilis, így  $Előd(x) = Követő(x)$ . Más esetben azonban igen nagy ötletességre lehet szükség.

Nézzük most, hogy mit is jelent az, hogy „cél”, ha a célállapotból hátrafelé kell keresni. A 8-as kirakójáték és a romániai útkeresés esetén egyetlenegy célállapot létezik csak, így a hátrafelé keresés és az előrefelé keresés igen hasonlítanak egymásra. Amennyiben létezik a célállapotoknak egy *explicit listája*, mint például a 3.3. ábra két koszmentes célállapota, akkor megkonstruálhatunk egy olyan által-célállapotot, amelynek közvetlen követői az aktuális célállapotok. Más módon, néhány redundáns csomópont-generálás elkerülhető azzal, hogy a célállapotok halmazát egyetlenegy célállapotnak tekintjük, amelynek minden elődje szintén egy állapothalmaz – konkréten azon állapotok halmaza, amelyek követője a célállapothalmaz eleme (lásd még 3.6. alfejezet).

A kétirányú keresés szempontjából a legnehezebb eset, amikor a célállapottesztnél a feltehetően nagy célállapothalmazról csak implicit leírás áll rendelkezésünkre, például a sakkban az összes állapot, ami kielégíti a „matt” célt. A hátrafelé keresésnek

az „ $m_1$  cselekvés révén a »matt« állapotba vezető összes állapot” tömör leírását kellene tudnia megkonstruálni, és hasonló módon folytatni. E leírásokat az előrefelé haladó keresés által generált állapotokkal kellene tesztelni. Ennek nincs általánosan hatékony módja.

## A keresési stratégiák összehasonlítása

A 3.17. ábra a 3.4. alfejezetben megfogalmazott négy kiértékelési kritérium tükrében összehasonlítja a keresési stratégiákat.

Kritérium	Szélességi	Egyenletes költségű	Mélységi korlátozott	Mélység-mélyülő	Iteratív	Kétirányú (amennyiben alkalmazható)
Teljes?	Igen <sup>a</sup>	Igen <sup>a, b</sup>	Nem	Nem	Igen <sup>a</sup>	Igen <sup>a, d</sup>
Időigény	$O(b^{d+1})$	$O(b^{l+\lfloor C/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Tárigény	$O(b^{d+1})$	$O(b^{l+\lfloor C/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimális?	Igen <sup>c</sup>	Igen	Nem	Nem	Igen <sup>c</sup>	Igen <sup>c, d</sup>

3.17. ábra. A keresési stratégiák értékelése.  $b$  az elágazási tényező,  $d$  a legsekelyebb megoldás mélysége,  $m$  a keresési fa maximális mélysége,  $l$  a mélységlimit. A felső indexszel jelzett kikötések a következők: <sup>a</sup> teljes, ha  $b$  véges; <sup>b</sup> teljes, ha a lépésköltség  $\geq \epsilon$ , pozitív  $\epsilon$ -ra; <sup>c</sup> optimális, ha a lépésköltségek mindenazonosak; <sup>d</sup> ha minden irányban szélességi keresést használunk.

## 3.5. AZ ISMÉTELT ÁLLAPOTOK ELKERÜLÉSE

Eddig figyelmen kívül hagytuk a keresés egyik legfontosabb megoldandó problémáját: a már korábban egy másik úton megtalált és kifejtett állapotok ismételt kifejtéséből adódó időpazarlást. Néhány probléma esetén ezen lehetőség soha nem merül fel, mert az állapottér egy fa, és minden állapotba csak egyetlen módon lehet eljutni. A 8-királynő probléma hatékony megfogalmazása – amikor minden új királynőt a bal szélső szabad oszlopba helyezzük – nagyrészt épp ennek köszönheti hatékonyságát, vagyis hogy minden egyes állapotba csak egyetlen úton lehet eljutni. Ha a 8-királynő problémát úgy fogalmazzuk meg, hogy egy királynőt báramelyik oszlopból el lehet helyezni, akkor az  $n$  királynőt tartalmazó minden állapotot  $n!$  különböző úton el lehet érni.

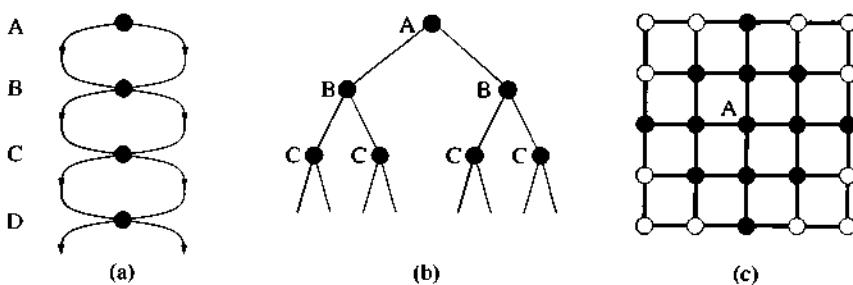
Számos problémánál azonban elkerülhetetlenek a megismételt állapotok. Ezek közé tartozik az összes olyan probléma, amelyben az operátorok reverzibilisek. Többek között ebbé a csoportba tartoznak az útkeresési problémák és a csúszó lapka fejtőrő játékok. Az ezen problémákhoz tartozó keresési fák végtelenek, de ha a megismételt állapotok egy részét levágjuk, akkor a keresési fát véges méretűre vághatjuk, ezáltal a keresési fának csak az állapottér gráfot kifeszítő részét generálva. Szélsőséges esetben egy  $d + 1$  méretű állapottérrel (lásd 3.18. (a) ábra) egy  $2^d$  levelű fa lesz (lásd 3.18. (b) ábra). Egy élethűbb példa a négyzetrács (rectangular grid) (lásd 3.18. (c) ábra). A rácson minden állapotnak négy követője van. Az ismétlődő állapotokat tartalmazó fának így  $4^d$  levele van. minden adott állapottól  $d$  lépéshire azonban csak kb.  $2d^2$  különböző állapot van.  $d = 20$  esetén ez kb. egybillió csomópontot, de csak 800 különböző állapotot jelent.

Az ismétlődő állapotok tehát a megoldható problémákat megoldhatatlan problémákká alakítják, amennyiben az algoritmus nem képes ezeket az állapotokat detektálni. A detektálás általában azt jelenti, hogy az új kifejtendő csomópontot a már kifejtett csomópontokkal hasonlítjuk össze. Egyezés esetén az adott csomóponthoz az algoritmus két utat talált, és valamelyiket eldobhatja.

A mélységi keresés csak a gyökeret és az aktuális csomóponttal összekötő úton fekvő állapotokat tárolja. E csomópontoknak az aktuális csomóponttal való összehasonlítása lehetővé teszi a hurkok felismerését, amiket ezek után azonnal el lehet dobni. Ez meg is véd attól, hogy a hurkok révén véges állapotterekből ne keletkezzenek végtelen keresési fák. Sajnos azonban nem tud megvédeni a nem hurkos utak exponenciális megsokszorozódásától az olyan problémákban, mint amilyenek a 3.18. ábrán látható. Ezt elkerülni csak úgy lehet, hogy több csomópontot tartunk a memóriában. Az idő és a tárigény között egy alapvető kompromisszum létezik. *Az az algoritmus, amely elfelejtja a történetét, kénytelen azt megismételni.*

Ha egy algoritmus minden meglátogatott állapotra emlékszik, akkor közvetlen módon fedez fel az állapotteret. A FA-KERESÉS algoritmust módosíthatjuk úgy, hogy egy zárt listának (*closed list*) nevezett adatszerkezetet tartalmazzon, amely minden kifejtett csomópontot tárol. (A még ki nem fejtett csomópontkból álló peremet néha nyitott listának [*open list*] nevezik.) Ha az aktuális állapot egybeesik a zárt listán lévő állapotok egyikével, akkor eldobható, ahelyett hogy a kifejtésével kellene foglalkozni. Az új algoritmus neve GRÁF-KERESÉS (3.19. ábra). Az ismétlődő állapotokat tartalmazó problémák esetén a GRÁF-KERESÉS sokkal hatékonyabb, mint a FA-KERESÉS. Az algoritmus a legrosszabb esetre számított tár- és időkomplexitása arányos az állapotter méretével, ami sokkal kisebb lehet, mint  $O(b^d)$ .

A gráfkeresés optimalitása már trükkös dolog. Korábban azt mondta, hogy egy ismételt állapot detektálásával az algoritmus két utat talált ugyanahhoz az állapothoz. A 3.19. ábrán látható GRÁF-KERESÉS algoritmus minden az újonnan felfedezett utat dobja el. Nyilvánvaló, hogy ha az újonnan felfedezett út rövidebb, mint az eredeti, a GRÁF-KERESÉS elvéhet egy optimális megoldást. Szerencsére ki tudjuk mutatni (lásd 3.12. feladat), hogy az egyenletes költségű és a konstans lépésköltségű szélességi keresés esetén ez nem történhet meg. E két optimális fakeresési stratégia egyben optimális



**3.18. ábra.** Egy exponenciálisan nagyobb keresési fát generáló állapotter. (a) Egy olyan állapotter, amelyben két lehetséges cselekvés vezethet A-ból B-be, kettő B-ből C-be és így tovább. Ez az állapotter  $d+1$  állapotot tartalmaz, ahol  $d$  a maximális mélység. (b) Az (a) állapotter megfelelő keresési fája,  $2^d$ -gal, amely az állapotter  $2^d$  útjának felel meg. (c) Egy négyzetrács. A kezdeti (A) állapottól kétlépényire fekvő állapotokat szürke színnel jelöltük.

```

function GRÁF-KERESÉS(probléma, perem) returns egy megoldás vagy kudarc
    zárt lista  $\leftarrow$  egy üres halmaz
    perem  $\leftarrow$  BESZÚR(CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma]), perem)
    loop do
        if ÜRES?(perem) then return kudarc
        csomópont  $\leftarrow$  VEDD-AZ-ELŐ-ELEMET(perem)
        if CÉL-TESTZI(probléma)[ÁLLAPOT[csomópont]] then return MEGOLDÁS(csomópont)
        if ÁLLAPOT[csomópont] nem eleme a zárt listának then
            adjuk hozzá az ÁLLAPOT[csomópont]-ot a zárt listához
        perem  $\leftarrow$  BESZÚR-MIND(KIFEJT(csomópont, probléma), perem)

```

**3.19. ábra.** Az általános gráfkereső algoritmus. A zárt halmazt egy hash-táblával lehet implementálni, hogy az ismételt állapotokat hatékonyan ellenőrizni tudjuk. Ez az algoritmus feltételezi, hogy az s állapothoz vezető első út a legolcsóbb (lásd szöveg).

gráfkeresési stratégia is. Az iteratívan mélyülő keresés azonban mélységi kifejtést használ és könnyen kerülhet egy csomópont felé vezető szuboptimális útra, mielőtt megtalálná a hozzá vezető optimális utat. Az iteratívan mélyülő gráfkeresésnél tehát egy újonnan felfedezett útnál meg kell vizsgálni, hogy ez nem jobb-e, mint az eredeti. És ha igen, szükség lehet az adott csomópont követőinél a mélységek és az útköltségek revíziójára.

Jegyezzük meg, hogy a zárt lista használata azt jelenti, hogy a mélységi és az iteratívan mélyülő keresés tárígénye már nem lineáris. Mivel a GRÁF-KERESÉS algoritmus minden csomópontot a memóriában megtart, egyes keresési fajták kivitelezhetetlenek a memóriakorlátok miatt.

## 3.6. KERESÉS RÉSZLEGES INFORMÁCIÓ MELLETT

A 3.3. alfejezetben azt tételeztük fel, hogy a környezet teljesen megfigyelhető és determinisztikus, valamint hogy az ágens tisztában van minden cselekvésének következményével. Ebből kifolyólag az ágens pontosan ki tudja számítani, hogy egy tetszőleges cselekvésszekvenciának milyen állapot az eredménye, és azt is tudja, hogy ö maga melyik állapotban van. Az érzékelései nem nyújtanak új információt az egyes cselekvések után. Mi történik, ha a cselekvésekéről és az állapotokról alkotott tudás nem teljes? Azt találjuk, hogy a nemteljesség különböző formái három elkülönülő problémápuszthoz vezetnek:

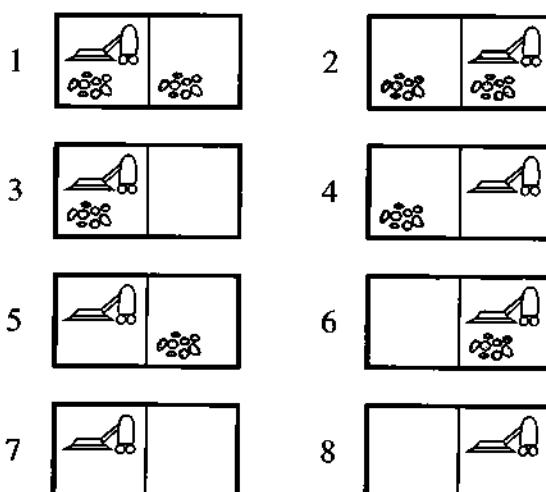
- 1. Szenzor nélküli (sensorless vagy conformant) problémák:** ha az ágensnek egyáltalán nincsenek szenzorai, akkor (a tudásának megfelelően) a több lehetséges kezdeti állapot egyikében lehet, és minden cselekvése a lehetséges követő állapotok egyikéhez vezethet.
- 2. Eshetőségi (contingency) problémák:** ha a környezet részben megfigyelhető, vagy ha a cselekvések bizonytalanok, akkor az ágens érzékszervei új információt nyújtanak minden cselekvés után. minden lehetséges érzékelés egy eshetőséget határoz meg, amire az ágensnek terveznie kell. A problémát ellenfél problémának (*adversarial*) hívják, ha a bizonytalanságot egy másik ágens cselekvései okozzák.

3. **Felfedezéses (explorational) probléma:** ha a környezet állapotai és cselekvései nem ismertek, az ágensnek külön cselekednie kell, hogy azokat felderíthesse. A felfedezéses problémákat az eshetőségi problémák szélsőséges változatának lehet tekinteni.

Példaként a porszívóvilág környezetét vessük. Emlékezzünk vissza, hogy az állapottér nyolc állapotból áll (lásd 3.20. ábra). Három cselekvés van – *Jobbra*, *Balra* és *Szív* –, és a cél az összes kosz felszedése (a 7. és 8. állapot). Ha a környezet megfigyelhető, determinisztikus és teljesen ismert, akkor a problémát triviálisan bármelyik ismeretettségű algoritmussal meg lehet oldani. Így például ha a kezdeti állapot az 5. állapot, akkor a [*Jobbra*, *Szív*] cselekvésszekvencia a 8. célállapotot éri el. A fejezet további részében e probléma szenzor nélküli és eshetőségi változatával foglalkozunk. A felfedezéses problémákat a 4.5. alfejezet, az ellenfélproblémákat a 6. fejezet tárgyalja.

## Szenzor nélküli problémák

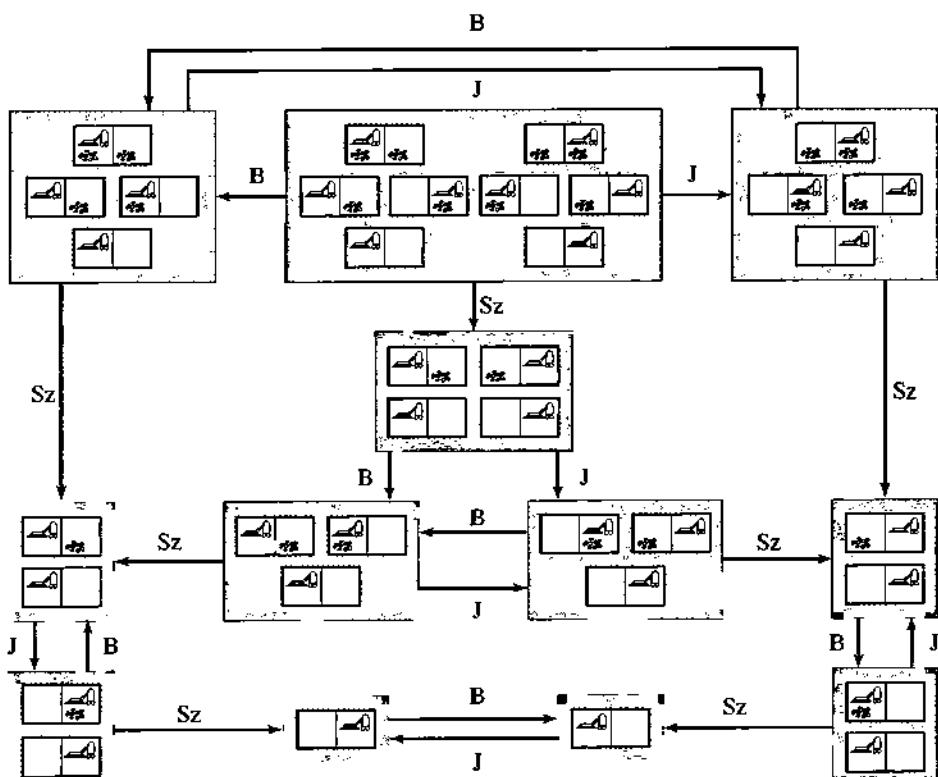
Tegyük fel, hogy a porszívóagens ismeri az összes cselekvésének a hatását, de szenzorokkal nem rendelkezik. Ekkor csak azt tudja, hogy kezdeti állapota az {1, 2, 3, 4, 5, 6, 7, 8} halmaz egyik eleme. Azt lehetne gondolni, hogy az ágens sorsa reménytelen, azonban ágensünk igen jól feltalálja magát. Mivel tudja, hogy cselekvései mire képesek, kiszámíthatja például, hogy a *Jobbra* cselekvés a {2, 4, 6, 8} állapotok egyikébe viszi, és a *Jobbra*, *Szív* cselekvéssorozat eredményeképpen a {4, 8} egyikében köt ki. Végül a *Jobbra*, *Szív*, *Balra*, *Szív* sorozat garantáltan eléri a 7. célállapotot, függetlenül a kezdeti állapottól. Azt mondjuk, hogy az ágens a világot a 7. állapotba bele tudja kényszeríteni (*coerce*), akkor is, amikor nem is tudja, honnan indul. Összegezve: ha a világ nem teljesen megfigyelhető, az ágensnek az egyedi állapotokkal szemben azon állapothalmazokra kell tudnia következtetni, amely halmazok állapotaiba eljuthat.



3.20. ábra. A porszívóvilág nyolc lehetséges állapota

Minden ilyen állapothalmazt egy **hiedelmi állapotnak** nevezünk (**belief state**), amely az ágens pillanatnyi hiedelmét fejezi ki, hogy ő maga vajon milyen fizikai állapotban lehet most. (A teljesen megfigyelhető környezetben minden hiedelmi állapot egyetlen fizikai állapotból áll.)

Ahhoz, hogy egy szenzor nélküli problémát megoldhassunk, a fizikai állapotok terében történő keresés helyett a hiedelmi állapotok terében kell keresni. A kezdeti állapot egy hiedelmi állapot, és minden cselekvés egy hiedelmi állapotból egy hiedelmi állapotba képezi le. Egy cselekvést úgy kell alkalmazni egy hiedelmi állapotra, hogy a hozzá tartozó minden fizikai állapotra kell azt alkalmazni, és az eredmények unióját kell képezni. Egy út most több hiedelmi állapotot köt össze, a megoldás pedig egy olyan utat jelent, amely csak célállapotkból álló hiedelmi állapotba vezet. A 3.21. ábra mutatja a determinisztikus szenzor nélküli porszívóvilág hozzáférhető hiedelmi állapoterét. Csak 12 hozzáférhető hiedelmi állapot létezik, az egész hiedelmi állapottér azonban tartalmazza a fizikai állapotok minden lehetséges halmazát, azaz  $2^8 = 256$  hiedelmi állapotot. Általában, ha a fizikai állapottérnek  $S$  állapota van, a hiedelmi állapottér állapotainak száma  $2^S$ .



**3.21. ábra.** A determinisztikus, szenzor nélküli porszívóvilág hiedelmi állapoterének hozzáférhető része. minden árnyalt doboz egyetlenegy hiedelmi állaphoz tartozik. minden pillanatban az ágens egy konkrét hiedelmi állaphoz tartozkodik, azonban nem tudja, hogy melyik fizikai állapotban van. A kezdeti hiedelmi állapot (a teljes tudatlanság) legfelül középen helyezkedik el. A cselekvéseket a címkézett nyílak jelölik. Az önhurkokat a nagyobb áttekinthetőség érdekében az ábrából kihagytuk.

A szenzor nélküli problémák eddigi elemzésében determinisztikus cselekvéseket tételeztünk fel. Az elemzés lényegében érvényes marad, ha a környezet nemdeterminisztikus, azaz ha a cselekvéseknek néhány lehetséges kimenetele lehet. Ennek az az oka, hogy szenzorok nélkül az ágens nem képes megmondani, hogy milyen kimenetel történt valójában. A különböző lehetséges kimenetelek így pótlólagos fizikai állapotok a követő hiedelmi állapotban. Tételezzük fel például, hogy a környezetben Murphy-törvény uralkodik: a Szív cselekvés *néha* koszt is hagy a szönyegen, *de csak akkor, ha ott előzetesen a kosznak nyoma sem volt*.<sup>6</sup> Ha a Szív cselekvést tehát a 4. fizikai állapotban alkalmazzuk (lásd 3.20. ábra), a két lehetséges kimenetel a 2. és a 4. állapot. A kezdeti {1, 2, 3, 4, 5, 6, 7, 8} hiedelemállapotra alkalmazva a Szív olyan hiedelemállapothoz vezet, ami a cselekvés eredményeinek uniója a nyolc fizikai állapotra. Kiszámítva, azt kapjuk, hogy az új hiedelmi állapot az {1, 2, 3, 4, 5, 6, 7, 8}. A Szív cselekvés a Murphy-törvény uralta világban egy szenzor nélküli ágens hiedelmi állapotát tehát nem változtatja meg! A probléma valójában megoldhatatlan (lásd 3.18. feladat). Ennek intuitív magyarázata az, hogy az ágens nem képes megmondani, hogy az aktuális mező koszos-e, ebből kifolyólag nem képes megmondani, hogy a Szív cselekvés takarítani fog-e, vagy még több koszt hagy maga után.

## Eshetőségi problémák

Ha a környezet jellege olyan, hogy az ágens a cselekvéseit követően szerezhet új információt a szenzorai révén, **eshetőségi problémáról** (*contingency problem*) beszélünk. Egy eshetőségi probléma megoldása sokszor egy *fa* alakját veszi fel, ahol egy-egy ágnak a kiválasztása azon múlik, hogy odáig az ágens milyen érzékelésekhez jutott. Tegyük fel, hogy az ágens a Murphy-törvény világában rendelkezik helyzetérzékelővel és lokális koszérzékelővel, nincs azonban olyan szenzora, amely képes lenne koszt detektálni más négyzeteken. Így a [B, Kosz] érzékelés azt jelenti, hogy az ágens az {1, 3} állapotok egyikében van. Az ágens most a [Szív, Jobbra, Szív] cselekvési szekvenciát fogalmazhatta meg. A szívó cselekvés az {5, 7} állapotok egyikébe viszi az ágenst, és jobbra elmozdulva az állapota a {6, 8} állapotok egyikére változik. A szívó cselekvés alkalmazása a 6. állapotban, a 8. állapotot – a célt – eredményezi. Ha azonban a szívó cselekvést a 8. állapotban alkalmazzuk, a 6. állapotba térünk vissza (a Murphy-törvény jóvoltából), amely esetben tervünk kudarcba fullad.

A hiedelmi teret a probléma e változatánál elemezve könnyen megállapíthatjuk, hogy semmilyen rögzített cselekvésszekvencia sem garantálja a probléma megoldását. De lesz megoldásunk, ha a cselekvések *szigorú sorrendjéről* lemondunk:

[Szív, Jobbra, ha [Jobbra, Kosz] akkor Szív]

A megoldások terét így azzal a lehetőséggel bővíjtük, hogy a cselekvéseket a végre-hajtás alatt megjelenő eshetőségek alapján választjuk. A valós fizikai világ számos problémája eshetőségi probléma, hiszen pontos előrejelzés lehetetlen. Sok ember ezért tartja mindenig nyitva a szemét séta és vezetés közben.

<sup>6</sup> Feltételezzük, hogy már minden olvasó kertült hasonló helyzetbe, így át tudják érezni ágensünk frusztráltságát. Elnézést kérünk a modern háztartási gépek tulajdonosaitól, akiknek ez a pedagógiai példa nem jelent segítséget.

Az eshetőségi problémák *néha* rögzített cselekvésszekvenciával is megoldhatók. Tekintsük például a *teljesen megfigyelhető* Murphy-törvény világát. Eshetőséggel áltunk szemben, ha az ágens a Szív cselekvést egy üres négyzet felett hajtja végre, hiszen a kosz akkor vagy meg fog jelenni, vagy sem. Ameddig az ágens soha nem teszi ezt, eshetőségek fel sem merülnek, és minden kezdeti állapotból létezik egy rögzített megoldási szekvencia (lásd 3.18. feladat).

Eshetőségi problémák esetén az algoritmusok az e fejezetben található standard keresési algoritmusoknál bonyolultabbak. Ezekkel a 12. fejezet foglalkozik. Az eshetőségi problémák egy kissé eltérő ágens tervezési sémát is támogatnak, amelyekben az ágens egy garantált terv megtalálása *előtt* is cselekedhet. Ez hasznos, mert ahelyett, hogy a végrehajtás során *előforduló* minden eshetőséget előre figyelembe venne, gyakran jobb elkezdeni a végrehajtást, és megnézni, hogy mely eshetőségek következnek be valójában. A járulékos információ megadása után az ágens folytathatja a probléma megoldását. A keresés és a végrehajtás íly módon történő összefűlése (*interleaving*) a felderítéses problémák (lásd 4.5. alfejezet) vagy a játékok (6. fejezet) esetén is hasznos.

## 3.7. ÖSSZEFOLGLALÁS

Ez a fejezet olyan módszereket ismertetett, amelyekkel az ágens a cselekvéseit választ-haja meg determinisztikus, megfigyelhető, statikus és teljesen ismert környezetben. Ilyen esetekben az ágens olyan cselekvéssorozatot konstruálhat, amelyekkel a céljait elérheti. Ezt a folyamatot keresésnek (*search*) nevezzük.

- A megoldás keresésének megkezdése előtt az ágensnek előbb meg kell fogalmaznia a célt (*goal*), majd a célt fel kell használnia a probléma megfogalmazására.
- Egy probléma (*problem*) négy részből: a kiinduló állapotból (*initial state*), egy cselekvés- (*action*) készletből, egy céteszt- (*goal test*) függvényből és egy útkölt-ség- (*path cost*) függvényből áll. A probléma környezetét az állapottér (*state space*) írja le. Az állapottérben a kiinduló állapotból a célállapotba vezető út (*path*) a probléma egy **megoldása** (*solution*).
- Egyetlen általános FA-KERÉSÉS algoritmus felhasználható bármilyen probléma megoldására. Az algoritmus konkrét változatai eltérő stratégiákat valósítanak meg.
- A keresési algoritmusokat azok **teljessége** (*completeness*), **optimalitása** (*optimality*), **időigénye** (*time complexity*) és **tárigénye** (*space complexity*) alapján értékeljük. A komplexitást a probléma  $b$  elágazási tényezője és a leg sekélyebben fekvő megoldás  $d$  mélysége határozza meg.
- A **szélességi keresés** (*breadth-first search*) először a leg sekélyebben fekvő csomópontot fejti ki a keresési fában. Ez a keresés teljes, azonos költségű operátorok esetén optimális és  $O(b^d)$  idő- és tárigénnel rendelkezik. Tárigénye miatt a legtöbb esetben nem célszerű alkalmazni. Az **egyenletes költségű keresés** (*uniform-cost search*) hasonló a szélességi kereséshez, de először a legkisebb  $g(n)$  útköltségű csomópontot fejti ki. Teljes és optimális, ha minden lépés költsége valamelyen pozitív  $\epsilon$  korlátnál nagyobb.
- A **mélységi keresés** (*depth-first search*) először a keresési fa legményebben fekvő csomópontját fejti ki. Se nem teljes, se nem optimális, és időigénye  $O(b^m)$ , tárigénye pedig  $O(bm)$ , ahol  $m$  az állapottérbeli utak maximális mélysége.

- A **mélységláthatatlan keresés (depth-limited search)** egy korlátot ad a mélységi keresés keresési mélységére.
- Az **iteratívan mélyülő keresés (iterative deepening search)** a cél megtalálásáig egyre növekvő mélységláthatattal hívja meg a mélységláthatatlan keresést. Teljes és egységnyi lépésköltség esetén optimális, időigénye  $O(b^d)$ , tárigénye pedig  $O(bd)$ .
- A **kétirányú keresés (bidirectional search)** radikálisan csökkentheti az időigényt, de nem minden alkalmasztó, és a tárigénye túl nagy lehet.
- Ha az állapotér inkább egy gráf, mint egy fa, kifizetődő lehet az ismétlődő állapotokat megvizsgálni. A **GRÁF-KERESÉS** algoritmus az összes duplikált állapotot ki-küszöböli.
- Ha a környezet csak részben megfigyelhető, az ágens a keresési algoritmust a **hiedelmi állapotok (belief states)** terében alkalmazhatja, vagyis olyan lehetséges állapotok terében, amelyekben az ágens tartózkodhat. Egyes esetekben egyetlen megoldási szekvencia konstruálható, más esetekben az ágensnek **eshetőségi tervre (contingency plan)** van szüksége, hogy a felmerülő ismeretlen helyzetekkel megbirkózzon.

## Irodalmi és történeti megjegyzések

Az ebben a fejezetben elemzett állappotter-keresési problémák többségének hosszú története van az irodalomban, és e problémák sokkal kevésbé triviálisak, mint azt első ránézésre gondolnánk. A 3.9. feladatban említett hittérítők és kannibálok problémáját Amarel elemzte részletesen (Amarel, 1968). A mesterséges intelligencia területén Simon és Newell (Simon és Newell, 1961), míg a számítástudomány és az operációkutatás területén Bellman és Dreyfus már korábban foglalkoztak e problémával (Bellman és Dreyfus, 1962). Az ilyen tanulmányok és Simon és Newell Logic Theoristra vonatkozó munkája (Simon és Newell, 1957), valamint a GPS (Simon és Newell, 1962) hatására váltak a keresési algoritmusok az 1960-as években az MI-kutatók fegyvertárának elsődleges eszközeivé, illetve ezek hatására vált a problémamegoldás kanonikus MI-feladattá. Sajnos eddig nagyon kevés történt a problémamegfogalmazás lépéseinek automatizálása irányában. A problémareprezentáció és -absztrakció frissebb tárgyalását Knoblock adta meg (Knoblock, 1990), beleértve olyan MI-programok leírását is, amelyek (részben) maguk végzik el ezeket a feladatokat.

A 8-as kirakójáték a bonyolultabb 15-ös kirakójáték kisebb rokona, amit a híres amerikai játéktervező, Sam Loyd talált ki az 1870-es években (Loyd, 1959). A játék akkoriban gyorsan óriási, Rubik bűvös kockájához mérhető népszerűségre tett szert Amerikában. A matematikusok is felfigyeltek a problémára (Johnson és Story, 1879; Tait, 1880). Az *American Journal of Mathematics* szerkesztői megállapították: „A 15-ös kirakójáték az elmúlt hetekben teljesen lebilincselte az amerikai embereket – 10 emberből 9 nemre, korra és származásra való tekintet nélküli a játék bűvöletébe került. Ez azonban még nem hatott volna a szerkesztőkre olyan mértékben, hogy rávegye őket, erről a témaáról cikkeket jelentessének meg az *American Journal of Mathematics* folyóiratban, de figyelembe véve azt a tényt, hogy...” (itt a 15-ös kirakójáték által felvetett érdekes matematikai kérdések rövid taglalása következett). A 8-as játék esetén P. D. A. Schofield számítógép segítségével elvégezte a probléma kimerítő elemzését (Schofield, 1967). Ratner és Warmuth megmutatta, hogy a legrövidebb megoldás megkeresése

a 15-ös játék  $n \times n$ -es általánosításánál az NP-teljes problémák osztályába tartozik (Ratner és Warmuth, 1986).

A 8-királynő problémát először 1848-ban a *Schach* német sakkszaklapban névtelenül publikálták, később a problémát Max Bezzel nevéhez kapcsolták. A problémát 1850-ben újra publikálták, ami akkor felkeltette a kiváló matematikus, Karl Friedrich Gauss figyelmét, aki megkísérelte felsorolni az összes megoldást, azonban csak 72 megoldást talált meg. Az összes, 92 megoldást Nauck publikálta 1850-ben. A problémát Netto  $n$  királynő esetére általánosította (Netto, 1901), Abramson és Young pedig találtak egy  $O(n)$  algoritmust (Abramson és Young, 1989).

A fejezetben említett valósvilág-beli keresési problémák közül mindegyik komoly kutatási erőfeszítések tárgya volt. Az optimális repülőjárat megválasztásának módszereit általában bizalmasan kezelik, azonban Carl de Marcken kimutatta (magánközlés), hogy a repülőtársaságok által alkalmazott jegytarifák és egyéb korlátozások annyira összekuszálódtak, hogy az optimális járat megválasztása formálisan *eldönthetetlen*. Az elméleti számítástudományban standard kombinatorikus probléma az utazó ügynök problémája (TSP) (Lawler, 1985; Lawler és társai, 1992). A problémáról Karp bizonyította be, hogy NP-nehéz (Karp, 1972), azonban megoldására hatékony heurisztikus közelítő módszereket dolgoztak ki (Lin és Kernighan, 1973). Az euklideszi térben zajló TSP-re Arora dolgozta ki a teljesen polinomiális közelítő sémát (Arora, 1998). A VLSI-elrendezés módszereit Shahookar és Mazumder (Shahookar és Mazumder, 1991) tekintették át, számos optimalizációs cikk VLSI-újságokban jelent meg. Robotnavigálási és szerelési problémákkal a 25. fejezet foglalkozik.

A problémamegoldás nem informált keresési algoritmusai központi szerepet töltnek be a klasszikus számításelméletben (Horowitz és Sahni, 1978), az operációkutatásban (Dreyfus, 1969). Deo és Pang, továbbá Gallo és Pallottino egy frissebb áttekintést adnak erről a területről (Deo és Pang, 1984; Gallo és Pallottino, 1988). A szélességi keresést labirintusok megoldására Moore (Moore, 1959) fogalmazta meg. A **dinamikus programozás** (**dynamic programming**) módszere (Bellman és Dreyfus, 1962), amely szisztematikusan jegyzi meg az összes növekvő nagyságú részprobléma megoldását, egy gráfokon futó szélességi keresésnek tekinthető. A Dijkstra-féle képpontos legrövidebb út algoritmus (Dijkstra, 1959) az egyenletes költségű keresés kiindulópontja.

Az iteratívan mélyülő algoritmus egy változatát Slate és Atkin használta először a CHESS 4.5 sakkszámítóban (Slate és Atkin, 1977), hogy a sakkörát hatékonyabban lehessen használni. De az ötletet gráfokban a legrövidebb út keresésére először Korf használta (Korf, 1985a). A Pohl bevezette kétirányú keresés (Pohl, 1969; 1971) bizonyos esetekben szintén igen hatékony lehet.

A részben megfigyelhető és a nemdeterminisztikus környezeteket a problémamegoldási megközelítésen belül nem tanulmányozták behatóan. A hiedelmi állapotok terében történő keresés bizonyos hatékonysági problémáival Genesereth és Nourbakhsh foglalkoztak (Genesereth és Nourbakhsh, 1993). Koenig és Simmons (Koenig és Simmons, 1998) az ismeretlen kezdeti pozícióból induló robotnavigálást tanulmányozták. Erdmann és Mason (Erdmann és Mason, 1988) pedig a szenzorok nélküli robot manipuláció problémáját vizsgálták a hiedelem állapottérbeli keresés folytonos változatát használva. Eshetőségi kereséssel a tervkészítés területen belül foglalkoztak (lásd 12. fejezet). A bizonystalan információt felhasználó tervkészítést és cselekvést a valószínűségelemlélet és döntéselmélet eszköztárával oldották meg (lásd 17. fejezet).

Nilsson könyvei a klasszikus keresési algoritmusokkal kapcsolatban hasznos információkat tartalmaznak (Nilsson, 1971; 1980). Korf egy átfogó, naprakészebb áttekintést ad a klasszikus keresési algoritmusokról (Korf, 1988). Az új keresési algoritmusokról – amelyek feltárása figyelemre méltó módon folytatódik – olyan folyóiratokban jelennek meg cikkek, mint például az *Artificial Intelligence*.

## Feladatok

- 3.1. Definiálja saját szavaival a következő fogalmakat: állapot, állapottér, keresési fa, keresési csomópont, cél, cselekvés, állapotátmenet-függvény és elágazási tényező.
- 3.2. Magyarázza meg, hogy a problémamegfogalmazásnak miért kell a célmegfogalmazást követnie.
- 3.3. Tegyük fel, hogy a LEGÁLIS-CSELEKVÉS( $s$ ) azon cselekvések halmazát jelöli, amelyek legálisak az  $s$  állapotban, és az EREDMÉNY( $a, s$ ) pedig azt az állapotot, ami az  $s$  állapotban a végrehajtott legális cselekvés eredménye. Definiálja az ÁLLAPOTÁTMENET-FV-t a LEGÁLIS-CSELEKVÉS és az EREDMÉNY függvényében, és megfordítva.
- 3.4. Mutassa meg, hogy a 8-as játék állapottere két diszjunkt részből áll, ahol egyik részhez tartozó állapotot semmilyen véges lépésszekvenciával sem lehet a másik részhez tartozó állapotba áttranszformálni. (Segítség: Berlekamp és társai, 1982). Dolgozzon ki egy eljárást, amely el tudja dönteni, hogy az állapot melyik részhez tartozik, és magyarázza meg, hogy ez miért hasznos, ha az állapotokat véletlen módon generáljuk.
- 3.5. Gondolja át az  $n$ -királynő problémát a 105. oldalon ismertetett „hatékony” inkrementális megfogalmazást felhasználva. Magyarázza meg, hogy az állapottér mérete miért lesz legalább  $\sqrt{n}!$  és becsülje meg azt a legnagyobb  $n$ -et, amire a kimerítő feltárás még elfogadható. (Segítség: számolja ki az elágazási tényező alsó korlátját figyelembe véve, hogy mekkora a királynő által támadható sakktáblahelyek maximális száma egy tetszőleges oszlopban.)
- 3.6. Egy véges állapottér minden véges keresési fához vezet-e? Mi a helyzet egy olyan véges állapottérrel, amely maga is egy fa? Meg tudja-e precízebben fogalmazni, hogy milyen állapottérfajták vezetnek minden véges keresési fához? Átvéve (Bender, 1996)-ből.
- 3.7. Az alábbiak mindegyikéhez adja meg a kiinduló állapotot, a céltesztet, az állapotátmenet-függvényt és az útköltségfüggvényt. Válasszon meg egy olyan megfogalmazást, amely elegendően precíz, hogy azt implementálni lehessen.
  - (a) Négy szín felhasználásával ki kell színeznie egy síkbeli térképet úgy, hogy minden szomszédos tartomány különböző színű legyen.
  - (b) Egy 1 m magas majom egy szobában tartózkodik, ahol van két 1 m magas lárda (amelyeket egymásra lehet rakni, lehet mozgatni, és meg lehet mászni),

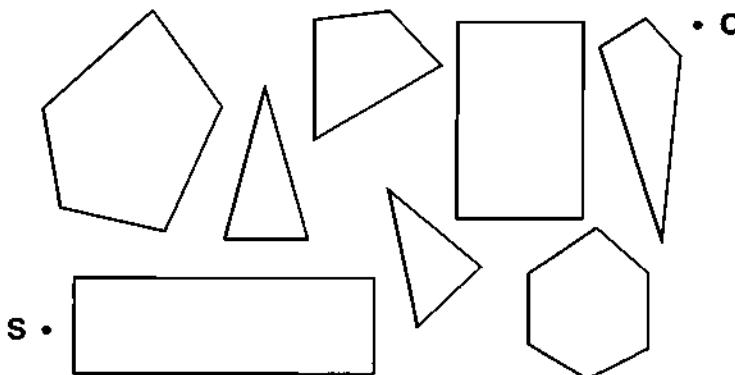
és néhány banán lóg a 2,5 m magas mennyezetről. A majom el szeretné érni a banánt.

- (c) Van egy olyan programja, amely „illegális input rekord” üzenetet ad, ha az input rekordok egy bizonyos állományát dolgozza fel. Ön tudja, hogy egy-egy rekord feldolgozása a többiből független. Szeretné az illegális rekordot megtalálni.
  - (d) Van három kancsója, egy 12 l-es, egy 8 l-es és egy 3 l-es és egy vízcsapja. A kancsókat megtöltheti, kiöntheti belőlük a vizet a földre, vagy az egyikből áttöltheti a vizet egy másikba. Pontosan 1 l vizet kell kimérnie.
- 3.8.** Tekintsen egy olyan állapotteret, ahol a kezdő állapot az 1-es számú és az állapot-átmennet-függvény az  $n$  sorszámú állapotra két állappal, a  $2n$  és a  $2n + 1$  sorszámú állapotokkal tér vissza.
- (a) Rajzolja fel az állapotternek az 1-es állappottól a 15-es állapotig terjedő részét.
  - (b) Tételezze fel, hogy a célállapot a 11-es. Listázza ki a csomópontok kifejtési sorrendjét szélességi, 3-as mélységekkel rendelkező mélységekkel és iteratívan mélyülő keresés esetén.
  - (c) Jó lenne-e a probléma megoldására a kétirányú keresés? Ha igen, magyarázza meg részletesen, hogyan fog ez működni.
  - (d) Mennyi a kétirányú keresés elágazási tényezője minden irányban?
  - (e) Vajon a (c) pontra adott válasz azt sugallja-e, hogy a problémának lehetséges olyan átfogalmazása, hogy az 1-es állapotból egy adott célállapothoz majdnem keresés nélkül el tudnánk jutni?
- 3.9.** A hittérítők és kannibálok problémáját általában az alábbi módon szokás megadni. Három hittérítő és három kannibál egy folyó azonos oldalán tartózkodik. Rendelkeznek csónakkal, amely egy vagy két embert bír el. Találjuk meg annak a módját, hogy mindenkit a folyó másik oldalára átjuttassunk, arra ügyelve, hogy a hittérítők soha, sehol ne legyenek kisebbségben a kannibálokkal szemben. Ez a probléma híres az MI-ben, mert ez volt téma az első olyan publikációnak, ahol a problémamegfogalmazást analitikus szemszögből kísérletek meg (Amarel, 1968).
- (a) Fogalmazza meg precízen a problémát. Csak azokat a megkülönböztetéseket tegye meg, amelyek a helyes megoldás érdekében szükségesek. Rajzolja fel a teljes állapotter diagramját.
  - (b) Implementálja és oldja meg optimális módon a problémát, megfelelő keresési algoritmust választva. Jó ötlet-e az ismétlődő állapotok ellenőrzése?
  - (c) Véleménye szerint miért nehéz e feladvány az emberek számára, annak ellenére, hogy az állapotter ilyen egyszerű?

**3.10.** Implementálja az állapotátmenet-függvény két változatát a 8-as jáék számára.

- Az egyik az összes követőt egyszerre generálja a 8-as jáék adatstruktúrájának másolásával és editálásával. A másik hívásakor, csak egy követőt generál, a szülvő állapot közvetlen módosításával (és szükség esetén a módosítások visszaállításával). Írja meg az iteratívan mélyülő keresés olyan változtatait, amelyek a függvény fenti változatait használják, és hasonlítsa össze a megoldások hatékonyságát.

- 3.11.** A 119. oldalon az **iteratívan megnyúló keresést** (*iterative lengthening search*) említettük, amely az egyenletes költségű keresés iteratív változata. Az ötlet az útköltségre vonatkozó növekvő korlát használata. Egy újonnan generált csomópontot eldobunk, ha az útköltsége a korlánál nagyobb. A következő iterációban a korlátot az előző iterációban eldobott csomópontok közül a legkisebb útköltségre állítjuk be.
- Mutassa meg, hogy ez az algoritmus általános útköltségek esetén optimális.
  - Tekintsen egy homogén fát  $b$  elágazási tényezővel,  $d$  megoldásmélységgel és egységesi lépésköltséggel. Hány iterációra lesz szüksége az iteratívan megnyúló keresésnek?
  - Most tekintsen egy olyan lépésköltséget, amit a  $[0, 1]$  folytonos tartományból sorsolnak, egy minimális pozitív  $\epsilon$  költséggel. Hány iterációra lesz szükség a legrosszabb esetben?
  - Implementálja az algoritmust és alkalmazza a 8-as játék és az utazó ügynök probléma konkrét eseteire. Hasonlítsa össze az algoritmus hatékonyságát az egyenletes költségű algoritmus hatékonyságával, és értelmezze az eredményeit.
- 3.12.** Bizonyítsa be, hogy az egyenletes költségű keresés és a szélességi keresés konstans lépésköltség mellett optimálisak, ha azokat a GRÁF-KERESÉS algoritmusával együtt alkalmazzuk. Mutasson egy olyan állapotteret változó lépésköltséggel, ahol a GRÁF-KERESÉS algoritmus, iteratív mélyülést használva, csak egy szuboptimális megoldást talál.
- 3.13.** Adjon meg egy olyan keresési teret, amelyben az iteratívan mélyülő keresési algoritmus a mélyes keresésnél lényegesen rosszabb teljesítményt nyújt (például  $O(n^2)$  az  $O(n)$ -nel szemben).
- 3.14.** Írjon egy olyan programot, amely bementként két weblap URL-jét kapja meg, és megoldásul megtalálja az azokat összekapcsoló hivatkozási utat (linkek). Mi a megfelelő keresési stratégia? Jó ötlet-e a kétirányú keresés? Alkalmazható-e egy keresőgép az előcsomópont függvény megvalósítására?
- 3.15.** Gondolja végig egy sík két pontja közötti legrövidebb út megtalálásának a problémáját, ha a síkon konvex sokszög akadályok találhatók (lásd 3.22. ábra). Ez annak a problémának egy idealizált változata, amivel egy robotnak kell szembesülnie, amikor egy zsúfolt környezetben kell navigálnia.
- Tegyük fel, hogy az állapotter a sík összes  $(x, y)$  pozíciójából áll. Hány állapotról beszélhetünk ekkor? Hány út vezet a célig?
  - Magyarázza el röviden, hogy egy sokszög csúcsát egy másikkal a síkban összekötő legrövidebb útnak miért kell olyan egyenes vonalszakaszokból állnia, amelyek az egyes sokszögek csúcsait kapcsolják össze? Adjon most egy jó állapotter definíciót. Milyen nagy ez a tér?
  - Definiálja a keresési probléma implementálásához szükséges függvényeket, beleérte az állapotámenet-függvényt, amely bemenetként egy csúcsot kap, és a belőle egyenes vonalban elérhető csúcsok halmazát adja vissza (ne felejt-



**3.22. ábra.** Egy elrendezés sokszögadályokkal

sük ki a szomszédos csúcsokat ugyanazon a sokszögön sem). Heurisztikus függvénynek az egyenes vonalbeli távolságot használja.

- (d) Alkalmazzon a fejezetben ismertetett algoritmusok közül egyet vagy többet a tárgyterülethez tartozó problémák megoldására, és értékelje azok teljesítményét.

### 3.16. A 3.15. feladat navigációs problémáját a következőképpen transzformálja át egy környezetbe:

- Az érzékelés az ágens által látható sokszögcímsok *az ágenshez viszonyított helyzetének* listája. Az érzékelés a robot pozícióját *nem* tartalmazza! A robotnak a saját pozícióját a térkép alapján kell megtanulnia. Azt is tételezzük fel ideiglenesen, hogy minden lokációnak eltérő a „panorámája”.
  - minden cselekvés egy a követendő egyenes utat leíró vektor lesz. Ha az úton nincsenek akadályok, a cselekvés sikeres. Egyébként a robot azon a ponton megáll, ahol az út az első akadályt metszi. Ha az ágens a zérus mozgás vektorát adja vissza és a célpozícióban van (ami rögzített és ismert), akkor a környezetnek teleportálnia kell az ágenst *egy véletlen lokációba* (de nem egy akadályon belülre).
  - A hatékonysági mérték 1 pontot kér az ágenstől a megtett távolság minden egységtávjaért és 1000 ponttal díjazza a cél mindenkorai elérését.
- (a) Implementálja ezt a környezetet és egy erre a környezetre vonatkozó problémamegoldó ágenst. minden teleportálás után az ágensnek új célt kell megfogalmaznia, beleértve a saját pozíció felfedezését.
  - (b) Dokumentálja az ágens teljesítményét (kommentálja az ágens mozgását), és írjon egy jelentést az ágens 100 epizódra vonatkozó teljesítményéről.
  - (c) Módosítsa a környezetet úgy, hogy az ágens az esetek 30%-ban ne a szándékolt helyen találja magát (a helyet a látható sokszögcímsokból sorsoljuk, ha van ilyen egyáltalán, különben nincs mozgás). Ez a valós robot hibás mozgásának egy durva modellje. Módosítsa ágensét úgy, hogyha az ilyen hibát detektál, derítse ki, hol van, és készítsen tervet, hogy az eredeti pozí-

cióba visszakerülhessen, ahonnan az eredeti tervét folytatja. Emlékezzen arra, hogy néha az eredeti pozícióba való visszatérés is kudarccal végezhető! Mutasson egy olyan ágenst, amely két egymás utáni hibás mozgással is sikeresen megbirkózik, és eléri a célt.

- (d) Most két különböző visszaállítási sémával kísérletezzen egy hiba meg-történt után: (1) induljon az eredeti út legközelebb eső csúcsa felé, és (2) tervezze át az utat a cél felé az új pozícióból kiindulva. Hasonlítsa össze a három visszaállítási séma teljesítményét. Meg fogja-e változtatni az összehasonlítás eredményét a keresési költségek figyelembevétele?
  - (e) Most tételezze fel, hogy vannak lokációk azonos „panorámával” (tételezzük fel például, hogy a világ egy négyzetrács négyzetes akadályokkal). Milyen problémákkal kell most az ágensnek szembenéznie? Milyenek most a megoldások?
- 3.17.** A 100. oldalon azt mondta, hogy nem tárgyaljuk a negatív útköltséggel rendelkező problémákat. Ennek a feladatnak a kapcsán részletesebben megvizsgáljuk a kérdést.
- (a) Tegyük fel, hogy a cselekvéseknek tetszőlegesen nagy negatív költségük lehet. Magyarázza meg, hogy ez a lehetőség miért kényszerít minden optimális algoritmust a teljes állapottér felkutatására.
  - (b) Segíteni fog-e, ha ragaszkodunk ahoz, hogy a lépésköltség egy negatív c konstansnál nagyobb vagy azzal egyenlő legyen? Mind a fákra, mind a gráfokra gondoljon.
  - (c) Tételezzük fel, hogy cselekvések egy halmaza ciklust alkot úgy, hogy a halmaz cselekvéseit valamelyen sorrendben végrehajtva az állapot nem változik. Milyen következményekkel jár egy ilyen környezetben tevékenykedő ágens optimális viselkedésére, ha ezen cselekvések mindegyike negatív költségű?
  - (d) Könnyen el tudunk képzelni nagy negatív költségű operátorokat még az útkeresési probléma esetén is. Például néhány útszakasz olyan gyönyörű tájakon halad, hogy az idő és üzemanyag tekintetében messze túlszárnjalja a normál költségeket. Pontos megfogalmazásban magyarázza el, hogy az emberek miért nem kocsikálnak a végtelenségig a szép tájak körül, és hogyan lehetne az útkeresési problémához az állapotteret és az operátorokat definiálni, hogy a mesterséges ágens is el tudja kerülni a ciklusokat.
  - (e) Tud olyan valódi problémakört mondani, amelyben a lépések költsége ciklust okoz?
- 3.18.** Gondoljon a Murphy-törvény uralma alatt álló kétlokációjú, szenzor nélküli porszívóvilágra. Rajzolja fel az  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  kezdeti hiedelmi állapotból elérhető hiedelem-állapotteret, és magyarázza meg, hogy a probléma miért megoldhatatlan. Mutassa meg azt is, hogy ha a világ teljesen megfigyelhető, minden kezdeti állapotból létezik egy megoldási szekvencia.

**3.19.** Tekintse a 2.2. ábrán látható porszívóvilágot.

-  (a) A fejezetben ismertetett algoritmusok közül amelyik lenne alkalmas e probléma megoldására? Szükséges-e, hogy az algoritmus ellenőrizze az ismételt állapotokat?

- (b) A  $3 \times 3$  világban alkalmazza a megválasztott algoritmust az optimális cselekvési szekvencia kiszámítására egy olyan kezdeti állapottal, ahol a felső három négyzet mindegyikében található piszok, és az ágens a középső négyzetben tartózkodik.
- (c) Építse kereső ágenst, és értékelje a teljesítményét egy olyan  $3 \times 3$  porszívó-világ halmaz számára, ahol minden négyzetben 0,2 a kosz valószínűsége. A teljesítmény mérésénél mind az útköltséget, mind a keresési költséget vegye figyelembe, egy értelmes átszámítási tényezőt használva.
- (d) Hasonlítsa össze az eddigi legjobb kereső ágens viselkedését egy olyan egyszerű, randomizált reflexszerű ágensével, amely koszt szív, ha azt megtalálja, különben véletlen módon mozog.
- (e) Értékelje, hogy mi történne, ha a világot  $n \times n$ -re nagyítanánk. Hogyan változna a kereső és a reflexszerű ágens teljesítménye  $n$  növelésével?

# 4. INFORMÁLT KERESÉSI ÉS FELFEDEZŐ MÓDSZEREK

*Ebben a fejezetben látni fogjuk, hogy az állapottérrel kapcsolatos információ segítségével az algoritmusok hogyan kerülhetik el a sötétben való tapogatózást.*

A 3. fejezetben láttuk, hogy a nem informált keresési stratégiák oly módon képesek problémák megoldásait megtalálni, hogy szisztematikusan új állapotokat generálnak és összehasonlítják azokat a célállapottal. Sajnos ezek a stratégiák a legtöbb esetben hiányosan tudnak alkalmazni a megoldást. A 4.1. alfejezet bemutatja a 3. fejezetben tanulmányozott algoritmusok informált változatait, a 4.2. alfejezet pedig elmagyarázza, hogy a szükséges problémáspecifikus információ hogyan szerezhető meg. A 4.3. és a 4.4. alfejezet olyan algoritmusokkal foglalkozik, amelyek az állapottérben tisztán lokális keresést (**local search**) hajtanak végre, egy vagy több aktuális állapotot értékelve és módosítva ahelyett, hogy szisztematikusan tárnák fel az utat a kezdeti állapottól indulva. Ezek az algoritmusok olyan problémák esetén jók, ahol az útköltség közömbös, és az egyetlen, ami számít, hogy megtaláljuk-e a megoldást. A lokális keresési algoritmusok családjába a statisztikai fizika inspirálta módszerek (**szimulált lehűtés, simulated annealing**) és az evolúciós biológia sugallta módszerek (**genetikus algoritmusok, genetic algorithms**) is beletartoznak. A 4.5. alfejezet végül az online kereséssel (**online search**) foglalkozik, ahol az ágens egy teljesen ismeretlen állapottérrel találja magát szembe.

## 4.1. INFORMÁLT (HEURISZTIKUS) KERESÉSI STRATÉGIÁK

Ez a fejezet megmutatja, hogy az **informált keresési (informed search)** stratégia – amely a probléma definícióján túlmenően problémáspecifikus tudást is felhasznál – hogyan képes hatékonyabban megtalálni a megoldást.

Az általunk vizsgált általános megközelítést a **legjobbat-először** keresésnek (**best-first search**) nevezük. A legjobbat-először keresés az általános FA-KERESÉS vagy GRÁF-KERESÉS algoritmusok olyan speciális esete, ahol egy csomópont kifejtésre való kiválasztása egy  $f(n)$  **kiértékelő függvénytől (evaluation function)** függ. Hagyományosan a *legkisebb* értékű csomópontot választjuk kifejtésre, mert a kiértékelő függvény a céltól való távolságot méri. A legjobbat-először keresés az eddigi általános keresési eljárások keretein belül egy prioritási sor segítségével implementálható, ami egy olyan adatstruktúra, mely a peremet a növekvő  $f$ -értékek szerint rendezzi.

A legjobbat-először keresés egy nagy múltú, azonban pontatlan elnevezés. Amennyiben *valóban* képesek lennének a legjobb csomópontot kifejeni, akkor egyáltalán nem

kellene keresnünk, nyílegyenesen elmasíroznánk a célohoz. Ezzel szemben csak a kiértékelő függvény szerint legjobbnak *tűnő* csomópontot tudjuk kiválasztani. Ha kiértékelő függvényünk minden tudó, akkor a kiválasztott csomópont egyben a legjobb csomópont is. A valóságban azonban a kiértékelő függvény néha pontatlan, és félrevezetheti a keresést. Azonban a továbbiakban is ragaszkodni fogunk a legjobbat-először keresés elnevezéshez, mert a legjobbnak tűnő először keresés egy kicsit furcsán hangzana.

A LEGJOBBAT-ELŐSZÖR-KERESÉS algoritmus tulajdonképpen egy keresési algoritmus család, amelynek az elemeit az eltérő kiértékelő függvények<sup>1</sup> különböztetik meg. Ezeknek az algoritmusoknak a kulcseleme a  $h(n)$ -nel jelölt heurisztikus függvény<sup>2</sup> (heuristic function):

$$h(n) = \text{az } n \text{ csomóponttól a célig vezető legolcsóbb út becsült útköltsége}$$

Például Romániában az Arad és Bukarest közötti legolcsóbb út költségét az Arad és Bukarest közötti légvonaltávolsággal meg lehetne becsülni.

A heurisztikus függvény a leginkább megszokott módja annak, hogy a problémára vonatkozó pótólágos tudást a keresési algoritmusba be tudjuk injektálni. A heurisztikus függvényekkel részletesen a 4.2. alfejezetben foglalkozunk. Egyelőre tetszőleges, ám problémáspecifikus függvényeknek fogjuk őket tekinteni, egy kikötéssel: ha  $n$  egy célállapot, akkor  $h(n) = 0$ . A jelen fejezet hátralévő része két olyan utat mutat be, ahol a heurisztikus információt a keresés irányítására használjuk fel.

## A mohó legjobbat-először keresés

A mohó legjobbat-először keresés<sup>3</sup> (greedy best-first search) azt a csomópontot fejti ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapot-hoz, abból kiindulva, hogy így gyorsan megtalálja a megoldást. A csomópontokat az algoritmus tehát az  $f(n) = h(n)$  heurisztikus függvénytelével értékeli ki.

Nézzük meg, hogy hogyan működik ez a romániai útkeresésnél a légvonalban mért távolságot (straight-line distance) felhasználva, amit  $h_{LMT}$ -vel fogunk jelölni. Ha a cél Bukarest, szükségünk lesz Bukarest légvonalbeli távolságaira, amelyeket a 4.1. ábra ad meg. Például  $h_{LMT}(\text{Benn(Arad)}) = 366$ . Vegyük észre, hogy a  $h_{LMT}$  értékeit magának a problémának a leírásából kiszámítani nem lehet. Továbbá egy kis tapasztalat is szükséges ahhoz, hogy rájöjjünk, hogy a  $h_{LMT}$  az úton megtett tényleges távolsággal korrelál, és így számunkra hasznos heurisztika lehet.

A 4.2. ábra egy Aradról Bukarestbe vezető út mohó legjobbat-először keresését mutatja,  $h_{LMT}$ -t alkalmazva. Aradból az első kifejtett csomópont Nagyszeben, mert ez közelebb van Bukaresthez, mint Nagyzerénd vagy Temesvár. A következő kifejtendő

<sup>1</sup> A 4.3. feladataban azt kérjük Öntől, hogy mutassa ki, hogy ez a család néhány jól ismert nem informált keresést is tartalmaz.

<sup>2</sup> A  $h(n)$  heurisztikus függvény bemenete ugyan egy csomópont, a függvény értéke azonban a csomóponthoz tartozó állapottól függ.

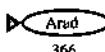
<sup>3</sup> A könyv első kiadásában ennek mohó keresés (greedy search) volt a neve. Más szerzők ezt legjobbat-először keresésnek (best-first search) nevezik. A második változat általunk választott általánosabb használata Pearl (Pearl, 1984) követi.

Arad	366	Mehadia	241
Bukarest	0	Nearmt	234
Craiova	160	Nagyvárad	380
Dobreta	242	Pitești	100
Eforie	161	Rimniciu Vilcea	193
Fogaras	176	Nagyszeben	253
Giurgiu	77	Temesvár	329
Hirsova	151	Csalámos	80
Iasi	226	Vaslui	199
Lugos	244	Nagyzerénd	374

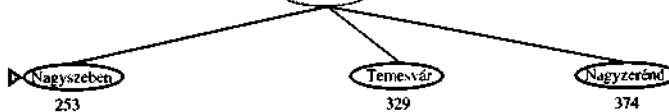
4.1. ábra. A  $h_{LMT}$  értékei – a légvonalbeli távolságok Bukarestig

csomópont pedig Fogaras, mert az van a legközelebb. Fogaras majd generálja Bukarestet, ami egyben a célállapot. Erre a konkrét problémára a  $h_{LMT}$ -t alkalmazó mohó legjobbat-először keresés úgy talál megoldást, hogy soha sem fejt ki olyan csomópontot, ami nem a megoldási úton fekszik. Következésképpen minimális a keresési költsége. Azonban nem optimális: a Nagyszebenen és Fogarason keresztül Bukarestbe vezető út 32 kilométerrel hosszabb a Rimnicu Vilceán és Piteștin keresztül vezető útnál. Ebből látszik,

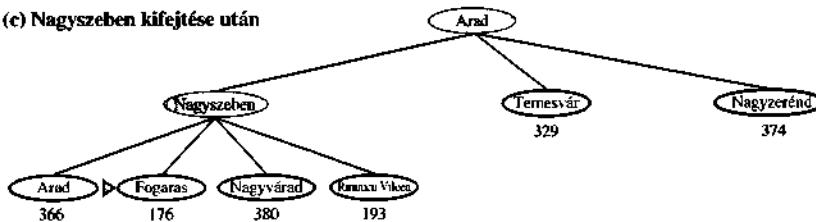
(a) Kezdeti állapot



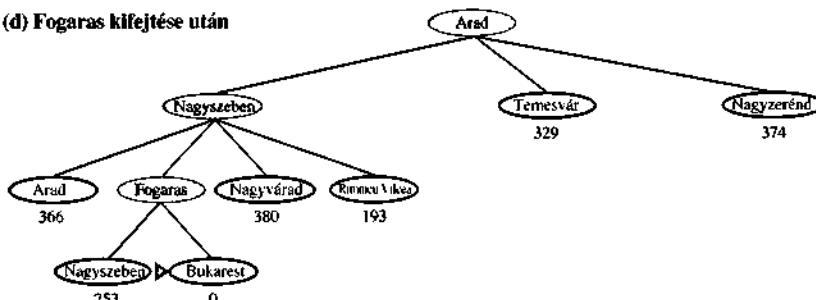
(b) Arad kifejtése után



(c) Nagyszeben kifejtése után



(d) Fogaras kifejtése után

4.2. ábra. A mohó legjobbat-először keresés lépései Bukarest esetén a légvonalban mért távolságot ( $h_{LMT}$ ) alkalmazva. A csomópontok a saját  $h$ -értékeikkal vannak felcímkézve.

hogy az algoritmus miért „mohó” – minden lépésben igyekszik annyira közel kerülni a célhoz, ahogy csak lehet.

A  $h(n)$  minimalizálása érzékeny a hibás kezdő lépésekre. Tekintsük például azt az esetet, amikor Iaşiról Fogarasra akarunk eljutni. A heurisztika alapján Neamtot kellene először kifejteni, hiszen ez fekszik Fogarashoz a legközelebb, azonban ez zsákutca. A megoldás, hogy először elmegyünk Vasluira – egy olyan lépést teszünk, ami a heurisztika szerint a céltól távolabb visz – és aztán elmegyünk Csalánosra, Bukarestre, majd Fogarasra. Ebben az esetben a heurisztika szükségtelen csomópontok kifejtését eredményezi. Továbbá, ha nem ügyelünk arra, hogy felismerjük az ismétlődő állapotokat, akkor soha nem találjuk meg a megoldást – a keresés Neamt és Iaşi között fog oszcillálni.

A mohó legjobbat-először keresés a mélységi keresésre hasonlít abból a szempontból, hogy egyetlen út végigkövetését preferálja a célig, azonban zsákutcába jutva visszalép. Ugyanazokkal a problémákkal küszködik, mint a mélységi keresés – nem optimális és nem teljes (mert elindulhat egy végtelen úton és soha sem tér vissza újabb lehetőségeket kipróbalni). A mohó keresés legrosszabb esetre számított (worst-case) idő- és tárigénye  $O(b^m)$ , ahol  $m$  a keresési térfelület maximális mélysége. Jól megválasztott heurisztikus függvénytel a komplexitás azonban jelentősen csökkenhető. A csökkenés mértéke az adott problémától és a heurisztikus függvény minőségétől függ.

## **A\* keresés: a teljes becsült útköltség minimalizálása**

A legjobbat-először keresés leginkább ismert változata az A\* keresés (a kiejtése ’A csillag’). A csomópontokat úgy értékeli ki, hogy összekombinálja  $g(n)$  értékét – az aktuális csomópontig megtett út költsége – és  $h(n)$  értékét – vagyis az adott csomópont-tól a célhoz vezető út költségének becslőjét:

$$f(n) = g(n) + h(n)$$

Mivel  $g(n)$  megadja a kiinduló csomóponttól az  $n$  csomópontig számított útköltséget, és  $h(n)$  az  $n$  csomóponttól a célcsomópontra vezető legolcsóbb költségű út költségének becslője, így az alábbi összefüggést kapjuk:

$$f(n) = \text{a legolcsóbb, az } n \text{ csomóponton keresztül vezető megoldás becsült költsége.}$$

Így amennyiben a legolcsóbb megoldást keressük, ésszerű először a legkisebb  $g(n) + h(n)$  értékkal rendelkező csomópontot kifejteni. Ezen stratégia kellemes tulajdonsága, hogy ez a stratégia több mint ésszerű: amennyiben a  $h$  függvény eleget tesz bizonyos feltételeknek, az A\* keresés teljes és optimális.

Az A\* optimalitását könnyű elemezni, ha az algoritmust a FA-KERESÉS-sel együtt alkalmazzuk. Ilyenkor A\* optimális lesz, ha  $h(n)$  egy **elfogadható heurisztika** (**admissible heuristic**), azaz ha  $h(n)$  soha nem becsüli felül a cél eléréséhez szükséges költséget. Az elfogadható heuristikák természetükön belül optimisták, mivel úgy gondolják, hogy a probléma megoldása kisebb költséggel jár, mint amekkora költséget a megoldás valójában igényel. Mivel  $g(n)$  az  $n$  csomópont elérésének pontos költsége, azonnali következményként adódik, hogy  $f(n)$  soha sem becsüli túl az  $n$  csomóponton keresztül vezető legjobb megoldás valódi költségét.

Az elfogadható heurisztikus függvények talán egyik legnyilvánvalóbb példája a Bukarestbe történő utazás során felhasznált  $h_{LMT}$  légyonalban mért távolság. A légyonalban mért távolság elfogadható, mert bármely két pont között a legrövidebb távolság a légyonalban mért távolság, így a légyonalbeli táv soha nem becsülhet túl. A 4.3. ábra a Bukarestet kereső A\* fakeresés előrehaladását mutatja. A g értékeket a 3.2. ábrán látható lépésköltségekből számítjuk ki, a  $h_{LMT}$  értékei a 4.1. ábrán adottak. Végük észre, hogy Bukarest először az (e) lépés peremében jelent meg, azonban kifejtésre nem került, mert Bukarest f-értéke (450) magasabb, mint Piteștié (417). Ezt úgy lehetne megmagyarázni, hogy *lehet*, hogy Piteștin keresztül létezik egy 417 költségű olcsó megoldás, így az algoritmus egy 450 költségű megoldást nem fog választani. Ebből a példából megalkothatjuk annak az általános bizonyítását, hogy a FA-KERESÉS-t használó A\* algoritmus optimális, ha  $h(n)$  elfogadható. Tegyük fel, hogy a peremen egy  $G_2$  szuboptimális célcsomópont jelenik meg, és az optimális megoldás költsége legyen  $C^*$ . Így mivel  $G_2$  szuboptimális és  $h(G_2) = 0$  (ami minden célállapotra igaz), tudjuk, hogy:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

Gondolunk most egy perembeli  $n$  csomópontra, amely a megoldási útvonalon fekszik (ilyennek minden léteznie kell, ha a megoldás létezik). Ha  $h(n)$  nem becsüli túl a megoldáshoz vezető út súlyát, akkor tudjuk, hogy:

$$f(n) = g(n) + h(n) \leq C^*$$

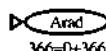
Kimutattuk tehát, hogy  $f(n) \leq C^* < f(G_2)$ , így  $G_2$  nem kerül kifejtésre, és az A\* egy optimális megoldással tér vissza.

A bizonyítás összeomlik, ha a FA-KERESÉS helyett a 3.19. ábra szerinti GRÁF-KERESÉS algoritmust használjuk. Az algoritmus visszatérhet szuboptimális megoldással, mert a GRÁF-KERESÉS algoritmus elvetheti az ismétlődő állapothoz vezető optimális utat, ha az nem elsőnek került kiszámításra (lásd 4.4. feladat). A problémát kétféle módon lehet megoldani. Az első megoldás a GRÁF-KERESÉS olyan kiterjesztése, hogy az az ugyanahhoz a csomóponthoz vezető két út közül a drágábbat fogja elvetni (lásd az értékelést a 3.5. alfejezetben). A pótolagos adminisztrálás nem egyszerű, de az optimalitást garantálni fogja. A második megoldásnál azt kell biztosítani, hogy a bármelyik ismétlődő csomóponthoz vezető optimális út mindenkorán követ – mint ahogy ez az egyenletes költségű keresésénél volt. Ez a tulajdonság akkor áll fenn, ha a  $h(n)$  függvényre extra követelményeket fogalmazunk meg, megkövetelve annak konzisztenciáját (**consistency**), másnéven **monotonitását** (**monotonicity**). A  $h(n)$  heurisztikus függvény konzisztens, ha minden  $n$  csomópontra és annak egy tetszőleges  $a$  cselekvéssel generált minden  $n'$  utódcsomópontjára az  $n$  csomóponttól elérő cél becsült költsége nem kisebb, mint az  $n'$ -be kerülés lépésköltsége és az  $n'$  csomóponttól elérő cél becsült költsége:

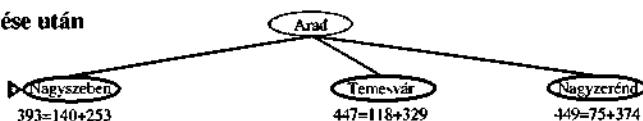
$$h(n) \leq c(n, a, n') + h(n')$$

Ez az általános háromszög egyenlőtlenség (**triangle inequality**) egy formája, amely azt fejezi ki, hogy egy háromszög egy oldala sem lehet hosszabb, mint a két másik oldal összege. Itt a háromszöget az  $n$ , az  $n'$  és az  $n$ -hez legközelebbi cél határozza meg. Könnyű megmutatni (4.7. feladat), hogy minden konzisztens heurisztika egyben elfogadható is. A konzisztencia legfontosabb következménye az, hogy: a GRÁF-KERESÉS-t használó A\* algoritmus optimális, ha  $h(n)$  konzisztens.

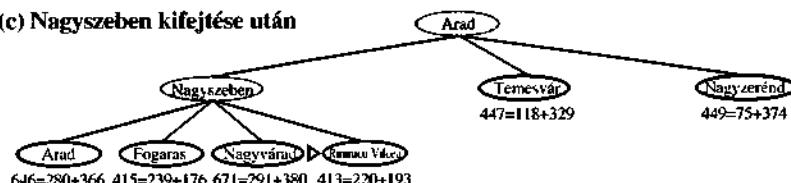
## (a) Kezdeti állapot



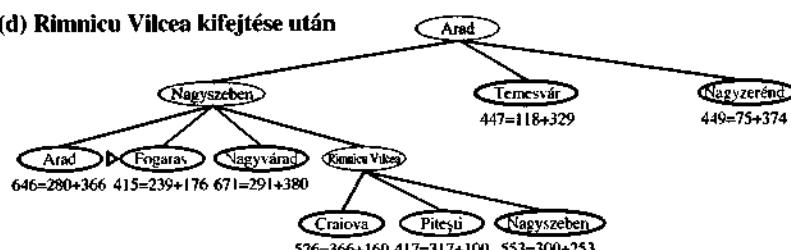
## (b) Arad kifejtése után



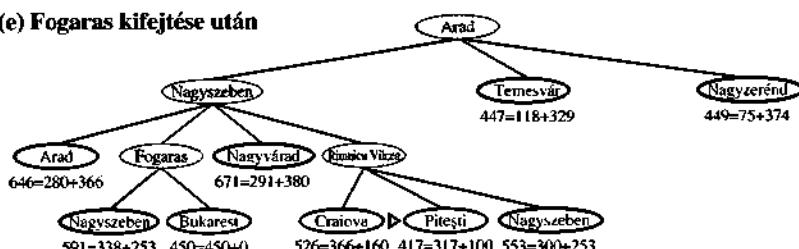
## (c) Nagyszeben kifejtése után



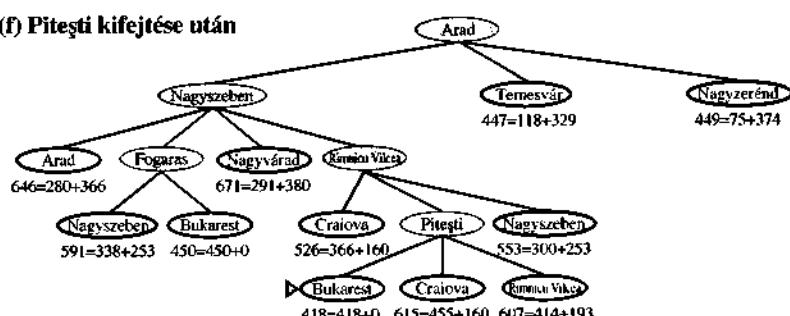
## (d) Rimnicu Vilcea kifejtése után



## (e) Fogaras kifejtése után



## (f) Pitești kifejtése után



**4.3. ábra.** Az A\* keresés lépései Bukarest keresése során. A csomópontok az  $f = g + h$  értékkal vannak felcímkézve. A  $h$ -értékek a Bukaresttől légyonalban mért távolságokat jelölik, melyeket a 4.1. ábrából vettünk át.

Bár a konzisztencia az elfogadhatóságnál szigorúbb követelmény, igazán nehéz olyan elfogadható heurisztikát találni, ami nem lenne egyben konzisztens. Az ebben a fejezetben tárgyalt összes elfogadható heurisztika minden konzisztens. Vegyük például a  $h_{LMT}$ -t. Tudjuk, hogy az általános háromszög egyenlőtlenség teljesül, ha az oldalakat egyenes vonalú távolságokkal mérjük, és hogy az  $n$  és  $n'$  közötti egyenes vonalú távolság  $c(n, a, n')$ -nél nem nagyobb. A  $h_{LMT}$  így egy konzisztens heurisztika.

A konzisztencia egy másik fontos következménye az, hogy ha  $h(n)$  konzisztens, akkor az  $f(n)$  értékek akármilyen út mentén nem csökkennek. A bizonyítás bizonyos  $a$  mellett közvetlenül következik a konzisztencia definíciójából. Tegyük fel, hogy  $n'$  az  $n$  utódja, ekkor:

$$g(n') = g(n) + c(n, a, n')$$

és

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

A GRÁF-KERÉSÉS-t használó A\* algoritmus által kifejtett csomópontok sorozata tehát  $f(n)$ -ben nem csökkenő. A kifejtésre választott első célcsomópont így egy optimális megoldás is egyben, mert minden utána következő csomópont legalább ilyen költséges lenne.

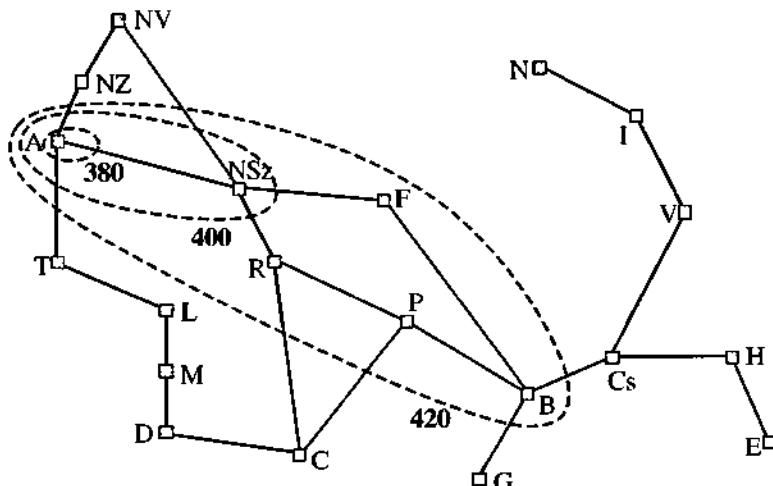
Ha az  $f$ -költségek sohasem csökkenek, bármilyen utat választunk, akkor az állapotterületben határvonalakat (**contour**) húzhatunk be, hasonlóan egy topografikus térkép kontúrjaihoz. A 4.4. ábra erre mutat egy példát. A 400-zal felcímzett határvonalon belül az összes csomópont  $f(n)$  értéke nem nagyobb 400-nál, és hasonló érvényes a többire is. Mivel az A\* keresési algoritmus a legkisebb  $f$  értékkel rendelkező levélcsomópontot fejti ki először, láthatjuk, hogy az A\* keresési algoritmus a gyökér csomópontból legyezőszerűen halad kifelé, növekvő  $f$  értékekhez tartozó koncentrikus sávokban hozzáadva a csomópontokat.

Az egyenletes költségű keresés esetén (A\* keresés  $h = 0$  mellett) a csomópontoknak a kiinduló csomópont köré húzott koncentrikus „körök” alkotnak. Pontosabb heurisztikus függvény alkalmazásával a sávok a célállapot felé elnyúlnak, és keskenyebben fókusztálónak az optimális út körül. Ha C\* az optimális megoldási út költségét jelöli, akkor az alábbiakat jelenthetjük ki:

- Az A\* keresési algoritmus kifejti az összes  $f(n) < C^*$  értékkel rendelkező csomópontot.
- Ezek után az A\* keresési algoritmus egy célcsomópont kiválasztása előtt még kifejthet néhány csomópontot a „celhatárvonalon”, amelyekre  $f(n) = C^*$ .

Intuitíven nyilvánvaló, hogy az első megtalált megoldásnak optimális megoldásnak kell lennie, hiszen a következő határvonalakon az összes csomóponthoz nagyobb  $f$  költség, ebből adódóan nagyobb  $g$  költség tartozik (mivel minden célállapotra  $h(n) = 0$ ). Intuitíven az is nyilvánvaló, hogy az A\* keresési algoritmus teljes. Ahogy egyre növekvő  $f$  értékű sávokat adunk a kereséshez, előbb-utóbb elérünk egy sávot, amelyhez tartozó  $f$  érték megegyezik egy célállapothoz vezető út költségével.<sup>4</sup>

<sup>4</sup> A teljesség megkívánja, hogy a  $C^*$  költségnél kisebb vagy azonos a  $f$  költségű csomópontokból véges sok legyen. Ez a feltétel akkor lesz igaz, ha minden lépéskötöség egy véges  $\epsilon$ -nál nagyobb és  $b$  véges.



4.4. ábra. Románia térképe. Az ábra Arad mint kiinduló állapot esetén az  $f = 380$ ,  $f = 400$  és  $f = 420$  értékekhez tartozó határvonalat mutatja. Egy adott határvonalon belüli csomópontokhoz a határvonal érékénél kisebb  $f$  költség tartozik.

Vegyük észre, hogy az  $A^*$  nem fejt ki  $f(n) > C^*$  tulajdonságú csomópontokat. A 4.3. ábrán például Temesvár nem kerül kifejtésre annak ellenére, hogy ez a gyökérnek egy utódcsomópontja. Azt mondjuk, hogy a Temesvár alatti fát **lenyestük (pruned)**. Mivel  $h_{LMT}$  elfogadható, az algoritmus biztonságosan figyelmen kívül hagyhat egy ilyen fát, miközben az optimalitást garantálja. A nyesés gondolata – a lehetőségek eliminálása anélkül, hogy megvizsgálnánk azokat – igen fontos az MI sok területén.

Egy végső észrevétel, az ilyen típusú – a gyökérből kiinduló utakat bővíti – optimális algoritmusok közül az  $A^*$  keresési algoritmus bármely adott heurisztikus függvény mellett **optimális hatékonyágú (optimally efficient)**. Ez azt jelenti, hogy egyetlen más optimális algoritmus sem fejt ki garantáltan kevesebb csomópontot, mint az  $A^*$  (kivéve talán az  $f(n) = C^*$  típusú csomópontok körét, ahol holtverseny alakulhat ki). Ez azért van, mert az összes olyan algoritmus, amelyik *nem* fejti ki az összes csomópontot, melyre  $f(n) < C^*$ , kockázatja az optimális megoldás elkerülését.

Felettesebb örömteli hír számunkra, hogy az  $A^*$  keresési algoritmus teljes, optimális és az összes ilyen jellegű algoritmus között optimálisan hatékony. Sajnos ez azonban nem jelenti azt, hogy az  $A^*$  algoritmus megoldja az összes kereséssel kapcsolatos problémákat. A buktató a dologban az, hogy a legtöbb probléma esetén a csomópontok száma a keresési tér céllhatáron belüli részén a megoldás hosszának még mindig exponenciális függvénye. Bár az eredmény bizonyítása túlmutat ezen könyv keretein, megmutatható, hogy az exponenciális növekedéssel mindenképp szembe kell néznünk, kivéve, ha a heurisztikus függvényünk hibája legfeljebb az aktuális útköltség logaritmusával nő. Az exponenciálisnál lassabb növekedés feltétele matematikai megfogalmazásban:

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

ahol  $h^*(n)$  az  $n$  csomópontból a célcsomópontba való eljutás *valódi* költségét jelöli. Majdnem minden, a gyakorlatban használt heurisztikus függvény esetén a hiba legalább

arányos az útköltséggel, és az ebből adódó exponenciális növekedéssel egyetlen számítógép sem tud megbirkózni. Ezért ahhoz ragaszkodni, hogy egy optimális megoldást találunk, gyakran nem is praktikus. Használhatjuk az A\* olyan változatát, amely a szuboptimális megoldásokat gyorsan megtalálja, vagy pedig dolgozhatunk pontosabb, de nem elfogadható heurisztikákkal. Egy jól megválasztott heurisztikus függvény ettől függetlenül a nem informált keresési algoritmusokhoz képest jelentős megtakarítást eredményezhet. A 4.2. alfejezetben megvizsgáljuk, hogy hogyan is lehet jó heurisztikus függvényt tervezni.

Az A\* algoritmusnak azonban nem a szükséges számítási idő a nagy problémája. Mivel az összes legenerált csomópontot a memóriában tárolja (ahogy ezt az összes GRÁF-KERESÉS algoritmus teszi), ezért az algoritmus általában lényegesen hamarabb felmészti a rendelkezésre álló memóriát, mintsem kifutna az időből. Ezért az A\* sok nagyméretű problémához nem praktikus. A közelmúltban kifejlesztett algoritmusok a végrehajtási idő kismértékű növekedése mellett a memóriaproblémát megoldották, anélkül hogy feláldoznának az optimalitást vagy a teljességet. Ezekkel az algoritmusokkal fogunk most foglalkozni.

## Memóriakorlátozott heurisztikus keresés

Az A\* memóriaigényének mérséklésére a legegyszerűbb módszer az iteratívan mélyülő algoritmus adaptálása heurisztikus keresés környezetére. Ennek eredménye az **iteratívan mélyülő A\*** algoritmus – **IMA\*** – (**iterative deepening A\*, IDA\***). Az IMA\* és a közönséges iteratívan mélyülő algoritmus közötti fő különbség az, hogy a vágási mechanizmus nem a mélységen, hanem az  $f$  költségen ( $g + h$ ) alapul. Ezáltal minden egyes iterációban a vágási érték az a legkisebb  $f$  költség, ami az előbbi iterációban használt vágási értéknél nagyobb. Az IMA\* praktikus megoldás számos olyan probléma esetén, ahol egységnyi a lépésköltség és elkerüli a rendezett csomópontsor memóriában való tartásának jelentős overheadjét. Sajnos, az IMA\* algoritmus a valós értékű költségektől ugyanúgy szenved, mint az egyenletes költségű keresés iteratív változata, amit a 3.11. feladataban írtunk le. Ebben a fejezetben megvizsgálunk a memóriakorlátozott algoritmusok köréből két frissebb ötletet – az RLEK-t és az MA\*-t.

A **rekurzív legjobbat-először** keresés, az **RLEK** (**recursive best-first search, RBFS**) egy egyszerű rekurzív algoritmus, amely megkíséri a rendes legjobbat-először algoritmus működését műmérni, de csak lineáris tárat használva. Az algoritmust a 4.5. ábra mutatja. A struktúrája hasonlít a rekurzív mélyiségi keresésre, azonban ahelyett, hogy az algoritmus egy utat a végtelenségig folytatna az aktuális pálya mentén, figyeli az aktuális csomóponthoz az elődeitől vezető eddig legjobb alternatív út  $f$ -értékét. Ha az aktuális csomópont ezt az értéket túlhaladja, a rekurzió az alternatív útra lép vissza. Ahogy a rekurzió visszalép, az RLEK minden csomópont  $f$ -értékét a pálya mentén a gyerekeinek legjobb  $f$ -értékével helyettesíti. Ily módon az RLEK emlékszik a legjobb levélértékre az elfelejtett alfában, és eldöntheti, vajon érdemes-e ezt a fát valamikor később újra kifejteni. A 4.6. ábra azt mutatja, hogy hogyan éri el Bukarestet az RLEK.

Az RLEK valamivel hatékonyabb, mint az IMA\*. azonban még mindig túlságosan szenved a csomópontok túlzott újból generálása miatt. A 4.6. ábrán látható példában az RLEK először a Rimnicu Vilceán átmenő utat követi, majd „meggondolja magát” és

```

function REKURZÍV-LEGOBBAT-ELŐSZÖR-KERESÉS(probléma) returns egy megoldás vagy kudarc
  RLEK(probléma, Csomópontot-Létrehoz(Kinduló-Állapot[probléma],  $\infty$ ))

function RLEK(probléma, csmópont, f_korlát) returns egy megoldás, vagy kudarc,
  és egy új f-költség korlát
  if CEL-TESZT[probléma] / ÁLLAPOT[csmópont] then return csmópont
  követők  $\leftarrow$  KIFEJT(csmópont, probléma)
  if követők egy üres lista then return kudarc,  $\infty$ 
  for each s in követők do
    f[s]  $\leftarrow$  max(g(s) + h(s), f[csmópont])
  repeat
    legjobb  $\leftarrow$  a legkisebb f-értékű csomópont a követők közül
    if f[legjobb] > f_korlát then return kudarc, f[legjobb]
    alternatíva  $\leftarrow$  a második legkisebb f-értékű csomópont a követők közül
    eredmény, f[legjobb]  $\leftarrow$  RLEK(probléma, legjobb, min(f_korlát, alternatíva))
    if megoldás ≠ kudarc then return megoldás
  
```

4.5. ábra. A rekurzív legjobbat-először keresés algoritmusá

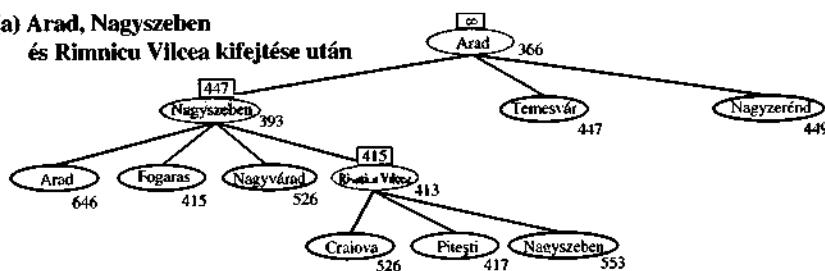
Fogarassal kísérletezik, majd újra meggondolja magát. Ezek a meggondolások azért történnek meg, mert amikor az aktuális legjobb utat tovább fejtjük, nagy az esély arra, hogy az *f*-érték nőni fog – *h* általában kevésbé optimista a célhoz közelí csomópontok esetén. Amikor ez megtörténik, különösképpen nagy keresési terekben, a második legjobb út a legjobb úttá válhat, és a keresésnek vissza kell lépnie, hogy ezt az utat tudja követni. minden meggondolás megfelel az IMA\* egy-egy iterációjának és az elfejejtett csomópontok számos újból kifejtését teszi szükségessé, hogy a keresés képes legyen a legjobb utat visszaállítani, és azt egy csomóponnal meghosszabbítani.

Az A\*-hoz hasonlóan az RLEK is optimális algoritmus, feltéve, hogy a *h(n)* heurisztikus függvény elfogadható. Tárkomplexitása  $O(bd)$ , az időkomplexitását azonban nehezebb meghatározni. Ez függ a heurisztika pontosságától és attól is, mennyire gyakran változik a csomópontok kifejtése közben a legjobb út. Mind az IMA\*, mind az RLEK elvileg ki vannak téve a gráfokban való kereséssel kapcsolatos exponenciális komplexitásnövekedésnek (lásd 3.5. alfejezet), tekintettel arra, hogy az ismétlődő állapotok jelenlétével csak az aktuális úton képesek ellenőrizni. Előfordulhat így, hogy ugyanazt az állapotot többször is kifejtik.

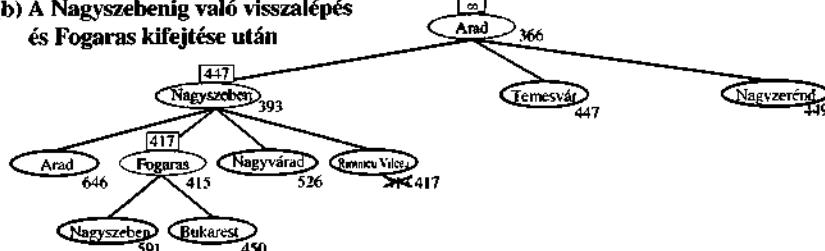
Az IMA\*-nak és az RLEK-nek az a bája, hogy *nincs* memóriát használ. Az egyes iterációk között az IMA\* egyetlen számot, az aktuális *f*-költség korlátot tárolja el. Az RLEK több információt tárol el a memóriában, azonban csak  $O(bd)$  memóriát használ. Még ha több memória is állna a rendelkezésre, az RLEK-nek nincs módja ezt kihasználni.

Észerűnek tűnik tehát az összes, rendelkezésre álló memóriát használni. Az erre képes két algoritmus az MA\* (memóriakorlátozott A\*) és az EMA\* (egyszerűsített MA\*). Itt az EMA\*-t írjuk le, ami – mitagadás – az egyszerűbb. Az EMA\* az A\* módjára halad a legjobb levelet kifejtve, amíg a memória be nem telik. Ezen a ponton a keresési fához új csomópontot hozzáadni nem képes, hacsak egy régit nem töröl ki. Az EMA\* mindenkor legrosszabb – a legmagasabb *f*-értékű – csomópontot hagyja ki. Majd, mint az RLEK, az elfejejtett csomópont értékét a szülőjéhez továbbítja. Ily módon egy elfejejtett részfa elője tudja a részfa legjobb útjának az értékét. Ezzel az információval az EMA\* csak akkor fejti ki újra a fát, ha kimutatta, hogy minden más út rosszabb-

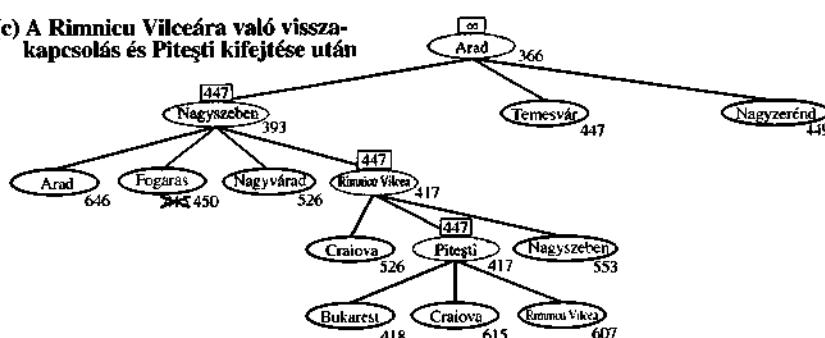
(a) Arad, Nagyszeben  
és Rimnicu Vilcea kifejtése után



**(b) A Nagyszebenig való visszalépés  
és Fogaras kifejtése után**



**(c) A Rimnicu Vilceára való visszakapcsolás és Piteşti kifejtése után**



**4.6. ábra.** Az RLEK lépései, miközben a legrövidebb utat keresi Bukarest felé. A mindenkorai rekurzív hívás f-korlát értéke a mindenkorai aktuális csomópont felett látható. (a) A Rimnicu Vilceán át vezető utat az algoritmus addig követi, amíg az aktuális legjobb levél (Pitești) értéke nem lesz rosszabb, mint a legjobb alternatív út értéke (Fogaras). (b) A rekurzió visszalép és az elfelejtett alfa legjobb levélértékét (417) Rimnicu Vilceánál feljegyezzük. Majd Fogaras kifejtése következik 450-nel, mint a legjobb levélértékkel (c) A rekurzió visszalép. Az elfelejtett alfa legjobb levélértékét (450) Fogarasnál jegyezzük fel. Következik Rimnicu Vilcea kifejtése. Ezúttal, mivel a legjobb alternatív út (Temesváron keresztül) 447-be kerül, a kifejtés folytatódik Bukarest felé.

nak tűnik, mint az elfelejtett út. Más szóval, ha az  $n$  csomópont minden utódját elfelejtjük, nem tudjuk, hogy  $n$ -ből merrefelé lehetne menni, tudni fogjuk azonban, hogy mennyire érdemes egyáltalán bárholva is menni az  $n$ -ből kiindulva.

A teljes algoritmus túlságosan bonyolult, hogy itt írunk róla,<sup>5</sup> azonban egy finom vonását érdemes megírni. Azt mondunk, hogy az EMA\* a legjobb levelet fejti ki és a legrosszabb levelet felelje el. Mi van akkor, ha minden levélnél ugyanaz az f-értéke?

<sup>5</sup> Az algoritmus vázlatos leírása a könyv első kiadásában megtalálható.

Az algoritmus ekkor ugyanazt a csomópontot kifejtésre is, és elhagyásra is kiválaszthatná. Az EMA\* ezt a problémát úgy oldja meg, hogy a kifejtésre a *legújabb* csomópontot választja, és a *legrégebbi* csomópontot törli. Ez a kettő ugyanaz a csomópont csak akkor lehet, ha csak egy levél van. Ebben az esetben az aktuális keresési fa egyetlen útból áll a gyökéről a levélig, ami kitölti a teljes memóriát. Ha a levél nem egy célcsomópont, akkor még ha a célhoz vezető optimális úton fekszik is, ez a cél az adott memóriával nem érhető el. Következésképpen a csomópontot ugyanúgy el lehet dobni, mintha nem is lenne követője.

Az EMA\* teljes, ha van egyáltalán elérhető megoldás – azaz ha  $D$ , a legsekélyebb célcsomópont mélysége kevesebb, mint a memória nagysága (csomópontokban kifejezve). Optimális, ha van elérhető optimális megoldás, másképpen az algoritmus a legjobb elérhető megoldással tér vissza. Gyakorlatilag az EMA\*-t messze a legjobb általánosan használatos algoritmusnak lehet tekinteni az optimális megoldások megkeresésére, különösképpen ha az állapottér egy gráf, a lépésköltség nem egyenletes, és a csomópont-kifejtés drága a nyitott és zárt listák karbantartásának pótlólagos overheadjéhez képest.

Nagyon nehéz problémák esetén azonban az EMA\* sokszor kénytelen folyamatosan oda-vissza kapcsolatba a lehetséges megoldási utak között, amelyekből csak kevés fér be a memóriába (hasonló ez a diszkalapú memória lapozó rendszer *vergődési* (*thrashing*) problémájára). Az ugyanazon csomópontok ismételt újrakifejtéséhez szükséges extra idő azt jelenti, hogy azok a problémák, amelyeket a végtelen memóriájú A\* gyakorlatilag meg tudna oldani, az EMA\* számára kezelhetetlenek. Ez azt jelenti, hogy *memóriakorlát a problémát a számítási idő szempontjából kezelhetetlennek teheti*. Bár a memória és az idő közötti kompromisszum magyarázatára nincs elmélet, úgy tűnik, hogy ettől a problémától megmenekülni nem lehet. Egyetlen kiút elvetni az optimalitás követelményét.



## Tanulunk, hogy jobban keressünk!

Az eddigiekben néhány rögzített stratégiát mutattunk be – szélességi, mohó legjobbat-először stb. keresés –, amelyeket számítógépes szakemberek terveztek. Meg tudná-e *tanulni* egy ágens, hogy jobban keressen? A válasz igen, és a módszer alapját egy fontos fogalom, a **metaszintű állapottér** (**metalevel state space**) adja. A metaszintű állapottér minden állapota az **objektumszintű állapottérbeli** (**object-level state space**) – amilyen például Románia – keresőprogram egy belső (számítási) állapotának felel meg. Az A\* algoritmus belső állapota például az aktuális keresési fa. A metaszintű állapottér minden cselekvése egy számítási lépés, amely a belső állapotot változtatja meg. Az A\* esetén például minden számítási lépés kifejt egy levelet, és a követőit a fához adjva hozzá. A 4.3. ábra, amely az egyre növekvő keresési fák sorozatát mutatja, értelmezhető lehetne úgy, hogy a metaszintű állapottérben egy utat mutat, ahol az út minden állapota egy objektumszintű keresési fa.

A 4.3. ábrán az út öt lépésből áll, Fogaras kifejtését is beleszámítva, ami nem volt valami hasznos. Nehezebb problémák esetén sok ilyen téves lépésre lehet számítani, és a **metaszintű tanulási algoritmus** (**metalevel learning**) ilyen tapasztalatokból tanulhat, hogy elkerülje a haszontalan részfák kifejtését. Az ilyen tanuláshoz alkalmas módszereket a 21. fejezetben tárgyaljuk. A tanulás célja a problémamegoldás **totális költségének** (**total cost**) a minimalizálása, kompromisszumot kövve a számítási kiadások és az útköltség között.

## 4.2. HEURISZTIKUS FÜGGVÉNYEK

Ebben a részben a 8-as kirakójátékhöz keresünk heurisztikus függvényeket, hogy rávíláglásunk a heurisztikus függvények általános természetére.

A 8-as kirakójáték a legrégebbi heurisztikus keresési feladatok egyike. Mint azt a 3.2. alfejezetben láttuk, a játék lényege, hogy a számoszott lapkákat vízszintesen és függőlegesen az üres helyre tolva a kiinduló állásból a célállásba jussunk (lásd 4.7. ábra).

Egy átlagos megoldás véletlen módon generált 8-as kirakójáték példányok esetén kb. 22 lépésből áll. Az elágazási tényező nagyjából 3 (amikor az üres lapka középen van, akkor négy, amikor a sarokban van, akkor kettő, és amikor valamelyik szélénél középső pozícióban van, akkor pedig három mozgatás lehetséges). Ebből adódóan egy 22 mélységig menő kimerítő keresés közelítőleg  $3^{22} \approx 3,1 \times 10^{10}$  állapotot vizsgálna meg. Az ismétlődő állapotok nyilvántartásával ezt a számot 170 000-ed részére le lehet csökkenteni, mert csak  $9!/2 = 181\,440$  különböző elérhető elrendezés létezik (lásd 3.4. feladat). Ez egy kezelhető szám, azonban ugyanez a szám a 15-ös kirakójáték esetén már durván  $10^{13}$ , így a következő teendő egy jó heurisztika megkeresése. Amennyiben a legrövidebb megoldásokat akarjuk megtalálni A\* bevetésével, olyan heurisztikus függvényre van szükségünk, ami soha sem becsüli túl a célállapot eléréséhez szükséges lépések számát. Az ilyen heurisztikák keresésének a 15-ös kirakójáték esetén nagy a múltja. Íme két lehetséges függvény:

- $h_1$  = a rossz helyen lévő lapkák száma. A 4.7. ábrán a 8 lapkából egyik sincs a helyén, így a kiinduló állapotban  $h_1 = 8$  lenne.  $h_1$  elfogadható heurisztikus függvény, mivel nyilvánvaló, hogy minden rossz helyen lévő lapkát legalább egyszer mozgatni kell.
- $h_2$  = a lapkáknak a saját célhelyeiktől mért távolságaik összege. Mivel a lapkákat nem lehet átlók mentén mozgatni, az általunk kiszámított távolság a vízszintes és függőleges távolságok összege lesz. Ezt néha háztömb- (city block distance) vagy Manhattan-távolságnak (Manhattan distance) is szokás nevezni.  $h_2$  szintén elfogadható heurisztikus függvény, mivel minden egyes mozgatással egy lapkát csak egy lépéssel lehet közelebb vinni a célhöz. A kiinduló állapotban az 1–8 lapkára számított Manhattan-távolság:

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Ahogy remélük, egyik sem becsüli túl a megoldás igazi költségét, ami 26.

7	2	4
5		6
8	3	1

Kiinduló állapot

	1	2
3	4	5
6	7	8

Célállapot

4.7. ábra. A 8-as kirakójáték egy tipikus feladata. A megoldás 26 lépés hosszú.

## A heurisztikus függvény pontosságának hatása a megoldás hatékonyságára

A heurisztikus függvény minősítésének egyik lehetséges módja a  $b^*$  effektív elágazási tényező (effective branching factor) megadása. Amennyiben az A\* algoritmus által kifejtett összes csomópont száma egy adott problémára  $N$ , és a megoldás mélysége  $d$ , akkor  $b^*$  annak a  $d$  mélységű kiegyensúlyozott fának az elágazási tényezőjével egyezik meg, amely  $N + 1$  csomópontot tartalmazna. Ebből adódóan:

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Például ha az A\* algoritmus egy 5 mélységen fekvő megoldást 52 csomópont kifejtésével talál meg, akkor az effektív elágazási tényező 1,92. Az effektív elágazási tényező a problémaesetek függvényében változhat, megfelelően nehéz problémák esetén azonban általában nagyjából állandó. Így a  $b^*$  kisszámú problémahalmazon végzett kísérleti mérése jó fogázót adhat a heurisztikus függvény általánosságban vett használhatóságáról. Egy jól megtervezett heurisztikus függvény effektív elágazási tényezője 1 köröli érték, ami lehetővé teszi felettesebb nagy problémák megoldását is.

A  $h_1$  és a  $h_2$  heurisztikus függvények teszteléséhez véletlenszerűen generáltunk 1200 problémapéldányt, 2–24 mélységű megoldással (100-100 példányt minden páros szám esetére), és megoldottuk azokat a  $h_1$  és a  $h_2$  heurisztikus függvényeket alkalmazva az A\* fakereső algoritmussal, illetve a nem informált iteratívan mélyülő keresési algoritmussal is. A 4.8. ábra minden egyes stratégiára megadja az átlagosan kifejtett csomópontok számát és az effektív elágazási tényezőt. Az eredmények azt mutatják, hogy  $h_2$  jobb, mint  $h_1$ , és hogy a nem informált keresés mindenkoránál sokkal rosszabb. A 14 hosszúságú megoldások esetén az A\*  $h_2$ -vel 30 000-szer hatékonyabb, mint a nem informált iteratívan mélyülő keresés.

$d$	Keresési költség			Effektív elágazási tényező		
	IMK	$A^*(h_1)$	$A^*(h_2)$	IMK	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	364404	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18 094	1219	–	1,48	1,28
24	–	39 135	1641	–	1,48	1,26

4.8. ábra. A keresési költség és az effektív elágazási tényező összehasonlítása ITERATÍV-MÉLYÜLŐ-KERESÉS és A\* esetén  $h_1$  és  $h_2$  használatával. Az adatokat különböző hosszúságú megoldásokra a 8-as kirakójáték 100 egyedét megoldva átlagolva kaptuk.

Feltehetjük magunknak a kérdést, hogy vajon a  $h_2$  mindenkorban jobb-e, mint a  $h_1$ ? A válasz: igen. A két heurisztikus függvény definíciójából jól látszik, hogy minden  $n$  csomópontra  $h_2(n) \geq h_1(n)$ . Azt mondjuk, hogy  $h_2$  dominálja  $h_1$ -et. A domináció közvetlenül átvihető a hatékonyságra: a  $h_2$ -t használó  $A^*$  algoritmus kevesebb csomópontot fog kifejteni, mint a  $h_1$ -et használó (talán csak az  $f(n) = C^*$  tulajdonságú néhány csomópontot kivéve). Ezt az alábbi egyszerű gondolatmenettel mutathatjuk meg. Idézzük fel a 142. oldalon tett észrevételünket, miszerint minden csomópont kifejtésre kerül, amelyre  $f(n) < C^*$ . Ezzel egyenértékű az az állítás, mely szerint minden csomópont kifejtésre kerül, amelyre  $h(n) < C^* - g(n)$ . Mivel azonban  $h_2$  minden csomópontra legalább akkora, mint  $h_1$ , így minden olyan csomópontot, amit kifejt a  $h_2$ -t alkalmazó  $A^*$  algoritmus, kifejti a  $h_1$ -et alkalmazó  $A^*$  algoritmus is, de  $h_1$  alkalmazása más csomópontok kifejtését is okozhatja. Ebből adódóan mindenkorban jobb nagyobb értékeket adó heurisztikus függvényeket alkalmazni, amíg nem becsüljük túl a valódi költséget, és a heurisztika számítási ideje nem túlságosan nagy.

## Elfogadható heurisztikus függvények kitalálása

Láttuk, hogy mindenkorban  $h_1$  (a nem a helyükön lévő lapkák száma), mindenkorban pedig  $h_2$  (Manhattan-távolság) egész jó heurisztikus függvények a 8-as kirakójáték problémájához, és hogy  $h_2$  jobbnak bizonyult. De hogyan is állt elő a  $h_2$ ? Egy számítógép számára vajon lehetséges-e mechanikusan megalkotni ilyen heurisztikus függvényeket?

A  $h_1$  és  $h_2$  a 8-as kirakójátékban a fennmaradó út hosszát becsülik, azonban a játék *egyszerűsített változatánál* a tökéletesen pontos úthossz értékét adják meg. Ha a játék szabályait úgy módosítanánk, hogy egy lapka bárhol áthelyezhető legyen, nemcsak a szomszédos mezőkre, akkor  $h_1$  pontosan megadná a legrövidebb megoldáshoz vezető lépések számát. Hasonlóan, ha egy lapkát bármelyik szomszédos mezőre átmozgathatnánk, még akkor is, ha az adott mezőn már van egy másik lapka, akkor  $h_2$  megadná a legrövidebb megoldás pontos lépésszámát. Az olyan problémát, amelyben az operátorokra kevesebb megkötést teszünk, mint az eredeti problémában, **relaxált problémának** (*relaxed problem*) nevezzük. A *relaxált probléma optimális megoldásának költsége egy elfogadható heurisztika az eredeti problémára*. A heurisztika elfogadható, mert az eredeti probléma optimális megoldása definíciószerűen megoldása a relaxált problémának is, és így legalább olyan költséges, mint a relaxált probléma optimális megoldása. Mivel a számított heurisztika a relaxált problémára egy pontos költség, teljesítenie kell a háromszög egyenlőtlenséget, és ebből kifolyólag **konziszens** (lásd 140. oldal).

Amennyiben a probléma megfogalmazása formális nyelven adott, akkor a relaxált problémákat automatikusan is elő lehet állítani.<sup>6</sup> Például ha a 8-as kirakójáték operátorait az alábbi módon írjuk le:

Egy lapka az  $A$  mezőről a  $B$  mezőre mozgatható,  
ha  $A$  és  $B$  szomszédosak, és a  $B$  mező üres

<sup>6</sup> A 8. és a 11. fejezetben erre a feladatra alkalmas formális nyelveket frunk le. Manipulálható formális leírásokkal a relaxált problémák konstruálása automatizálható. Egyelőre természetes nyelvet használunk.

egy vagy több feltétel törlésével három relaxált problémát hozhatunk létre:

- (a) Egy lapka az A mezőről a B mezőre mozgatható, ha A és B szomszédosak.
- (b) Egy lapka az A mezőről a B mezőre mozgatható, ha a B mező üres.
- (c) Egy lapka az A mezőről a B mezőre mozgatható.

Az (a)-ból a  $h_2$  (Manhattan-távolság) vezethető le. A magyarázat az, hogy  $h_2$  helyes eredményt adna, ha minden lapkát sorra elmozgatnánk a saját célhelyére. A (b)-ból származtatott heurisztikával a 4.9. feladatban foglalkozunk. A (c)-ból a  $h_1$  (a nem a helyükön lévő lapkák száma) vezethető le, mert ez lenne a helyes eredmény, ha a lapkákat a célpozíciójukba egy lépéssel el lehetne mozgatni. Vegyük észre, hogy lényeges, hogy e módszer által generált relaxált problémákat lényegében keresés nélkül meg lehet oldani, mert a relaxált szabályok a probléma 8 független részproblémává történő dekompozícióját teszik lehetővé. Ha a relaxált problémát nehéz megoldani, akkor a kapcsolatos heurisztikus értékek számítása drágának fog bizonyulni.<sup>7</sup>

Az ABSOLVER nevű program a „relaxált probléma” módszer és más egyéb módszer alkalmazásával (Prieditis, 1993), képes a probléma definíciójából automatikusan heurisztikus függvényeket generálni. Az ABSOLVER egy, az eddigieknel jobb heurisztikus függvényt hozott létre a 8-as kirakójáték megoldására, és ez a program találta meg az első használható heurisztikus függvényt a híres Rubik-kocka kirakásához.

Az új heurisztikus függvények előállításának egyik problémája, hogy felettesebb nehéz felismerni a „nyilvánvalóan legjobb” heurisztikus függvényt. Ha egy problémához adottak a  $h_1, \dots, h_m$  elfogadható heurisztikus függvények, és egyik sem dominálja a többöt, melyiket kell választanunk? Mint az majd kiderül, nem kell választanunk. Az alábbi formulával a lehető legjobbat kaphatjuk meg:

$$h(n) = \max\{h_1(n), \dots, h_m(n)\}$$

Az így megkonstruált összetett heurisztikus függvény mindig azt a függvényt használja, amelyik az adott csomópontra a legfontosabb. Mivel az alkotóelemként felhasznált heurisztikus függvények minden elfogadhatók, ezért  $h$  is elfogadható. Ugyancsak könnyű bizonyítani, hogy  $h$  konzisztens. Továbbá  $h$  dominálja az összes, benne alkotóelemként felhasznált heurisztikus függvényt.

Elfogadható heurisztikus függvényeket származtathatunk az adott probléma részproblémájának (**subproblem**) megoldási költségéből is. A 4.9. ábra például a 4.7. ábrán látható 8-as kirakójáték egy részproblémáját mutatja. A részprobléma lényege az 1, 2, 3 és 4 lapkák helyükre való mozgatása. Világos, hogy e részprobléma optimális megoldásának költsége a teljes probléma megoldásának költségét alulról korlátozza. Bizonyos esetekben ez a függvény a Manhattan-távolságnál sokkal pontosabb.

A mintaadatházból (**pattern databases**) háttérében húzódó ötlet az, hogy tároljuk el az egyes részprobléma esetekhez tartozó pontos megoldási költségeket – ebben az esetben a négy lapka és az üres hely minden konfigurációjához (jegyezzük meg, hogy a másik négy lapka helye a részprobléma megoldása szempontjából közömbös, mozgásai azonban a költségbe bele fognak számítani). A probléma teljes állapotához tartozó

<sup>7</sup> Vegyük észre, hogy a tökéletes heurisztikát könnyítszertel megkaphatnánk, ha megengednék  $h$ -nak a teljes szélességi keresés lefuttatását „stíkában”. A heurisztikus függvény pontossága és számítási ideje között így kompromisszumot kell kötni.

*	2	4
*		*
*	3	1

Kiinduló állapot

	1	1
3	4	*
*	*	*

Célállapot

**4.9. ábra.** A 4.7. ábrán látható 8-as kirakójáték eset egy részproblémája. A feladat az, hogy 1, 2, 3 és 4 lapkát a helyes pozícióba juttassuk el, nem törödvé azzal, hogy a többi lapkával mi fog történni.

elfogadható  $h_{AB}$  heurisztikus függvényt ezek után keresés közben úgy számítjuk ki, hogy a megfelelő részprobléma-konfigurációt az adatbázisból kikeressük. Maga az adatbázis úgy lett megtervezve, hogy a célállapottól visszafelé keresve minden új minta költségét feljegyeztük. Ennek a keresésnek a költsége a sok egymás után jövő problémaeset-re kivetítve amortizálódik.

Az 1-2-3-4 lapka megválasztása nyilván tetszőleges, az adatbázist az 5-6-7-8 és a 2-4-6-8 stb. lapkákra is meg tudnánk konstruálni. minden adatbázis egy elfogadható heurisztikát szolgáltat, és ahogy korábban elmagyaráztuk, ezen heurisztikákat kombinálni lehet, maximális értékükkel számolva. Egy ilyen típusú kombinált heurisztika a Manhattan-távolságánál sokkal pontosabb. A 15-ös kirakójáték véletlenszerűen generált egyedeire a kifejtett csomópontok számát egy 1000-es tényezővel lehet csökkenteni.

El lehetne tűnődni azon, hogy az 1-2-3-4 és az 5-6-7-8 adatbázisokból kinyert heurisztikus függvényeket nem lehetne-e összeadni, mivel úgy tűnik, a két részproblémában nincs fedés. A válasz nemleges, mert egy adott állapotban az 1-2-3-4 és az 5-6-7-8 részproblémák megoldásai majdnem biztosan osztoznak közös lépéseken – igen valószínűtlen, hogy az 1-2-3-4 lapkákat a helyükre el lehet mozdítani, anélkül hogy az 5-6-7-8-hoz ne nyúlnánk hozzá, és fordítva. És mi lenne, ha ezeket a lépéseket nem is számolnának be? Azaz nem az 1-2-3-4 részprobléma teljes költségével számolnánk, hanem csak az 1-2-3-4 lapka lépéseiit vennénk figyelembe. Könnyű ilyenkor rájönni, hogy a két költség összege még mindig a teljes probléma megoldási költségének egy alsó korlátja. Ez a **diszjunkt mintaadatbázis (disjoint pattern database)** háttérében húzódó ötlet. Az ilyen adatbázisok használatával lehetséges válik a 15-ös kirakójátékot milliszekundumok alatt megoldani, a Manhattan-távolsághoz képest a generált csomópontok száma egy 10 000-es tényezővel kisebb. A 24-es kirakójáték esetén kb. egymilliós gyorsítást lehet elérni.

A diszjunkt mintaadatbázisok jól működnek a csúszólapka-játékok esetén, mert a problémát úgy lehet felbontani, hogy egy-egy lépés csak egy részproblémára van hatással, hiszen egyszerre csak egy lapkát mozgatunk. A Rubik-kocka problémára ilyen bontást elvégezni nem lehet, mert minden mozgás 8, 9 vagy 26 kockát érint. Nem tudjuk egyelőre, hogy az ilyen problémák számára a diszjunkt mintaadatbázisokat hogyan lehetne definiálni.

## A heurisztikus függvény tanulása tapasztalatból

A  $h(n)$  heurisztikus függvénytől elvárjuk, hogy az  $n$  csomópontbeli állapotból kezdve becsülje a megoldás költségét. Hogyan lenne képes egy ágens egy ilyen függvény megalkotására? Egy megoldást az előbbi részben adtunk meg – ami nevezetesen egy olyan relaxált problémának a kitalálása, amihez egy optimális megoldást könnyű találni. Egy másik megoldás a tapasztalatból való tanulás. A „tapasztalaton” itt azt értjük, hogy nagyon sok 8-as kirakójátékot kell megoldani. A 8-as játék minden optimális megoldása egy példát jelent, amiből  $h(n)$  tanulható. minden példa egy, a megoldási úton elhelyezkedő állapotból és az onnan számított valós megoldási költségből áll. Ilyen példák alapján **induktív tanulási algoritmus (inductive learning)** segítségével egy olyan  $h(n)$  függvényt konstruálhatunk, amely (szerencsével) képes megjósolni a megoldás költségét a keresés során felbukkanó más állapotok esetén is. Ennek módszertanát, legyenek ezek neurális hálók, döntési fák vagy más módszerek, a 18. fejezet mutatja be (a 21. fejezetben leírt megerősítéses tanulás szintén alkalmazható).

Az induktív tanulási módszerek akkor működnek a legjobban, ha nem az állapot nyers leírását, hanem inkább az állapot kiértékeléséhez releváns **jellemzőket (feature)** kapják bemenetként. A „nem a helyén lévő lapkák száma” jellemző segítség lehet, ha az aktuális állapotnak a céltól vett távolságát szeretnénk megjósolni. Hívjuk ezt a jellemzőt  $x_1(n)$ -nek. Vehetnénk a 8-as kirakójáték 100 véletlenszerűen generált konfigurációját, és gyűjthetnék statisztikákat a megoldás aktuális költségéről. Esetleg azt találnánk, hogy ha  $x_1(n)$  értéke 5, akkor a megoldás átlagos költsége 14 és így tovább. Ilyen adatok birtokában  $x_1(n)$  értékéből jósolni lehetne  $h(n)$  értékét. Persze több jellemzőt is lehetne használni. Egy másik jellemző,  $x_2(n)$  lehetne például „azon szomszédos lapkapárok száma, melyek a célállapotban is szomszédosak”. Hogyan lehetne  $x_1(n)$ -et és  $x_2(n)$ -et összekombinálni  $h(n)$  jóslása érdekében? Egy szokásos ötlet a lineáris kombináció használata:

$$h(n) = c_1x_1(n) + c_2x_2(n)$$

A  $c_1$  és  $c_2$  konstansokat módosítani lehet, hogy a megoldási költség aktuális adataira legjobban illeszkedjenek. Feltételezhetően  $c_1$ -nek pozitívnak,  $c_2$ -nek pedig negatívnak kellene lennie.

## 4.3. LOKÁLIS KERESŐ ALGORITMUSOK ÉS OPTIMALIZÁCIÓS PROBLÉMÁK

Az eddig látott keresési algoritmusokat arra terveztek, hogy a keresési tereket szisztematikusan táráják fel. A szisztematikusságot úgy érik el, hogy egy vagy több utat tartanak a memóriában, és azt is feljegyzik, hogy minden pontban az út mentén melyik alternatívát vizsgálták már meg, és melyiket nem. Amikor megtalálják a célt, a célhoz vezető út egyben a probléma *megoldását* is jelenti.

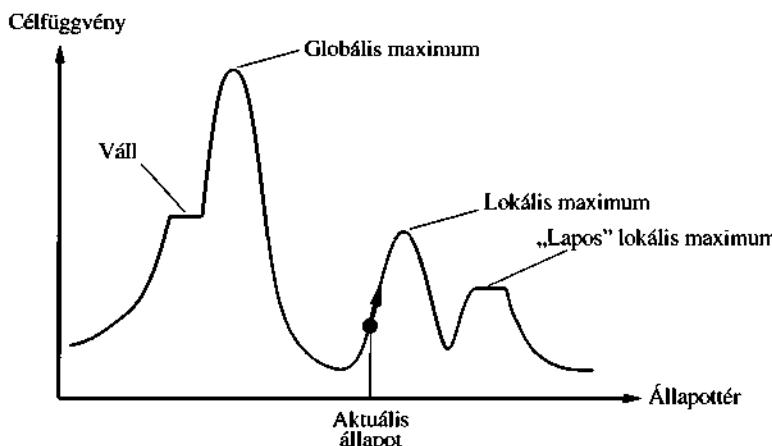
Számos problémában azonban a célhoz vezető út érdektelen. A 8-királynő problémában például (lásd 104–105. oldal) a királynők végleges konfigurációja számít, és nem az a sorrend, ahogyan az újabb királynőket felhelyezzük. A problémák ezen osztályába olyan fontos problémák tartoznak, mint a VLSI-tervezés, a gyári gépelrendezés, a gyá-

tási műveletek ütemezése, az automatikus programozás, a hírközlési hálózatok optimalizálása, a gépkociútvonal-tervezés és a portfóliómenedzsment.

Ha a célohoz vezető út nem számít, más algoritmusokra is gondolhatunk, amelyek az utakkal egyáltalán nem foglalkoznak. A lokális keresési algoritmusok (*local search*) csak egy aktuális állapotot (*current state*) vesznek figyelembe (a többszörös utak helyett) és általában csak ennek az állapotnak a szomszédjaira lépnek tovább. A keresés által követett utat tipikusan nem is tárolják el. Bár a lokális keresési algoritmusok nem szisztematikusak, két kulcsfontosságú előnyük van: (1) igen kevés – általában konstans mennyiségi – memóriát használnak, és (2) sokszor nagy vagy végtelen (folytonos) keresési térben elfogadható megoldást produkálnak ott, ahol a szisztematikus algoritmusok alkalmatlanok lennének.

A cél megfogalmazásán túl, a lokális keresési algoritmusok a tisztán **optimalizációs problémák** (*optimization problems*) megoldásában is hasznosak, ahol a cél a legjobb állapot megtalálása egy **célfüggvény** (*objective function*) értelmében. Számos optimalizációs probléma nem felel meg a 3. fejezetben bevezetett „standard” keresési modellnek. Így például a természet előállít egy célfüggvényt – a szaporodási fitnesset – amit úgy tűnik, a darwi evolúció igyekszik optimalizálni, e probléma esetén nem beszélhetünk azonban sem „céltesztről”, sem „útköltségről”.

A lokális keresés megértéséhez igen hasznosnak találjuk az **állapottér-felszínt** (*state space landscape*) (mint amilyen a 4.10. ábrán látszik). A felszínnek van „pontja” (amit az állapot definiál) és „magassága” (amit a heurisztikus vagy a célfüggvény értéke határozza meg). Ha a magasság a költséggel arányos, akkor a cél a legalacsonyabban fekvő völgyet – a **globális minimumot** – megtalálni. Ha a magasság a célfüggvénynek felel meg, akkor a cél a legmagasabb csúcs – a **globális maximum** – megtalálása (az egyikről a másikra előjel váltásával könnyen áttérhetünk). A lokális keresés ezt a felületet vizsgálja. Egy **teljes** lokális keresés mindenkor megtalálja a globális minimumot vagy maximumot. Egy **optimális** algoritmus mindenkor megtalálja a globális minimumot vagy maximumot.



**4.10. ábra.** Egy egydimenziós állapottér-felszín, ahol a magasság a célfüggvénynek felel meg. A cél a globális maximum megtalálása. A hegymászó keresés az aktuális állapotot módosítja a javulás irányában, ahogy ezt a nyíl is mutatja. A különböző topografikus jellemzőkkel a szövegben foglalkozunk.

## Hegymászó keresés

A **hegymászó keresési algoritmust** (**hill-climbing**) a 4.11. ábra mutatja. A keresés egyszerűen csak egy ciklus, ami minden javuló értékek felé – azaz felfelé – lép. Az algoritmus megáll, amikor felér a csúcsra, ahol nincsenek már magasabb értékű szomszédjai. Az algoritmus nem tart nyilván keresési fát, ezért a csomópontot leíró adatszerkezetnek csak az állapotot és a célfüggvény értékét kell nyilvántartania. A hegymászó keresés nem néz előre az aktuális állapotot közvetlenül követő szomszédokon túl. Egy kicsit arra hasonlít, mintha a Mount Everest csúcsát szeretnénk megtalálni sűrű ködben és emlékezetkihagyásban szenvedve.

```

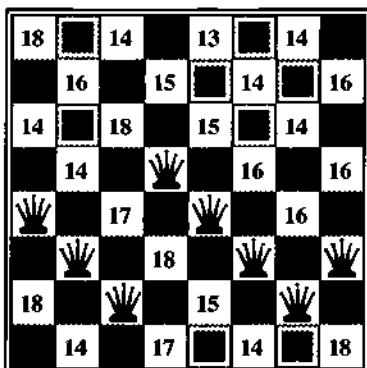
function HEGYMÁSZÁS(probléma) returns egy lokális maximumot jelentő állapot
  inputs: probléma, egy probléma
  local variables: aktuális, egy csomópont
                    szomszéd, egy csomópont

  aktuális  $\leftarrow$  CSOMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma])
  loop do
    szomszéd  $\leftarrow$  az aktuális legnagyobb értékű követő csomópontja
    if ÉRTÉK[szomszéd]  $\leq$  ÉRTÉK[aktuális] then return ÁLLAPOT[aktuális]
    aktuális  $\leftarrow$  szomszéd
  end
```

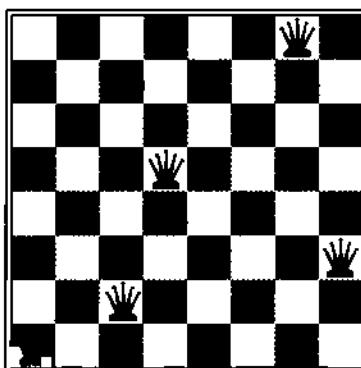
**4.11. ábra.** A hegymászó keresési algoritmus (a legmeredekebb emelkedő (steepest ascent) változat) a lokális keresés alapvető módszere. minden lépésben az aktuális csomópontot a legjobb szomszédjával cseréli le, ami ebben a változatban a legmagasabb ÉRTÉK-ű szomszédot jelenti. Ha azonban egy *h* heurisztikus költségbecslést alkalmazunk, ez akkor a legalacsonyabb *h*-jú szomszéd lenne.

A hegymászó keresés illusztrálására a 104. oldalon bevezetett **8-királynő problémát** (**8-queens problem**) fogjuk használni. A lokális keresési algoritmusok tipikusan a **teljes állapot leírással** (**complete-state formulation**) élnek, ahol minden állapotban a táblán 8 királynő helyezkedik el, oszloponként egy. Az állapotátmenet-függvény minden olyan lehetséges állapotot visszaad, amit úgy kapunk, hogy egy királynót ugyanabban az oszlopban egy másik mezőre mozgatunk (így minden állapotnak  $8 \times 7 = 56$  követője van). A *h* heurisztikus függvény a közvetlenül vagy közvetetten módon egymást támadó királynőpárok száma. E függvény globális minimuma 0, ami csak a tökéletes megoldásban érhető el. A 4.12. (a) ábra egy *h* = 17 értékű állapotot mutat. Az ábra az összes követőnek az értékét is mutatja, ahol a legjobb követő *h* = 12. Ha a legjobból több van, a hegymászó algoritmusok tipikusan véletlen módon sorsolnak belőlük egyet.

A hegymászó keresést néha **mohó lokális keresésnek** (**greedy local search**) is hívják, mert egy jó követő állapotot megragad, anélkül hogy megfontolná, merre lenne érdemes továbbmenni. Bár a mohóság a hét főből egyike, a mohó algoritmusok sokszor igen jól teljesítnek. A hegymászás gyakran igen gyorsan halad a megoldás felé, mert egy rossz állapoton általában nagyon könnyű javítani. Így például a 4.12. (a) ábrán látható állapotból indulva öt lépés elegendő, hogy elérjük a 4.12. (b) ábrán látható állapotot, melynek költsége *h* = 1, és amely igen közel van egy megoldáshoz. Sajnos a hegymászás gyakran megakad az alábbi problémák miatt:



(a)



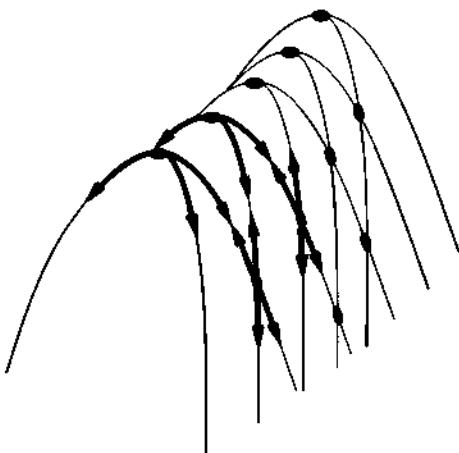
(b)

**4.12. ábra.** (a) A 8-királynő probléma  $h = 17$  heurisztikus költségbecslésű állapota, ahol minden olyan követő állapot  $h$  értékét megadtuk, amikor a királynőt a saját oszlopában mozgatjuk. A legjobb lépéseket bejelöltük. (b) Egy lokális minimum a 8-királynő térben. Az állapot  $h$  értéke egységes, azonban minden követőnek magasabb a költsége.

- **Lokális maximumok:** egy lokális maximum egy csúcs, amely minden szomszédjánál magasabb, de a globális maximumnál alacsonyabb. A hegymászó algoritmusok, ha egy lokális maximum közelébe érnek, kénytelenek a csúcs felé tartani, ott azonban nincs tovább, és megakadnak. Ezt a problémát sematikusan a 4.10. ábra mutatja. Konkrétabban, a 4.12. (b) ábra állapota egy tényleges lokális maximum (azaz egy lokális minimum a  $h$  költségre nézve), bármely királynő bármilyen lépése csak ront a helyzetén.
- **Hegygerincek (ridges):** egy hegycsúcsot a 4.13. ábra mutat. A hegycsúcs egy olyan lokális maximum sorozatot eredményez, ahol egy mohó algoritmusnak igen nehéz navigálnia.
- **Fennsík (plateaux):** a fennsík az állapottérnek egy olyan területe, ahol a kiértékelő függvény gyakorlatilag lapos. Lehet ez egy lapos lokális maximum, amelyből nincs tovább felfelé, de lehet egy váll (shoulder), ahonnan még lehetséges az előrehaladás (lásd 4.10. ábra). Egy hegymászó keresés képtelen lehet arra, hogy egy fennsíkról megtalálja a kivezető utat.

Az algoritmus minden esetben elér egy pontot, ahonnan már nem tud továbblépni. Egy véletlenül módon generált 8-királynő állapotból kiindulva a legmeredekebb emelkedő hegymászó algoritmus az esetek 86%-ában megakad, a problémaeseteknek csupán 14%-át oldja meg. Az algoritmus gyorsan dolgozik, átlagosan 4 lépést tesz, amikor sikerkel jár, és 3 lépést, amikor megakad. Ez egyáltalán nem is olyan rossz a  $8^8 \approx 17$  millió állapotot tartalmazó állapottérben.

A 4.11. ábra algoritmusa megáll, ha fennsíkra ér, ahol a legjobb követőnek is ugyanaz az értéke, mint az aktuális állapotnak. Nem lenne-e jó ötlet mozgásban maradni – oldallépéseket (sideways move) megengedve annak reményében, hogy a fennsík igazából egy váll, mint amilyent a 4.10. ábra mutat? A válasz általában igen, azonban óvatosan kell eljárnunk. Ha oldallépéseket mindenkor megengedünk, akkor ha felfelé haladás nincs, végig a hurokba kerülünk minden olyan esetben, amikor az algoritmus olyan lapos lokális



**4.13. ábra.** Annak illusztrálása, hogy a gerinc miért jelent nehézséget a hegymászó keresés számára. Az átlapotok rácshálózata (sötét körök) rá van illesztve a balról jobbra emelkedő gerincre, egy olyan lokális maximum sorozatot eredményezve, ahol a maximumok nincsenek közvetlen módon egymással összekapcsolva. minden egyes lokális maximumból az összes lehetséges cselekvés a lejtőn lefelé mutat.

maximumot talál, amely nem váll. Gyakori megoldás az egymás után alkalmazott oldallépések számát korlátozni. A 8-királynő problémában például megengedünk 100 egymás utáni oldallépést. Ezzel a hegymászás által megoldott esetek arányát 14%-ról 94%-ra emelhetjük. A sikerek azonban ára van. Átlagosan 21 lépés hosszú minden sikerrel megoldott eset, és 64 lépés hosszú a kudarc.

A hegymászó algoritmus számos változatát fejlesztették ki. A **sztochasztikus hegymászó keresés (stochastic hill climbing)** a felfelé mutató irányokból véletlen módon választ. A választás valószínűsége a felfelé mutató irány meredekségével változhat. Az algoritmus általában a legmeredekebb emelkedő módszernél lassabban konvergál, egyes állapotfelszíneken azonban képes jobb megoldást találni. Az **elsőnek-választott hegymászó algoritmus (first-choice hill climbing)** sztochasztikus hegymászó keresést használ, a követőket véletlen módon addig generálva, amíg az az aktuális állapotnál nem lesz jobb. Ez jó stratégia, ha egy állapotnak sok (például több ezer) követője van. Ennek vizsgálatára invitál a 4.16. feladat.

Az eddig leírt hegymászó algoritmusok nem teljesek – sokszor kudarcozt vallanak a cél megkeresésében, mert egy lokális maximumba beragadtak. A **véletlen újraindítású hegymászás (random-restart hill-climbing)** az ismert közmanódás szerint jár el: „Ha nem megy elsőre, csinál újra.” Véletlenül generált kiinduló állapotokból<sup>8</sup> hegymászó keresést végez, amíg célba nem ér. Az algoritmus 1-hez tartó valószínűséggel teljes annál a triviális oknál fogva, hogy előbb-utóbb a célállapotot kezdőállapotként is fogja generálni. Ha minden hegymászó keresés  $p$  valószínűséggel sikeres, a véletlen újraindítások várható száma  $1/p$ . A 8-királynő problémában, ha oldallépéseket nem engedünk meg,  $p \approx 0.14$ , így a cél megtalálásához átlagosan 7 iterációra van szükség (6 kudarc és

<sup>8</sup> Véletlen állapotot generálni egy implicit módon specifikált állapotterén azonban önmagában is igen kemény probléma lehet.

1 siker). A várható lépésszám a sikeres iteráció költsége valamint  $(1-p)/p$ -szerese a kudarc költségének, durván 22 lépés. Ha oldallépéset is engedünk, átlagosan  $1/0,94 \approx 1,06$  iterációra és  $(1 \times 21) + (0,06/0,94) \times 64 \approx 25$  lépéstre van szükség. A 8-királynő probléma számára a véletlen újraindítású hegymászó keresés valóban hatékony. Még hárommillió királynő mellett is ez a megközelítés egy percen belül talál megoldást.<sup>9</sup>

A hegymászás sikere nagyban függ az állapotter „felszínének” alakjától: ha azon csak néhány lokális maximum és fennsík található, akkor a véletlen újraindítású hegymászó algoritmus gyorsan meg fog találni egy jó megoldást. Egy valódi problémához egy olyan felszín tartozik, ami leginkább egy sündisznócsaládhoz hasonlít egy sima padlón, ahol minden sündisznótűske csúcsát egy további miniatűr sündisznó lakja és így a végtelenséig. Ha a probléma NP-teljes, akkor minden bizonnyal exponenciálisan sok lokális maximummal rendelkezik, melyekben beragadhatunk. Ennek ellenére általában kisszámú újraindítás után már elfogadhatóan jó megoldást lehet találni.

## Szimulált lehűtés

A hegymászó keresés, amely soha nem indul „lefelé a lejtőn” a kisebb értékű (vagy nagyobb költségű) állapotok felé, garantáltan nem teljes, mert egy lokális maximumban beragadhat. Ezzel ellentétben a tisztán véletlen vándorlás – azaz a követők halmazából egyenletesen véletlen módon sorsolt követőre való átlépés – teljes, de hihetetlenül nem hatékony. Értelmes dolognak tűnik a hegymászás és a véletlen vándorlás valamiféle ötvözése, hogy mind a teljességet, mind a hatékonyságot megtarthassuk. Egy ilyen algoritmus a szimulált lehűtés (simulated annealing). A kohászatban a lehűtés (annealing) a fémeket, illetve az üveget edző, keményítő folyamat, amikor azokat magas hőmérsékletre felmelegítjük, majd fokozatosan lehűtjük, lehetővé téve, hogy az anyag alacsony energiájú kristályos állapotba kerüljön. Hogy a szimulált lehűtést megértsük, a hegymászásról térjünk át a gradiens leereszkedésre (gradient descent) (azaz a költség minimalizálására), és képzeljük el, hogy az a feladatunk, hogy egy hepehupás asztalon egy pingponglabdát a legmélyebb szakadékba juttassunk. Ha a labdát gurulni hagyjuk, egy lokális minimumba kerül. Ha a felületet megrázunk, a labdát kiugraszthatjuk a lokális minimumból. A trükk az, hogy olyan erősen kell megrázni a felületet, hogy a labda a lokális minimumból kikerüljön, de mégsem annyira erősen, hogy a labda a globális minimumból kiugorjon. A szimulált lehűtés olyan megoldás, hogy először erősen rázunk (azaz egy magas hőmérsékleten), majd fokozatosan csökkentjük a rázás intenzitását (vagyis csökkentjük a hőmérsékletet).

A szimulált lehűtés legbelső ciklusa (lásd 4.14. ábra) nagyon hasonlít a hegymászáshoz. A legjobb lépés megtétele helyett azonban egy véletlen lépést tesz. Ha a lépés javítja a helyzetet, akkor az minden végrehajtásra kerül. Ellenkező esetben az algoritmus a lépést csak valamilyen 1-nél kisebb valószínűséggel teszi meg. A valószínűség exponenciálisan csökken a lépés „rosszaságával” – azzal a  $\Delta E$  mennyiséggel, amivel a kiértékelő

<sup>9</sup> Luby (Luby és társai, 1993) azt bizonyította be, hogy egyes esetekben az a legjobb, ha a véletlen keresést egy konkrét rögzített időtartam letelte után újraindítjuk, és ez sokkal hatékonyabb lehet annál, mintha minden egyik keresést a végtelenséig hagynánk folytatóni. Az oldallépések tiltása vagy a számuk korlátozása ennek egy példája.

```

function SZIMULÁLT-LEHÚTÉS(probléma, lehűtési terv) returns egy megoldási állapot
  inputs: probléma, egy probléma
           lehűtési terv, egy leképzés időről „hőmérsékletre”
  local variables: aktuális, egy csomópont
                      következő, egy csomópont
                      T, egy „hőmérséklet”, ami a lefelé lépések valószínűségét szabályozza

  aktuális  $\leftarrow$  CSMÓPONTOT-LÉTREHOZ(KIINDULÓ-ÁLLAPOT[probléma])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  lehűtési terv[t]
    if T = 0 then return aktuális
    következő  $\leftarrow$  az aktuális egy véletlenszerűen kiválasztott követő csomópontja
     $\Delta E$  = ÉRTÉK[következő] - ÉRTÉK[aktuális]
    if  $\Delta E > 0$  then aktuális  $\leftarrow$  következő
    else aktuális  $\leftarrow$  következő csak  $e^{-\Delta E/T}$  valószínűséggel
  
```

**4.14. ábra.** A szimulált lehűtés keresési algoritmus. A hegymászó keresés egy olyan változata, ahol lefelé tartó lépések is megengedettek. A lefelé tartó lépéseket a lehűtési terv elején inkább, az idő műlásával egyre kevésbé fogadjuk el. A lehűtési terv, mint az eljárás bemenete, a *T* értékeit mint időfüggvényt adja meg.

függvény értéke romlott. A valószínűség a *T* „hőmérséklet” csökkenésével is csökken. A „rossz” lépések az indulásnál *T* magasabb értékeinél valószínűbbek, *T* csökkenésével egyre valószínűlenebbé válnak. Be lehet bizonyítani, hogy ha a *hűtési karakterisztika* *T* értékeit kellően lassan csökkenti, az algoritmus 1-hez tartó valószínűséggel a globális minimumban köt ki.

A szimulált lehűtési algoritmust először elterjedten VLSI-elrendezési problémák megoldására használták a '80-as évek elején. Azóta széles körben alkalmazzák ipari termelés ütemezésére és egyéb nagy volumenű optimalizációs feladatokra. A 4.16. feladat a szimulált lehűtési algoritmus és a véletlen újraindítású hegymászó algoritmus teljesítményének összehasonlítását kéri az *n*-királynő problémán.

## Lokális nyaláb keresés

Mindössze egyetlen csomópontot tartani a memóriában elégé extrém reagálásnak tűnik a memóriakorlát problémájára. A **lokális nyaláb keresés** (local beam search) algoritmus<sup>10</sup> nem egy, hanem *k* állapotot követ nyomon. Az algoritmus *k* véletlen módon generált állapottal indul. minden lépésben a *k* állapot mindegyikének összes követőit kifejt. Ha ezek valamelyike egy cél, az algoritmus leáll. Egyébként a teljes listából kiválasztja a legjobb *k* követőt, és ezt az eljárást ismétli.

Első látásra a *k* állapotú lokális nyaláb keresés nem tűnik másnak, mint a *k* véletlen újraindítás parallel futtatása a szekvenciális futtatás helyett. Valójában a két algoritmus igen különböző. A véletlen újraindítású algoritmusban minden keresési folyamat a többitől függetlenül fut le. A **lokális nyaláb keresési algoritmusban a *k* parallel keresési szál megosztja az információt**. Például ha az egyik állapot számos jó követőt generál,

<sup>10</sup> A lokális nyaláb keresés a nyalábkeresés (beam search) algoritmus egy adaptációja, ami viszont egy út-alapú algoritmus.

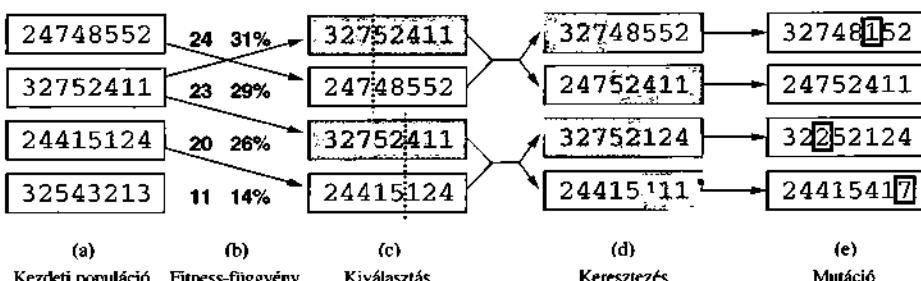
és a többi  $k - 1$  mindenrosszabban, akkor az eredmény az, mintha az első állapot megüzenné a többinek: „Gyertek át ide, itt zöldebb a fű!” Az algoritmus gyorsan abbahagyja az eredménytelen keresésekét, és az erőforrásait oda viszi, ahol a legnagyobb előrehaladás érzékelhető.

A lokális nyaláb keresés a legegyszerűbb formájában a  $k$  állapot közötti változatosság hiányától szenvedhet. Az állapotok gyorsan koncentrálódnak a tér egy kicsi részében, amitől a keresés csak kevessel lesz több, mint a hegymászó keresés egy drága változata. A problémát a sztochasztikus nyaláb keresés (stochastic beam search) segíti megoldani, ami analóg a sztochasztikus hegymászó kereséssel. A  $k$  legjobb követő megválasztása helyett az algoritmus a  $k$  követőt véletlen módon választja ki, ahol egy adott követő kiválasztásának valószínűsége az állapot értékének növekvő függvénye. A sztochasztikus nyalábkeresés egy kicsit hasonlít a természetes kiválasztódásra, ahol egy „állapot” (szervezet) „követői” (utódai) a következő generációt az állapot „értéke” (fitness) alapján népesítik be.

## Genetikus algoritmusok

A **genetikus algoritmus**, GA (genetic algorithm) a sztochasztikus nyaláb keresés egy olyan variánsa, ahol a követő állapotokat nem egy állapot módosításával, hanem két szülő állapot összekombinálásával állítjuk elő. A természetes kiválasztódás analógiája itt is ugyanaz, mint a sztochasztikus nyaláb keresés esetén, azzal a különbséggel, hogy most az utódlétrehozásnak a szexuális és nem az aszexuális mechanizmusról van szó.

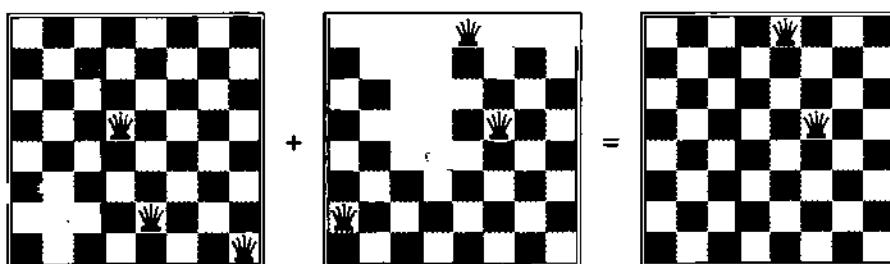
A nyalábkereséshez hasonlóan, a GA is  $k$  véletlen módon generált állappal indul, aminek **populáció** (population) a neve. minden állapotot vagy **egyedet** (individual) egy véges ábécé fölött értelmezett füzér képvisel – leggyakrabban egy 0-ból és 1-ekből álló füzér. A 8-királynő állapotnak például a 8 királynő pozícióját kell specifikálnia, mindegyik egy 8 négyzetet tartalmazó oszlopban, így  $8 \times \log_2 8 = 24$  bitre van szükség. Az állapotot 8 számjeggyel is lehetne jellemzni, melyek mindegyike az 1 – 8 tartományból való (később látni fogjuk, hogy a két kódolás viselkedése eltérő). A 4.15. (a) ábra egy olyan populációt mutat be, mely négy, a 8-királynő állapotait reprezentáló 8 számjegyes füzérből áll.



4.15. ábra. A genetikus algoritmus. Az (a)-beli kezdeti populációt (b)-ben fitness-függvény alapján rangsoroljuk, aminek eredményét a (c)-ben látható reprodukáló párok adják. A létrehozott utódok a (d)-ben láthatók, melyekre még hat a mutáció (e).

Az állapotok következő generációjának az előállítása a 4.15. (a)–(e) ábrán követhető. A (b)-ben minden állapotot a kiértékelő függvény vagy (GA-terminológiában) a **fitness-függvény** (**fitness function**) alapján rangsoroljuk. Egy fitness-függvénynek a jobb állapotokra magasabb értékeket kell visszaadnia, így a 8-királynő problémában fitness-függvényként a *nemtámádó* királynőpárok számát használjuk. Ennek értéke egy megoldás esetén 28. A négy állapot értéke 24, 23, 20 és 11. A genetikus algoritmus e konkrét változatában a szaporodásra való kiválasztás valószínűsége a fitness-értékkel egyenesen arányos. A százalékos mennyiségeket a fitness-értékek mellett adjuk meg.

A (c)-ben két párt véletlenszerűen választunk ki reprodukcióra, a (b)-beli valószínűségeknek megfelelően. Figyeljük meg, hogy egy egyedet kétszer, egy másikat viszont egyszer sem választottunk ki.<sup>11</sup> minden keresztezendő párnál egy **keresztezési** (**crossover**) pontot választunk a füzérbeli pozíciók közül. A 4.15. ábrán a kereszteződési pontok az első párnál a harmadik számjegy után, a második párnál az ötödik számjegy után vannak.<sup>12</sup>



**4.16. ábra.** A 4.15. (c) ábrán látható első két szülőnek és a 4.15. (d) ábrán látható két utódnak megfelelő 8-királynő állapotok. Az árnyalt oszlopok a kereszteződés során elvesznek, a nem árnyalt oszlopok megmaradnak.

A (d)-ben az utódok generálását látjuk, a szülő füzéreket a keresztezási pontoknál keresztezve. Az első pár első gyereke például az első szülőtől az első három számjegyét kapja és a másik szülőtől a többöt. A második gyerek pedig az első három számjegyét a második szülőtől kapja és az első szülőtől a többöt. Az ennél a reprodukciós lépésnél szereplő 8-királynő állapotokat a 4.16. ábra mutatja. A példa azt a tényt szemlélteti, hogy ha a két szülő állapot igen különböző, a keresztezés eredménye mindenkor szülőtől igen távol eshet. Sokszor előfordul, hogy a keresési folyamat elején a populáció elégő változatos, és a keresztezés (a szimulált lehűtéshez hasonlóan) nagy léptekkel halad előre, majd később, ha az egyedek többsége már igen hasonlít egymásra, kisebb lépések jönnek.

Végül az (e)-ben a füzér minden elemét valamelyen kis független valószínűsséggel **mutációnak** (**mutation**) vetjük alá. Az első, a harmadik és az ötödik utódban egy-egy

<sup>11</sup> Ennek a kiválasztási szabálynak számos változata van. Kimutatható, hogy a **selejelezés** (**culling**) módszere, ahol egy küszöb alá eső minden egyedet eldobunk, a véletlen változatnál gyorsabban konvergál (Baum és társai, 1995).

<sup>12</sup> Itt számít a kódolás. Ha a 8 számjegyes kódolás helyett a 24 bites kódolást használjuk, a kereszteződési pontnak 2/3-ad esélye van, hogy egy számjegy belséjébe essen, ami ennek a számjegynek lényegében tetszőleges mutációját eredményezi.

```

function GENETIKUS-ALGORITMUS(populáció, FITNESS-Fv) returns egy egyed
  inputs: populáció, az egyedek halmaza
    FITNESS-Fv, egy egyed fitness-értékét mérő függvény
  repeat
    új_populáció  $\leftarrow$  üres halmaz
    loop for i from 1 to MÉRET(populáció) do
      x  $\leftarrow$  VÉLETLEN-KIVÁLASZTÁS(populáció, FITNESS-Fv)
      y  $\leftarrow$  VÉLETLEN-KIVÁLASZTÁS(populáció, FITNESS-Fv)
      utód  $\leftarrow$  REPRODUKCIÓ (x, y)
      if (kis véletlen valószínűség) then utód  $\leftarrow$  MUTÁCIÓ(utód)
      utód hozzáadása az új_populáció-hoz
    populáció  $\leftarrow$  új_populáció
  until valamelyik egyednek kellően magas a fitness-értéke, vagy már lejárt az idő
  return a populáció legjobb egyede, a FITNESS-Fv-nek megfelelően

function REPRODUKCIÓ(x, y) returns egy egyed
  inputs: x, y a szülőegyedek

  n  $\leftarrow$  HOSSZ(x)
  c  $\leftarrow$  1 és n közötti véletlen szám
  return HOZZÁFŰZ(RÉSZFÜZÉR(x, 1, c), RÉSZFÜZÉR(y, c + 1, n))

```

**4.17. ábra.** Egy genetikus algoritmus. Ez az algoritmus ugyanaz, mint amelyet a 4.15. ábrán mutattunk, egy kivétellel. Ebben a jobban elterjedt változatban a két szülő reprodukciójára nem két, hanem csak egy utódot hoz létre.

számjegyet mutáltunk. A 8-királynő problémában ez annak felel meg, hogy egy királynő véletlen módon kiválasztunk, és egy, az oszlopban szintén véletlen módon kiválasztott mezőre áthelyezünk. Az ezeket a lépéseket implementáló algoritmust a 4.17. ábra mutatja.

A sztochasztikus nyaláb kereséshez hasonlóan a genetikus algoritmus is kombinálja a hegymászó tendenciát, a véletlen feltárást és a parallel keresési szálak közötti információcsereit. A genetikus algoritmus elsődleges előnye, ha ilyen egyáltalán létezik, a keresztezéstől származik. Matematikailag azonban kimutatható, hogy ha kezdetben a genetikus kód pozícióit véletlen sorrendben permutáljuk, a keresztezés semmilyen előnyt nem jelent. Intuitíve, az előny a keresztezés azon képességből jön, hogy a keresztezés képes hasznos funkciókat teljesítő nagy betűblokkok kombinálására, melyek egymástól függetlenül alakultak ki, emelve ezzel a keresénél a granuláltság szintjét. Az első három királynőt például a 2, 4 és 6 pozícióba helyezve (ahol nem támadják egymást), egy hasznos blokk alakul ki, amit más blokkokkal kombinálva egy megoldás nyerhető.

Ennek működését a genetikus algoritmusok elmélete a **séma** (*schema*) fogalommal magyarázza. A séma egy olyan részfüzér, amelyben bizonyos pozíciók nem specifikáltak. Így például a 246\*\*\*\*\* séma az összes olyan 8-királynő állapotot leírja, ahol az első három királynő a 2., 4. és 6. pozícióban van. A sémára illeszkedő füzérek, mint például a 24613578 füzér, a séma **példányai** (*instances*). Kimutatható, hogy ha egy séma példányainak átlagos fitness-értéke az átlag felett van, akkor idővel a populációban a sémához tartozó példányok száma nőni fog. Világos, hogy ez a hatás elenyésző lesz,

ha a szomszédos báteknek semmi közük egymáshoz, mert kevés olyan tömör blokk lesz, ami konzisztens módon előnyt fog jelenteni. A genetikus algoritmus a legjobban akkor működik, ha a sémkák a megoldás értelmes komponenseinek felelnek meg. Ha például a füzér egy antenna reprezentációja, akkor a sémkák képviselhetik az antenna egyes komponenseit, például a reflektorokat és a deflektorokat. Egy jó komponensnek nagy valószínűséggel különböző konstrukciók sokaságában is jónak kell lennie. Ez azt sugallja, hogy a genetikus algoritmusok sikeres alkalmazása a reprezentáció gondos kialakítását igényli.

A gyakorlatban a genetikus algoritmusoknak nagy hatása volt az olyan optimalizációs problémákra, mint például az áramkör elrendezéstervezés és a gyártósor-ütemezés. Jelenleg nem világos, hogy a genetikus algoritmusok vonzereje a hatékonyságuk vagy az evolúció elméletében rejlö esztétikus eredetük eredménye. Sok munka van még hártra, hogy azokat a körfelülményeket azonosítsuk, amikor a genetikus algoritmusok igazán jól teljesítenek.

## 4.4. LOKÁLIS KERESÉS FOLYTONOS TEREKBEN

A 2. fejezetben bemutattuk a diszkrét és a folytonos környezet közötti különbséget, arra is rámutatva, hogy a valóvilág-beli környezetek többsége folytonos. Az eddig leírt algoritmusok közül azonban egyik sem képes a folytonos állapottereket kezelní. Az állapotátmennet-függvény az esetek többségében végtelen számú állappittal térne vissza! Ez a részfejezet egy *nagyon rövid* bevezetőt ad néhány olyan lokális keresési technikához, melyek folytonos téren keresik az optimális megoldást. A téma irodalma óriási. Sok alaptechnika már a 17. században napvilágot látott, Newton és Leibniz kalkulusának kifejlődését<sup>13</sup> követően. E technikákhoz a könyvben több helyen fogunk folyamodni, beleértve a tanulásról, a látásról és a robotikáról szóló fejezeteket. Röviden, mindenhol, ahol a valós világgal foglalkozunk.

### Evolúció és keresés

Az evolúció elméletét Charles Darwin az *On the Origin of Species by Means of Natural Selection* c. művében (Darwin, 1859) dolgozta ki. A központi gondolat igen egyszerű: a reprodukcióban (**mutációként ismert**) eltérések fordulnak elő, és azokat a következő generációk nagyjából a reprodukciós fitnessre gyakorolt hatásuk arányában megtartják.

A darwini elméletet annak ismerete nélkül dolgozták ki, hogy a szervezetek tulajdonságai hogyan öröklődnek és módosulnak. E folyamatokat irányító valószínűségi törvényeket első ízben Gregor Mendel szerzetes azonosította (Mendel, 1866), aki borsóval kísérletezett, saját szavaival mesterséges meghibridizációval alkalmazva. Sokkal később Watson és Crick feltárták a DNS-molekula szerkezetét és ábécéjét – AGTC (adenin, guanin, timin, citozin) – (Watson és Crick, 1953). A standard modellben a betűszekvenciában változás pontmutáció és „keresztezés” révén áll be (ahol az utód DNS-e a szülei DNS-ek hosszú részleteinek kombinálásával jön létre).

A lokális keresési algoritmus analógiájáról írtunk már. A sztochasztikus nyaláb keresés és az evolúció közötti alapvető különbség a **szexuális reprodukció használata**, ahol az utódokat

<sup>13</sup> A többváltozós kalkulus és a vektoraritmetika ismerete hasznos ennek a részfejezetnek az olvasásához.

több egyedből hozzuk létre, és nem csak egyből. Az evolúció tényleges mechanizmusai azonban sokkal gazdagabbak, mint amit a genetikus algoritmus lehetővé tenne. A mutációhoz például az átfordítás, a duplikálás, a DNS nagy szegmenseinek a mozgatása is hozzátarozik. Egyes vírusok a DNS-t az egyik szervezetből veszik és egy másik szervezetbe beillesztik. Vannak átvilágító gének is, amelyek nem tesznek mást, csak egy génállományon belül többezreszer lemásolják magukat. Olyan gének is vannak, amelyek a potenciális reprodukciós partnereknél ezeket a géneket nem tartalmazó sejteket megmérgezik, növelte így a sokszorozódás esélyét. A legfontosabb tény, hogy *maguk a gének tartalmazzák annak a mechanizmusnak a kódját*, amely által a génállomány reprodukálódik, és egy szervezetted alakul át. A genetikus algoritmusokban ezek a mechanizmusok különálló programok részei, vagyis a manipulált füzérekben nem jelennek meg.

A darwini evolúció igencsak kis hatékonyágú mechanizmusnak tűnhet, hiszen vakon létrehozott kb.  $10^{45}$  szervezetet, anélkül hogy a keresési heurisztikán egy csöppet is javított volna. Darwin előtt 50 ével a különben neves francia természettudós Jean Lamarck egy olyan evolúció elméletet javasolt (Lamarck, 1809), ahol egy szervezet az élete során, adaptációja révén megszerzett tulajdonságokat is képes az utódoknak átadni. Az ilyen mechanizmus hatékony lenne, de úgy tűnik, a természetben nem fordul elő. Sokkal később James Baldwin látszólag hasonló elmélettel állt elő (Baldwin, 1896), hogy a szervezet élete alatt megtanult viselkedés növelhetné az evolúció sebességét. Lamarck elméletével ellentétben a Baldwin-elmélet a darwini evolúcióval teljesen konziszens, hiszen alapja egy szelekciós nyomás, amely olyan egyedekre hat, amelyek lokális optimumokat találtak a genetikus felépítésük által engedélyezett viselkedések között. A korszerű számítógépes szimulációk alátámasztják a „Baldwin-effektus” valós voltát, feltéve, hogy a „közönséges” evolúció olyan szervezeteket képes kialakítani, amelyknél a belső hatékonyági mérték a saját tényleges fitness-értékükkel valahogy korrelál.

Kezdjük egy egyszerű példával. Tegyük fel, hogy valahol Romániában három új repülőteret szeretnénk létesíteni úgy, hogy a városoknak (lásd 3.2. ábra) a hozzájuk legközelebb eső repülőtérrhez való távolságösszegük legyen minimális. Az állapotteret ekkor a repülőterek  $(x_1, y_1)$ ,  $(x_2, y_2)$  és  $(x_3, y_3)$  koordinátái definiálják. Ez egy *hatdimenziós* tér, azt is mondhatjuk, hogy az állapotokat hat változó (*variable*) definiálja (általánoosságban az állapotokat a változók  $n$ -dimenziós x vektora definiálja). Ebben a térben való mozgás a térképen a repülőterek áthelyezésének felel meg. Az  $f(x_1, y_1, x_2, y_2, x_3, y_3)$  célfüggvényt, ha a legközelebbi városok már megvannak, bármely állapot esetére viszonylag könnyű kiszámítani, általánosságban azonban igen nehéz ezt leírni.

A folytonosság problémái egyszerűen elkerülhetők, ha az egyes állapotok szomszéd-ságát diszkretizáljuk. Egyszerre például csak egy repülőteret, csak x vagy y irányban egy rögzített  $\pm \delta$  lépéssel lehet áthelyezni. Ez 6 változó mellett, minden állapotban 12 követőt eredményez. Ehhez az előbb megismert bármelyik lokális keresési algoritmust alkalmazhatjuk. A sztochasztikus hegymászó keresést és a szimulált lehűtést közvetlenül, a tér diszkretizálása nélkül is alkalmazhatjuk. Ezek az algoritmusok a követőket véletlen módon választják ki, amelyek megvalósítása δ hosszúságú véletlen vektorok generálásával lehetséges.

Sok módszer található, melyek megkísérik a felszín gradiensét (**gradient**) felhasználni a maximum megtalálásához. A célfüggvény gradiense egy  $\nabla f$  vektor, amely a legmeredekebb emelkedő nagyságát és irányát adja meg. Problémánk esetén:

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

Egyes esetekben a maximumot a  $\nabla f = 0$  egyenlet megoldásával találhatjuk meg (ezt megtéhetnénk például akkor, ha csak egy repülőteret helyeznénk el; a megoldás az összes város koordinátáinak számtani közepe lesz). Sok esetben azonban ezt az egyenletet zárt alakban nem lehet megoldani. Három repülőtér esetén például a gradiens ki-fejezése attól függ, hogy az adott állapotban mely városok esnek legközelebb az egyes repülőterekhez. Ez azt jelenti, hogy a gradiens *lokálisan*, és nem *globálisan* tudjuk számítani. Ennek ellenére még mindig folyamodhatunk a legmeredekebb emelkedő hegymászáshoz a pillanatnyi állapotot az

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

képpel frissítve, ahol  $\alpha$  egy kis konstans. Más esetekben nem biztos, hogy a célfüggvény differenciálható formában áll a rendelkezésünkre – például a repülőterek helyeit egy konkrét esetben esetleg egy nagyméretű gazdasági szimulációs szoftvercsomag futtatásával határozzák meg. Az ilyen esetekben az ún. **empirikus gradiens (empirical gradient)** meghatározásához folyamodhatunk, minden koordináta mentén egy kis pozitív és negatív változáshoz kiszámítva a választ. Az empirikus gradiens keresés ugyanaz, mint a legmeredekebb emelkedő hegymászás az állapottér diszkrétált változatában.

Az „ $\alpha$  egy kis konstans” kifejezés mögött az  $\alpha$ -t beállító módszerek óriási választéka rejlik. Az alapprobléma az, hogyha  $\alpha$  túl kicsi, túlságosan sok lépéstre van szükség. Ha  $\alpha$  túl nagy, a keresés könnyűszerrel túllő a maximumon. A **vonalkeresés (line search)** módszere e problémát úgy kísérli megoldani, hogy a pillanatnyi gradiens irányát – általában  $\alpha$  ismételt megduplázásával – meghosszabbítja, amíg  $f$  nem kezd újra csökkeni. Az a pont, ahol ez megtörténik, lesz az új aktuális állapot.

Sok probléma esetén a legjobb algoritmus a jó öreg **Newton–Raphson-módszer** (Newton, 1671; Raphson, 1690). Ez a gyökhelykeresés, azaz a  $g(\mathbf{x}) = 0$  alakú egyenletek megoldásának általános módszere. Működésének alapja az  $x$  gyök új becslésének számítása a Newton-formula szerint:

$$\mathbf{x} \leftarrow \mathbf{x} - g(\mathbf{x}) / g'(\mathbf{x})$$

$f$  maximumának vagy minimumának megkereséséhez olyan  $\mathbf{x}$ -et kell megkeresni, amire a *gradiens nulla* (azaz  $\nabla f(\mathbf{x}) = 0$ ). A Newton-formula  $g(\mathbf{x})$ -e így  $\nabla f(\mathbf{x})$  lesz, és a frissítési egyenlete az alábbi mátrixos formában

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

írható fel, ahol  $\mathbf{H}_f(\mathbf{x})$  a második deriváltak mátrixa (**Hesse-mátrix**), melynek  $H_{ij}$  elemeit a  $\partial^2 f / \partial x_i \partial x_j$  második deriváltak adják meg. Mivel a második deriváltak mátrixának  $n^2$  eleme van, a Newton–Raphson-módszer sokdimenziós terekben drága lesz, ami számos közelítő módszer kifejlesztéséhez vezetett.

A lokális keresési módszerek számára a lokális maximumok, a gerincek és a fennsíkok folytonos terekben ugyanúgy gondot jelentenek, mint diszkrét terekben. A véletlen újraindítás és a szimulált lehűtés alkalmazható (itt is), és gyakran segít. A sokdimenziós folytonos terek azonban terebélyes helyek, ahol könnyű elveszni.

Az utolsó téma, amivel érdemes futó ismeretséget kötni a **korlátozott optimalizálás (constrained optimization)**. Egy optimalizálási probléma korlátozott, ha a megoldás-

nak minden változójának értékeire nézve valamilyen kemény korlátozást kell teljesítenie. A repülőteres problémáinkban korlátozhatjuk például a helyszíneket, hogy a repülőterek Románián belül és szárazföldön (nem tavak közepén) helyezkedjenek el. A korlátozott optimalizálás nehézségei a korlátozások és a célfüggvény természetén múlnak. A legismertebb kategóriát a **lineáris programozási (linear programming)** problémák jelentik, ahol a korlátozások lineáris egyenlőtlenségek, amelyek egy *konvex* régiót képeznek, és ahol a célfüggvény szintén lineáris. A lineáris programozási problémákat változó számban polinomiális időben meg lehet oldani. Olyan problémákat is tanulmányoztak, ahol más típusú korlátozások és célfüggvények fordulnak elő. Ilyen problémák például a kvadratikus programozási feladat, a másodrendű kónikus programozási feladat stb.

## 4.5. ONLINE KERESŐ ÁGENSEK ÉS ISMERETLEN KÖRNYEZETEK

Eddig olyan ágensekre összpontosítottunk, amelyek **offline** keresési (**offline search**) algoritmusokat használnak. Egy teljes megoldást számítanak ki mielőtt a valós világba beteszik a lóbukat (lásd 3.1. ábra), majd a megoldást az érzékelések megvizsgálása nélkül végrehajtják. Ezzel ellentétben az **online kereső (online search)**<sup>14</sup> ágensek működésében a számítás és a végrehajtás **átlapolódik (interleaving)**: először végrehajtanak egy cselekvést, majd megfigyelik a környezetüket és kiszámítják a következő cselekvést. Az online keresés jó ötlet a dinamikus és a szemidinamikus környezetekben – olyan környezetekben, ahol büntetik a semmitteést és a túlságosan hosszú számításokat. Az online keresés még jobb ötlet sztochasztikus környezetekben. Általánosságban egy offline keresésnek egy exponenciálisan nagy, minden lehetséges történést figyelembe vévő eshetőségi tervvel kellene előállnia, míg az online keresés csak az aktuálisan meg-történteket veszi figyelembe. Például ajánlatos, ha egy sakköz ágens előbb megtesz az első lépést és csak ezután számítja ki a játszma teljes lefolyását.

Az online keresés szükségszerű ötlet a **felfedezési problémák (exploration problems)** esetén, ahol az állapotok és a cselekvések ismeretlenek az ágens számára. Egy ilyen tudatlan állapotban az ágensnek a cselekvéseit kísérletekként kell használnia, hogy megállapíthassa, mit tegyen a következő pillanatban, így a számítás és a cselekvés szükségszerűen átlapolódik.

Az online keresés kanonikus példája egy új épületben elhelyezett robot, melynek fel kell tárnia a környezetét, hogy felépíthesse azt a térképet, amit majd arra használ, hogy A-tól B-ig eljusson. A labirintusból való menekülési módszerek – az ókori hősök ellen-gedhetetlen képessége – szintén példái az online keresésnek. A térbeli feltárást azonban nem az egyedüli formája a feltárásnak. Gondoljunk egy újszülöttre: sok lehetséges cselekvéssel rendelkezik, de egyiknek sem tudja a hatását, és csak néhány közvetlenül elérhető állapotról van tudomása. Az, ahogyan egy újszülött fokozatosan felfedezi, hogy hogyan működik a világ, részben egy online keresési folyamat.

<sup>14</sup> Az „online” kifejezést minden használják a számítógépes tudományokban olyan folyamatok megjelölésére, amelyek a bemeneti adatokat azok bejövetelekor dolgozzák fel, és nem várnak addig, amíg a teljes bemeneti adathalmaz hozzáférhetővé válik.

## Online keresési problémák

Egy online keresési problémát egy tisztán csak számítást végző folyamat helyett csak egy olyan ágens tud megoldani, amely a cselekvéseit végrehajtja. Feltételezzük, hogy az ágens csak az alábbiakat tudja:

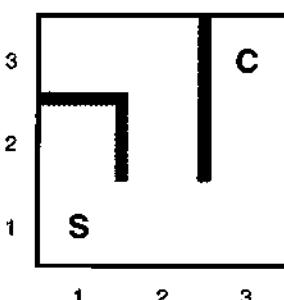
- **CSELEKVÉSEK( $s$ )**, amely az  $s$  állapotban engedélyezett cselekvések listáját adja vissza,
- A lépésköltség  $c(s, a, s')$  függvény – jegyezzük meg, hogy ez nem használható addig, amíg az ágens nem tudja, hogy  $s'$  az eredmény, és
- **CÉL-TESZT( $s$ )**.

Vegyük észre azt is, hogy az ágens *nem képes* egy állapot követőit másképpen elérni, mint úgy, hogy az adott állapotban az összes cselekvését kipróbálja. A 4.18. ábrán látható labirintusproblémában az ágens nem tudja, hogy a *Fel* cselekvés (1,1)-ből elvezeti őt az (1,2)-be, valamint ennek végeztével, hogy a *Le* cselekvés visszaviszi őt az (1,1)-be. Bizonyos alkalmazásokban a tudatlanság ezen szintje mérsékelhető – egy felfedező robot ismerheti a mozgási cselekvéseinek mechanizmusát, és csak az akadályok hollétéről nincs tudomása.

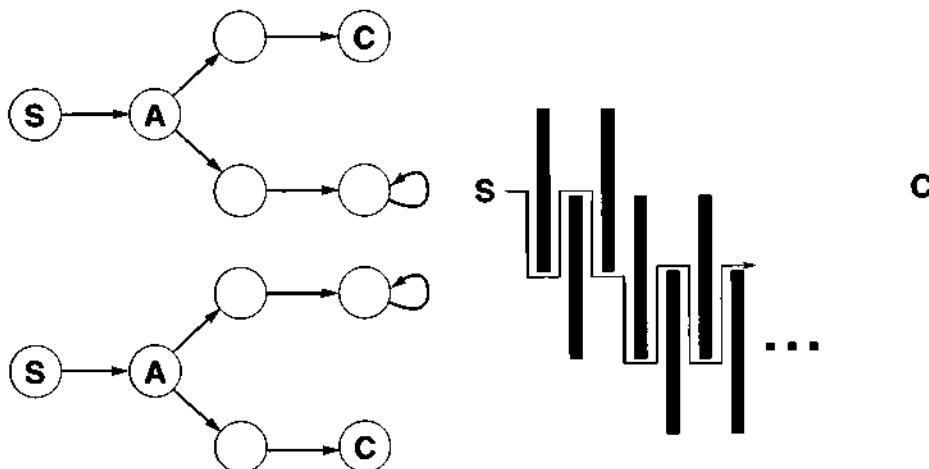
Feltételezzük, hogy az ágens minden képes felismerni azt az állapotot, amiben már járt, és hogy a cselekvései determinisztikusak (e két feltételezéstől a 17. fejezetben eltekintünk). Végül az ágens hozzáférhet egy  $h(s)$  elfogadható heurisztikus függvényhez, amely a pillanatnyi állapot és a célállapot távolságát becsüli. A 4.18. ábrán például az ágens tudhatja a célpozícióját, és képes lehet a Manhattan-távolsági heurisztika használatára.

Az ágens célja tipikusan az, hogy elérjen egy célállapotot, és eközben minimalizálja a költségeket (egy másik lehetséges cél egyszerűen az egész környezet feltárása). A költség az ágens által megtett tényleges út teljes költsége. Szokásos ezt a költséget azaz az útköltséggel összehasonlítani, amit az ágens követne, ha az egész keresési teret előre ismerné – azaz az aktuális legrövidebb úttal (vagy a legrövidebb teljes feltárással). Az online algoritmusok nyelvén ezt kompetitív aránynak (**competitive ratio**) nevezik, és azt szeretnénk, ha ez a lehető legkisebb lenne.

Bár ez ésszerű követelménynek tűnik, könnyű belátni, hogy egyes esetekben a legjobb elérhető kompetitív arány a végletes. Ha például bizonyos cselekvések irreverzibilisek, az online keresés esetleg egy olyan zsákutcába kerülhet, ahonnan a célállapot nem elérhető.



4.18. ábra. Egy egyszerű labirintusprobléma. Az ágens S-nél kezd és C-t kell elérnie, a környezetéről azonban semmit sem tud.



**4.19. ábra.** (a) Két olyan állapottér, amely az online kereső ágenst egy zsákutcába viheti. (b) Egy két-dimenziós környezet, amely arra készítheti az online kereső ágenst, hogy a célhoz legkevésbé hatékony utat kövesse. Akármit is választ az ágens, az ellenség elkorlásolja az útját egy másik hosszú, vékony fallal úgy, hogy a követett út sokkal hosszabb lesz, mint a legjobb lehetséges út.

Lehet, hogy az „esetleg” megfogalmazást nem találja kellően meggyőzőnek – végül is létezhetne olyan algoritmus, amely feltárás közben a zsákutcákban nem köt ki. A ki-jelentéstünk pontosabban megfogalmazva az, hogy *nincs olyan algoritmus, amely bármilyen állapottérben el tudná kerülni a zsákutcákat*. Nézzük meg közelebbről a 4.19. (a) ábra két zsákutcás állapotterét. Egy olyan online kereső algoritmus számára, amely az  $S$  és az  $A$  állapotokat már meglátogatta, a két állapottér *azonosnak* tűnik, tehát mindenket-tőben ugyanolyan döntéshez kell folyamodnia. Ez az **ellenség érv (adversary argument)** egy példája – el tudunk képzelní egy ellenséget, amely módosítja az állapotteret, miközben az ágens feltárja azt, és a célokat és a zsákutcákat tetszszerint átrendezí. A zsákutcák a robotfeltárás igazi nehézségei – lépcsőházak, felhajtók, szakadékok és a természetes terep minden fajtája irreverzibilis cselekvésekhez vezethetnek. Hogy elő-rehaladhassunk, egyszerűen feltételezzük, hogy az állapottér **biztonságosan feltárható (safely explorable)** – bizonyos célállapotok az összes elérhető állapottóból elérhetők. A visszafordítható cselekvéseket tartalmazó állapottereket, mint amilyenek a labirintusok és a kirakójátékok, irányítatlan gráfoknak lehet tekinteni, és nyilván biztonságosan feltárhatók.

Még a biztonságosan feltárható környezetekben sem biztosítható korlátozott kompetitív arány, ha léteznék benne korlátlan költségű utak. Ezt könnyű kimutatni irreverzibilis cselekvéseket tartalmazó környezetekben, de igaz marad visszafordítható cselekvések esetén is, ahogy ezt a 4.19. (b) ábra mutatja. Emiatt általános, hogy az online algoritmu-sok hatékonyságát nem a leg sekélyebben fekvő célállapot mélységevel, hanem az egész állapottérrel jellemzzük.

## Online kereső ágensek

Egy online kereső ágens minden cselekvés után érzékelni, hogy milyen állapotba került. Ebből az információból felépítheti környezetének térképét. Az aktuális térkép alapján eldönti, hogy legközelebb merre menjen. Ez az átlapolódó tervkészítés és végrehajtás azt jelenti, hogy az online kereső algoritmusok egészen mások, mint az előbb megismert offline algoritmusok. Az olyan offline algoritmusok, mint például az A\* képesek egy csomópont követőit kiszámítani a tér egy részében, majd azonnal egy másik csomóponttal foglalkozni a tér egy másik részében, mert a csomópontkifejtés inkább szimulált, mint valódi cselekvéseket takar. Egy online algoritmus ezzel szemben csak azt a csomópontot fejtheti ki, amelyben fizikailag tartózkodik. Hogy a következő csomópont kifejtése érdekében a fa keresztül-kasul való végignézsét elkerüljük, célszerűbbnek tűnik a csomópontokat *lokális* sorrendben kifejteni. Ezzel a tulajdonsággal éppen a mélységi keresés rendelkezik, mert (a visszalépést kivéve) a következő kifejtendő csomópont az előbb kifejtett csomópont gyereke.

Egy online mélységi kereső ágenst a 4.20. ábra mutat. Ez az ágens a térképet egy *eredmény*[ $a, s$ ] táblázatban tárolja, amely az  $s$  állapotban végrehajtott  $a$  cselekvés hatására előálló állapotot tartalmazza. Ha az aktuális állapotban marad még ki nem használt cselekvés, az ágens ezzel próbálkozik. A nehézség akkor áll be, ha az ágens egy állapotban már minden cselekvést kipróbált. Offline mélységi keresésnél ezt az állapotot a sorból egyszerűen elhagyjuk. Online keresésnél az ágensnek fizikailag kell visszalépnie. A mélységi keresésnél ez azt jelenti, hogy abba az állapotba kell visszakerülnie, amelyből az ágens legutóbb belépett a vizsgált állapotba. Ezt egy olyan táblázat karbantartásával lehet megoldani, amely minden állapot számára azokat az előző állapotokat

```

function ONLINE-MÉLYSÉGI-ÁGENS( $s'$ ) returns egy cselekvés
  inputs:  $s'$ , egy érzékelés, amely az aktuális állapotot azonosítja
  static: eredmény, cselekvésekkel és állapotokkal indexelt táblázat, kezdetben üres
    nemfeltárti, egy táblázat, amely minden meglátogatott állapotra listázza
      a még nem próbált cselekvéseket
    nemvisszalépetti, egy táblázat, amely minden meglátogatott állapotra listázza
      a még nem próbált visszalépéseket
   $s, a$ , az előbbi állapot és cselekvés, kezdetben nulla

  if CÉL-TESZT( $s'$ ) then return stop
  if  $s'$  egy új állapot then nemfeltárti[ $s'$ ]  $\leftarrow$  CSELEKVÉSEK( $s'$ )
  if  $s$  nem nulla then do
    eredmény[ $a, s$ ]  $\leftarrow$   $s'$ 
     $s$  hozzáadása a nemvisszalépetti[ $s'$ ] elejéhez
  if nemfeltárti[ $s'$ ] üres then
  if nemvisszalépetti[ $s'$ ] üres then return stop
  else  $a \leftarrow$  egy olyan  $b$  cselekvés, hogy eredmény[ $b, s'$ ] = POP(nemvisszalépetti[ $s'$ ])
  else  $a \leftarrow$  POP(nemfeltárti[ $s'$ ])
   $s \leftarrow s'$ 
  return  $a$ 

```

4.20. ábra. Egy mélységi feltárást használó online kereső ágens. Az ágens csak a kétirányú keressésekben alkalmazható.

listázza, melyekhez az ágens még nem lépett vissza. A keresés teljes, ha az ágens a viszszaléphető állapotokból kifut.

Azt javasoljuk, hogy az olvasó kövesse végig az ONLINE-MÉLYSÉGI-ÁGENS előrehaladását a 4.18. ábrán mutatott labirintus esetén. Könnyű észrevenni, hogy az ágens a térben minden csatlakozást legrosszabb esetben pontosan kétszer fog végigpásztázni. Feltárást esetében ez optimális eredmény. Egy cél megtalálása szempontjából azonban az ágens kompetitív aránya tetszőlegesen rossz lehet, ha egy hosszú túrára indul, holott a cél a kezdeti állapot közvetlen szomszedságában fekszik. Ezt a problémát az iteratívan mélyülő algoritmus online változata oldja meg. Egy homogén fát eredményező környezetben egy ilyen ágens kompetitív aránya egy kis konstans.

Az alkalmazott viaszalépési módszer miatt az ONLINE-MÉLYSEGI-ÁGENS csak olyan terekben működik, ahol minden cselekvés visszafordítható. Az általános állapotterekhez egy kicsit bonyolultabb algoritmusok léteznek, azonban ezek egyike sem garantál korlátosított kompetitív arányt.

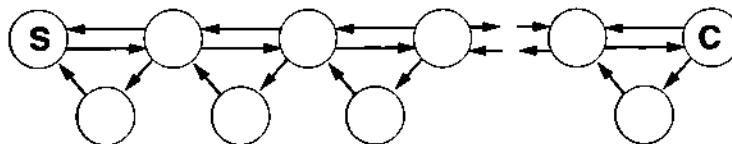
## Online lokális keresés

A mélységi kereséshez hasonlóan a **hegymászó keresésre** (*hill-climbing search*) jellemző a csomópontkifejtés lokalitása. A tény az, hogy mivel a hegymászó keresés a memóriában csak egy aktuális állapotot tart, ez az algoritmus már egy online keresési algoritmus! A legegyszerűbb formájában azonban nemigen használatos, mert az ágenst a lokális maximumokban bennrekedve hagyja, ahonnan nincs hová mennie. Ráadásul a véletlen újraindítást sem lehet használni, mert az ágens nem képes magát egy új állapotba áthelyezni.

A véletlen újraindítások helyett, a környezet feltárására a **véletlen vándorlást** (*random walk*) használhatjuk. A véletlen vándorlás egyszerűen véletlen módon választ egy lehetséges cselekvést az aktuális állapotban. Elsőbbséget élvezhetnek az eddig még nem kipróbált cselekvések. Könnyű bebizonyítani, hogy a véletlen vándorlás *valamikor* megtalálja a célt, vagy feltárja a környezetet, feltéve, hogy a tér véges.<sup>15</sup> Másfelől a folyamat igen lassú lehet. A 4.21. ábra egy olyan környezetet mutat, ahol a véletlen vándorlásnak exponenciálisan sok lépésre van szüksége a cél megtalálásához, mert minden lépésnél a visszafelé haladás kétszer olyan valószínű, mint az előrehaladás. A példa persze kitalált, azonban sok olyan valós állappotter létezik, melynek topológiája ilyenfajta „csapdát” állít a véletlen vándorlás elő.

Sokkal hatékonyabb megközelítés, ha a hegymászást nem véletlen működéssel, hanem *memóriával* látjuk el. Az alapötlet az, hogy a cél minden meglátogatott állapotból való elérési költségének egy  $H(s)$  „aktuális legjobb becslését” eltároljuk.  $H(s)$  a  $h(s)$  heurisztikus becslésből indul, és ahogy az ágens egyre több tapasztalatot gyűjt az állappotterben, értéke folyamatosan frissül. A 4.22. ábra egy egyszerű példát mutat egy-dimenziós állappottrére. Az (a)-ban úgy tűnik, az ágens egy lapos lokális minimumban, az árnyalt állapotban bennragadt. Ahelyett hogy ott maradna, ahol volt, az ágensnek követnie

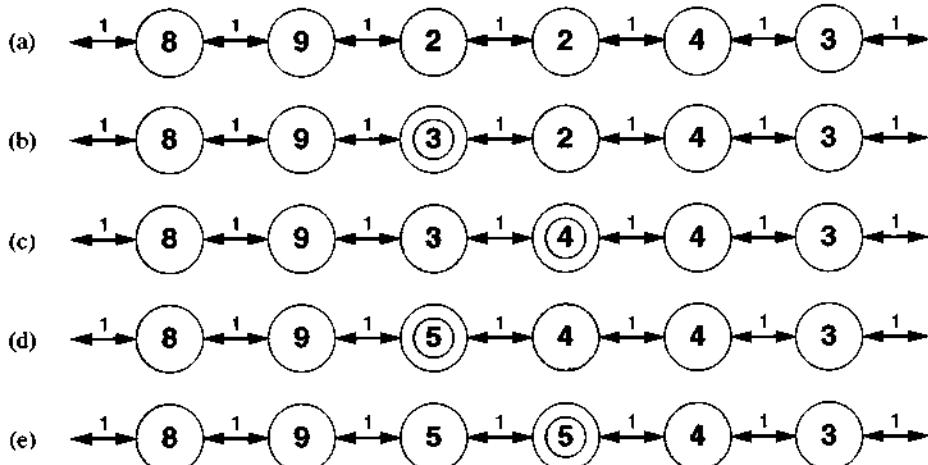
<sup>15</sup> A végtelen eset sokkal trükkösebb. A véletlen vándorlás teljes egy- és kétdimenziós végtelen rácsban, de nem a háromdimenziósban! Ebben az utóbbi esetben annak a valószínűsége, hogy a vándorlás valamikorra a kiindulópontra visszatér, csak 0,3405 körüli. Egy általános bevezető: (Hughes, 1995).



4.21. ábra. Egy környezet, amelyben a véletlen vándorlásnak exponenciálisan sok lépére lesz szüksége, hogy a célt megtalálja

kellene azt az utat, ami a szomszédos állapotok aktuális költségbecslései alapján a célhoz vezető legjobb útnak néz ki. Annak becsült költsége, hogy a célt az  $s'$  szomszédon keresztül érjük el az  $s'$  elérésének költsége, valamint a cél elérésének becsült költsége az  $s'$  állapotból, azaz  $c(s, a, s') + H(s')$ . A példában két cselekvés van, 1 + 9 és 1 + 2 becsült költségekkel, legcélszerűbbnek tűnik tehát jobbra menni. Most világos, hogy az árnyékolt állapot költségének becslésére a kettes érték túl optimista. Mivel a legjobb cselekvés költsége 1, és ez elvezet a céltól legalább kétlépésnyire lévő állapothoz, az árnyalt állapot legalább háromlépésnyire van a céltól. A  $H$  függvényt így ennek megfelelően frissíteni kell, ahogy a 4.22. (b) ábra mutatja. Ezt a folyamatot folytatva az ágens még kétszer mozdul el előre-hátra, a  $H$ -t frissítve és a lokális minimumot „laposítva”, amíg jobbra nem tud elmenekülni.

Ezt a sémát, aminek **tanuló valós idejű A\*** (**TRTA\***) a neve, a 4.23. ábrán látható ágens implementálja. Ez az ágens az **ONLINE-MÉLYÉGI-ÁGENS**-hez hasonlóan az *eredmény* táblázat felhasználásával környezetének egy térképet épít meg. Felfrissíti az éppen elhagyott állapot költségbecslését, és az aktuális költségbecslés szerint a „lát-szólag legjobb” cselekvést választja. Fontos részlet, hogy az  $s$  állapotban még ki nem próbált cselekvésekkel minden iterációban bekarakítuk.



4.22. ábra. A TRTA\* öt iterációja egydimenziós állapottérben. minden állapotot  $H(s)$ -sel, a cél elérésének aktuális becslésével, és minden eltér lépésköltséggel címkeztük. Az árnyalt állapot az ágens helyét jelöli meg, és a frissített értékeket minden iterációban bekarakítuk.

közvetlenül a célhoz vezetnek. Ez a **bizonytalanság melletti optimizmus (optimism under uncertainty)** felbátorítja az ágenst, hogy új, feltehetően gyümölcsöző utakat tárjon fel.

Egy TRTA\* ágens garantáltan megtalálja a célt akármilyen véges, biztonságosan felférható környezetben. Az A\*-gal ellentétben azonban nem teljes a végletes állapotokban – vannak olyan esetek, amikor az ágenst reménytelenül félelhet vezetni. Egy  $n$  állapotból álló környezetet legrosszabb esetben  $O(n^2)$  lépéssel tár fel, de sokszor jóval hatékonyabb ennél. A TRTA\* ágens a cselekvéskiválasztási és a frissítési szabályokkal különböző módon definiálható online ágensek nagy családjának egy partikuláris tagja. Ezzel a családdal, melyet eredetileg sztochasztikus környezetekhez találtak ki, a 21. fejezetben foglalkozunk.

## Tanulás online keresés során

Az online kereső ágens kezdeti tudatlansága több lehetőséget teremt a tanulásra. Először az ágens a környezetének „térképét” – pontosabban minden állapotban minden cselekvés kimenetelét – tanulja meg egyszerűen regisztrálva a tapasztalatát. (Jegyezzük meg, hogy a determinisztikus környezet feltételezése azt jelenti, hogy egy-egy cselekvés esetén egy tapasztalás elég lesz.) Másodszor a lokálisan kereső ágensek lokális frissítési szabályok segítségével pontosabb értékbecslésekre tesznek szert minden állapot esetén, ahogy ez a TRTA\*-nál történt. Látni fogjuk a 21. fejezetben, hogy ezek a frissítések előbb-utóbb minden állapot *egzakt* értékéhez fognak konvergálni, feltéve, hogy az ágens az állapotteret megfelelő módon tárja fel. Amikor az egzakt értékek már

```

function TRTA*-ÁGENS( $s'$ ) returns egy cselekvés
  input:  $s'$ , egy érzékelés, amely az aktuális állapotot azonosítja
  static: eredmény, cselekvésekkel és állapotokkal indexelt táblázat, kezdetben üres
     $H$ , költségbecslések táblázata állapotokkal indexelve
     $s, a$ , az előbbi állapot és cselekvés, kezdetben nulla

  if CÉL-TESZT( $s'$ ) then return stop
  if  $s'$  egy új állapot (nincs  $H$ -ban) then  $H[s'] \leftarrow h(s')$ 
  unless  $s$  nulla
    eredmény[ $a, s$ ]  $\leftarrow s'$ 
     $H[s] \leftarrow \min_{b \in \text{CSELEKVÉSEK}(s)} \text{TRTA}^*$ -KÖLTSÉG ( $s, b, \text{eredmény}[b, s], H$ )
     $a \leftarrow$  egy olyan  $b$  cselekvés a CSELEKVÉSEK( $s'$ )-ben, ami TRTA*-KÖLTSÉG( $s', b, \text{eredmény}[b, s'], H$ )-t
      minimalizálja
     $s \leftarrow s'$ 
  return  $a$ 

function TRTA*-KÖLTSÉG( $s, a, s', H$ ) returns egy költségbecslés
  if  $s'$  nem definiált then return  $h(s)$ 
  else return  $c(s, a, s') + H[s']$ 
```

**4.23. ábra.** A TRTA\* ágens a cselekvéseit a környező állapotok értékei alapján választja ki, mely az állapotokat az állapottérben mozogva folyamatosan frissít

ismertek, az optimális döntések egyszerűen a legmagasabb értékű utódállapotba való átmenetekkel hozhatók meg – ilyenkor tehát a tiszta hegymászás az optimális stratégia.

Ha elfogadta a javaslatunkat, hogy az ONLINE-MÉLYSÉGI-ÁGENS viselkedését kövesse nyomon a 4.18. ábrán bemutatott környezetben, észreveheti, hogy az ágens nem valami lángész. Így például, miután már láta, hogy a *Fel* cselekvés az (1,1)-ből az (1,2)-be visz, még mindig fogalma sincs, hogy a *Le* cselekvés az (1, 1)-be visz vissza, vagy hogy a *Fel* cselekvés a (2,1)-ből a (2,2)-be, a (2,2)-ből a (2,3)-ba stb. visz. Általánosságban azt szeretnénk, ha az ágensünk megtanulná, hogy a *Fel* növeli az y koordinátát, hacsak egy fal nincs útan, a *Le* a koordinátát csökkenti stb. Hogy ez megtörténhessen, két dologra van szükség. Először az ilyen általános szabályokhoz szükségünk van egy formális és manipulálható reprezentációra. Az információt egyelőre az állapotátmennet-függvény nevű fekete dobozba rejtettük. Ezzel a témaival a III. rész foglalkozik. Másodszor szükségünk van egy algoritmusra, amely képes a megfelelő szabályokat az ágens által megtett konkret megfigyelésekből konstruálni. Ezzel pedig a 18. fejezet foglalkozik.

## 4.6. ÖSSZEFoglalás

Ebben a fejezetben áttekintettük, hogyan lehet a keresési költséget heurisztikus függvények használatával csökkenteni. Megvizsgáltunk számos heurisztikus függvényt alkalmazó algoritmust, és láttuk, hogy az optimalitásért a keresési költség viszonylatában drága árat kell fizetnünk, még akkor is, ha sikerült jó heurisztikus függvényt találnunk.

- A **legjobbat-először keresés** (**best-first search**) egyszerűen egy olyan GRÁF-KERESÉS, ahol először (valamelyen mérték szerint) a legkisebb költségű, még ki nem fejtett csomópontokat fejtjük ki. A legjobbat-először algoritmusok tipikusan egy *h(n)* heurisztikus (*heuristics*) függvényt használnak, amely a cél költségét becsüli az *n* állapotból kiindulva.
- A **mohó legjobbat-először keresés** (**greedy best-first search**) a minimális *h(n)* értékű csomópontokat fejt ki. Nem optimális, azonban sokszor hatékony.
- Az **A\* keresési algoritmus** (**A\* search**) a minimális  $f(n) = g(n) + h(n)$  értékű csomópontokat fejt ki. Az A\* algoritmus teljes és optimális, feltéve, hogy garantálni tudjuk, hogy *h(n)* elfogadható (a FA-KERESÉS számára) vagy konziszens (a GRÁF-KERESÉS számára). Az A\* algoritmus tárkomplexitása még mindig elfogadhatatlan.
- A heurisztikus algoritmusok hatékonyさga a heurisztikus függvény minőségétől függ. Jó heurisztikus függvények készíthetők néha például a probléma definíciójának relaxálásával, a mintaadatbázis részproblémáihoz tartozó megoldási költségek előzetes kiszámításával vagy a problémaosztályon belül a tapasztalatból való tanulással.
- Az **RLEK (RBFS)** és az **EMA\* (SMA\*)** robusztus, optimális keresési algoritmusok, amelyek korlátozott mennyiségi memóriát használnak. Ha elegendő idő áll rendelkezésre, olyan problémákat is megoldanak, melyeket az A\* algoritmus nem képes megoldani, mert elfogy a memóriája.
- A **lokális keresési módszerek**, mint például a **hegymászó keresés** (**hill climbing**) teljes állapotleírásokkal dolgoznak, de a memóriában csak csekély számú csomópontot tartanak. Több sztochasztikus algoritmust is kifejlesztettek, beleértve a **szimulált lehűtést** (**simulated annealing**), amely megfelelő hűtési karakterisztika esetén

optimális megoldással tér vissza. A folytonos térbeli problémákra számos lokális keresési algoritmus is használható.

- A **genetikus algoritmus** (*genetic algorithm*) egy olyan sztochasztikus hegymászó keresés, ahol az állapotok nagy populációjával dolgozunk. Új állapotokat **mutációval** (*mutation*) és a populációbeli állapotpárokat összekombináló **keresztezéssel** (*cross-over*) generálunk.
- **Felfedezési problémákról** (*exploration problems*) akkor van szó, ha az ágensnek fogalma sincs környezete állapotairól és cselekvéseiről. Biztonságosan feltártató környezetek esetén az **online kereső** (*online search*) ágens felépítheti a környezetek térképét, és megtalálja a célt, ha az létezik. A heurisztikus becslések tapasztalat alapján történő frissítése hatékony módszer, az ágens lokális minimumokból való kimeneküléshez.

## Irodalmi és történeti megjegyzések

A heurisztikus információ alkalmazása a problémamegoldásban Simon és Newell egy korai írásában jelenik meg (Simon és Newell, 1958). A „heurisztikus keresés” frázis és a célhoz való távolságot becsült heurisztikus függvény használata azonban valamivel későbbőről származik (Newell és Ernst, 1965; Lin, 1965). Doran és Michie részletesen, kísérleti alapon tanulmányozták a heurisztikus keresési algoritmusok alkalmazását számos problémára, nagy hangsúlyt fektetve a 8-as és a 15-ös kirakójátékre (Doran és Michie, 1966). Habár Doran és Michie a heurisztikus keresésnél elméleti úthossz és „behatolás” (az úthossz és az eddig vizsgált csomópontok számának aránya) elemzéseket végzett, úgy tűnik, figyelmen kívül hagyták az aktuális úthossz nyújtotta információt. A heurisztikus keresésbe az aktuális úthosszt is beszámító A\* algoritmust Hart, Nilsson és Raphael dolgozták ki (Hart és társai, 1968), néhány későbbi korrekcióval (Hart és társai, 1972). Az A\* algoritmus optimális hatékonyságát Dechter és Pearl (Dechter és Pearl, 1985) mutatták ki.

Az eredeti A\* algoritmust ismertető cikk bevezette a heurisztikus függvények konziszenciájának feltételét. A heurisztikák monotonitási feltételét a konziszenciafeltételek egy egyszerűbb feltétellel való kiváltására Pohl vezette be (Pohl, 1977), de Pearl megmutatta, hogy a két feltétel ekvivalens (Pearl, 1984). Az A\*-ot megelőző számos algoritmus használta a nyitott és a zárt listával ekvivalens fogalmakat. Ezek közé tartozik a szélességi, a mélységi és az egyenletes költségű keresés (Bellman, 1957; Dijkstra, 1959). Különösképpen Bellman munkája mutatta meg az additív útköltség fontosságát az optimalizálási problémák egyszerűsítésében.

Pohl elsőként tanulmányozta az A\* algoritmus heurisztikahibája és időigénye közötti összefüggést (Pohl, 1970; 1977). Annak bizonyítása, hogy az A\* algoritmus lineáris időben fut, ha a heurisztikus függvény hibája egy állandó korlát alatt van, Pohl és Gaschnig cikkében található (Pohl, 1977; Gaschnig, 1979). Pearl ennek az eredménynek egy erősebb megfogalmazását adta meg, amely megengedte a hiba logaritmikus növekedését (Pearl, 1984). A heurisztikus keresés hatékonyságát jellemző „effektív elágazási tényező” mértéket Nilsson javasolta (Nilsson, 1971).

Az A\* algoritmusnak rengeteg változata létezik. Pohl javasolta a dinamikus súlyozási technikát, amelyben kiértékelő függvényként az A\*-ban használatos egyszerű  $f(n) =$

$g(n) + h(n)$  helyett az aktuális úthossz és a heurisztikus függvény alábbi súlyozott összege szerepel:  $f_w(n) = w_g g(n) + w_h h(n)$  (Pohl, 1973). Ez a módszer a keresés előrehaladával dinamikusan állítja a  $w_g$  és  $w_h$  súlyokat. Kimutatható, hogy a Pohl-féle algoritmus  $\varepsilon$ -elfogadható – azaz garantálja a megoldás megtalálását az optimális megoldáshoz képest  $1 + \varepsilon$  tényezőn belül. Itt  $\varepsilon$  az algoritmus egy bemeneti paramétere. Ugyanilyen tulajdonságú a  $A^*$  algoritmus (Pearl, 1984), amely a perem minden olyan csomópontját kiválasztja, amely a perembeli minimális  $f$ -költségű csomóponttól  $f$ -költségenben  $(1 + \varepsilon)$ -os tényezőn belül van. A kiválasztást úgy lehet lebonyolítani, hogy ez a keresési költségeket minimalizálja.

Az  $A^*$  keresési algoritmus és más állapottérben kereső algoritmusok nagyban hasonlítanak az operációkutatás területén elterjedten alkalmazott *elágazik- és korlátoz* (*branch-and-bound*) technikákhoz (Lawler és Wood, 1966). Az állapottérbeli keresés és az elágazik- és korlátoz algoritmus közötti kapcsolatot részletesen vizsgálták Dana Nau, Laveen Kanal és Vipin Kumar (Kumar és Kanal, 1983; Nau és társai, 1984; Kumar és társai, 1988). Martelli és Montanari meghamtatták a kapcsolatot a dinamikus programozás (lásd 17. fejezet) és bizonyos típusú állapottérbeli keresések között (Martelli és Montanari, 1978). Kumar és Kanal CDP – „összetett döntési folyamat” – néven megpróbálta „egységes alapokra hozni” a heurisztikus keresést, a dinamikus programozást és az elágazik- és korlátoz technikákat (Kumar és Kanal, 1988).

Mivel az 1950-es évek végén, az 1960-as évek elején a számítógépek még csak legfeljebb pár száz szónyi memóriával rendelkeztek, így a memóriakorlátozott keresés már a kezdeti időkben is intenzíven kutatott terület volt. Doran és Michie Graph Traverser programja (Doran és Michie, 1966) az egyik legkorábbi keresőprogram, a memóriakorlát által megengedett mértékben legjobbat-először keresést hajt végre, majd kiválaszt egy operátor. Az IMA\* algoritmus (Korf, 1985a, 1985b) volt az első széles körben alkalmazott optimális, memóriakorlátozott heurisztikus keresési algoritmus; ennek számos változatát ki is dolgozták. Az IMA\* algoritmus hatékonyságának részletes elemzése és az algoritmus valós értékű heurisztikákkal kapcsolatos nehézségeinek tárgyalása a (Patrick és társai, 1992)-ben jelent meg.

Az RLEK (Korf, 1991, 1993) valójában egy kicsit bonyolultabb, mint a 4.5. ábrán bemutatott algoritmus, amely közelebb áll a flüggetlenül kifejlesztett ún. **iteratívan kifejtő** (iterative expansion) vagy **IK** (IE) (Russell, 1992) algoritmushoz. Az RLEK felső és alsó korlátot használ. A két algoritmus viselkedése azonos elfogadható heurisztikák esetén. Az RLEK azonban a csomópontokat a legjobbat-először sorrendben akkor is kifejti, ha a heurisztika nem elfogadható. A legjobb alternatív út számítartásának gondolata korábban jelent meg Bratkónál (Bratko, 1986) az  $A^*$  elegáns Prolog implementációjában és a DTA\* algoritmusban (Russell és Wefald, 1991). Ez utóbbi a metaállapoterekkel és a metaszintű tanulással is foglalkozik.

Az MA\* algoritmus először Chakrabartinál jelent meg (Chakrabarti és társai, 1989). Az EMA\* vagy Egyszerűsített MA\* az MA\* implementációs kísérleteiből született meg, mint az IK egy összehasonlító algoritmusa (Russell, 1992). Kaindl és Khorsand az EMA\* felhasználásával megalkottak egy kétirányú keresési algoritmust, amely jelentősen gyorsabb, mint a korábbi algoritmusok (Kaindl és Khorsand, 1994). Korf és Zhang az oszd-meg- és-uralkodj megközelítést írják le (Korf és Zhang, 2000). Zhou és Hansen pedig bevezették a memóriakorlátozott  $A^*$  gráfkeresést (Zhou és Hansen, 2002). Korf áttekintést ad a memóriakorlátozott keresési technikákról (Korf, 1995).

Held és Karp (Held és Karp, 1970) nagy hatású cikke a minimális-feszítőfa alkalmazását (lásd 4.8. feladat) taglalja az utazó ügynök problémára, megmutatva, hogy a relaxált probléma vizsgálatával hogyan lehet elfogadható heurisztikákra jutni.

Prieditis a Jack Mostow-val végzett korábbi kutatásokra építve (Mostow és Prieditis, 1989) sikeresen automatizálta a probléma relaxálási folyamatát (Prieditis, 1993). A mintaadatbázisok használatát az elfogadható heurisztikák előállítására Gasser, valamint Culberson és Schaeffer kezdeményezték (Gasser, 1995; Culberson és Schaeffer, 1998). A diszjunkt mintaadatbázisokat Korf és Felner írták le (Korf és Felner, 2002). A heurisztikák valószínűség-alapú értelmezését Pearl és Hansson és Mayer mélyrehatóan vizsgálták (Pearl, 1984; Hansson és Mayer, 1989).

A heurisztikus keresési algoritmusok messze legátfogóbb irodalma Pearl *Heuristics* c. könyve (Pearl, 1984). Ez a könyv különösen jó áttekintést nyújt számos oldalhajtásról és az A\* különféle változatairól, és azok tulajdonságainak szigorú bizonyításait is tartalmazza. Kanal és Kumar elkészítették az alapvető és lényeges heurisztikus keresés-sel foglalkozó cikkek antolói-ját (Kanal és Kumar, 1988). A keresési algoritmusokkal kapcsolatos új eredmények rendszeresen az *Artificial Intelligence* folyóiratban jelennek meg.

A lokális keresési technikáknak a matematikában és a számítógépes tudományokban hosszú történetük van. Valóban, a Newton–Raphson-módszert (Newton, 1671; Raphson, 1690) egy igen hatékony lokális keresési módszernek lehet tekinteni folytonos terekben, ahol a gradiens információ rendelkezésre áll. Az ilyen információt nem igénylő optimális algoritmusok klasszikus forrása Brent munkája (Brent, 1973). A nyalábkeresés, melyet lokális keresési algoritmusként mutattunk be, a beszédfelismerési célokra alkalmazott dinamikus programozás korlátozott szélességű változataiként indult a HARPY rendszerben (Lowerre, 1976). A megfelelő algoritmust részletesen Pearl vizsgálta (Pearl, 1984, 5. fejezet).

A lokális keresés témaköre a nagy kényszerkielégítési problémák, mint például az  $n$ -királynő (Minton és társai, 1992) és a logikai következetés (Selman és társai, 1992) területén elérte meglepően jó eredményeknek és a véletlenség, a többszörös egyidejű keresés és más javítás beépítésének köszönhetően az utóbbi években megélenkült. Az ilyen algoritmusok reneszánsza, melyeket Christos Papadimitriou „New Age” algoritmusoknak nevez, felkelte az elméleti számítógép-tudományokkal foglalkozó kutatók érdeklődését is (Koutsoupias és Papadimitriou, 1992; Aldous és Vazirani, 1994). Az operációkutatás területén népszerűségeknek örvend a hegymászó keresés egy változata, amit tabukeresési algoritmusnak (tabu search) hívnak (Glover, 1989; Glover és Laguna, 1997). Az emberi rövid távú memóriamodellre alapozva ez az algoritmus karbantartja az előbb meglátogatott  $k$  állapot tabulistáját, melyeket újra meglátogatni nem szabad. Ezáltal az algoritmus hatékonysága megnőtt a gráfkeresésben, és képes egyes lokális minimumokból kimenekülni. A hegymászó keresés másik hasznos javítása a STAGE algoritmus (Boyan és Moore, 1998). Az ötlet az, hogy használjuk a véletlen újraindítású hegymászó keresés révén nyert lokális maximumokat a tájfelszín általános alakjának felderítésére. Az algoritmus a lokális maximumokra sima felületet illeszt, és a globális maximumot analitikusan számítja ki. Ez lesz az új újraindítási pont. Az algoritmus a gyakorlatban működőképesnek bizonyult nehéz problémák esetén. Gomes (Gomes és társai, 1998) azt mutatta meg, hogy a futási idő eloszlása a szisztematikusan visszalépő algoritmusok esetén sokszor lassan lecsengő eloszlás (heavy-tail distribution), ami azt jelenti, hogy

a nagyon hosszú futási idők valószínűsége nagyobb, mint amit a normális eloszlás alapján jósolni lehetne. Ez megadja a véletlen újraindítási mechanizmus elméleti igazolását.

A szimulált lehűtést először Kirkpatrick és szerzőtársai írták le (Kirkpatrick és társai, 1983), akik az algoritmust közvetlenül a statisztikus fizika területén összetett rendszerek szimulálására használt **Metropolis-algoritmustól** kölcsönözték (Metropolis és társai, 1953). A Metropolis-algoritmust állítólag Los Alamosban egy estély során találták ki. A szimulált lehűtés mára már különálló kutatási részterületet alkot, ahol évente több száz cikket publikálnak.

Optimális megoldások keresése folytonos térben több terület alapfeladata, az **optimizálás elméletet**, az **optimális szabályozás elméletet** és a **variációkalkulust (optimization theory, optimal control theory, calculus of variations)** beleértve. Ezekről alkalmas (és gyakorlati) áttekintést Press (Press és társai, 2002) és Bishop (Bishop, 1995) adnak. A **lineáris programozás (LP, linear programming)** a számítógépek egyik első alkalmazása volt. A **szimplex algoritmust (simplex-algorithm)** (Wood és Dantzig, 1949; Dantzig, 1949) még mindig alkalmazzák annak ellenére, hogy a legrosszabb esetben exponenciális komplexitású. Az LP egy gyakorlati polinomiális idejű algoritmusát Karmarkar (Karmarkar, 1984) dolgozta ki.

A genetikus algoritmusok fejlődésének fontos előfutára volt Sewall Wright munkája a **fitness tájfelszín (fitness landscape)** fogalmáról. Néhány statisztikus, Boxot (Box, 1957) és Friedman (Friedman, 1959) is beleértve, az '50-es években használt már evolúciós technikákat optimalizálási problémára, e megközelítés azonban csak akkor kezdett népszerűvé válni, amikor Rechenberg (Rechenberg, 1965; 1973) az **evolúciós stratégiákat (evolution strategies)** a szárnyprofilok optimalizálási problémáinak megoldásába vezette. Az 1960-as és 1970-es években John Holland (Holland, 1975) kiált a genetikus algoritmusok mellett, azt tartva, hogy a genetikus algoritmusok hasznos eszköznek bizonyulnak és segítenek abban, hogy jobban megértsük a biológiai és egyéb adaptációt (Holland, 1995). A **mesterséges élet (artificial life)** mozugalom (Langton, 1995) ezt a gondolatot egy lépéssel tovább viszi, a genetikus algoritmusokkal létrehozott eredményeket nem problémamegoldások, hanem inkább **szervezeteknek** tekintve. Hinton és Nowlan (Hinton és Nowlan, 1987), valamint Ackley és Littman (Ackley és Littman, 1991) munkája ezen a területen sokban hozzá járult, hogy a Baldwin-hatás következményeit feltártuk. Smith és Szathmáry (Smith és Szathmáry, 1990) munkáját mint az evolúció általános hátterét bermutató munkát ajánljuk.

A legtöbb összehasonlítás a genetikus algoritmusok és más megközelítések (különösképpen a hegymászó keresés) között azt találta, hogy a genetikus algoritmusok lassabban konvergálnak (O'Reilly és Oppacher, 1994; Mitchell és társai, 1996; Juels és Wattenberg, 1996; Baluja, 1997). Az ilyen eredmények a GA közösségen nem nagyon népszerűek, azonban e közösségen belül a legutóbbi olyan kísérletek, hogy a populációalapú keresést a Bayes-tanulás egy formájaként értelmezzük (lásd 20. fejezet) talán segít, hogy e terület és a kritikusai közötti szakadék eltűnjön (Pelikan és társai, 1999). A GA hatékonyságára magyarázatot adhat a **kvadratikus dinamikus rendszerek (quadratic dynamical systems)** elmélete (Rabani és társai, 1998) is, lásd például Lohn és társainak (Lohn és társai, 2001) a GA antennatervezésre alkalmazott példáját és Larrañaga (Larrañaga és társai, 1999) munkáját, ahol a GA-t az utazó ügynök problémára alkalmazták.

A **genetikus programozás (genetic programming)** a genetikus algoritmusokkal közelei rokonságban van. Az elsődleges különbség az, hogy a mutált és az összekombinált

reprezentációk nem bitfüzérek, hanem programok. A programokat kifejezésfák alakjában reprezentáljuk. A kifejezéseket valamilyen standard nyelvből meríthetjük, mint például a Lispból, de lehetnek egy olyan nyelv elemei, melyet kifejezetten áramkörök, robotsabályozók stb. reprezentálására tervezünk. A kereszteszés a részfák és nem a részfüzérek összeillesztését jelenti. A mutáció ezen formája garantálja, hogy az utódok jól definiált kifejezések lesznek, ami nem biztos, hogy sikerülne, ha a programok manipulációját füzérszinten oldanánk meg.

A genetikus programozást övező jelenlegi érdeklődést John Koza (Koza, 1992) munkája váltotta ki, a módszer azonban visszavezethető Friedbergnek (Friedberg, 1958) a gépi kóddal folytatott korai kísérleteihez és Fogelnek (Fogel és társai, 1996) a véges állapotú automatákkal folytatott vizsgálataihoz. Mint a genetikus algoritmusok esetén itt is vita folyik a módszer hatékonyságát illetően. Koza (Koza és társai, 1999) sok olyan kísérletet ír le, amelyeket genetikus programozás felhasználásával az automatizált áramkörtervezés területén végzett.

A genetikus algoritmusokról és a genetikus programozásról az *Evolutionary Computation* és az *IEEE Transactions on Evolutionary Computation* folyóiratok írnak. Idevágó cikkeket még a *Complex Systems*, az *Adaptive Behavior* és az *Artificial Life* folyóiratokban is találunk. A legfontosabb konferenciák az *International Conference on Genetic Algorithms* és a *Conference on Genetic Programming*, amelyek a közelműltben *Genetic and Evolutionary Computational Conference* néven fuzionáltak. A terület jó áttekintését adják Melanie Mitchell és David Fogel írásai (Mitchell, 1996; Fogel, 2000).

Az ismeretlen állapottereket feltáró algoritmusokkal évszázadok óta foglalkoztak. Egy labirintusban való mélységi keresés implementálható például úgy, hogy bal kezünket folyamatosan a falon tartjuk. A hurkokat elkerülhetjük, ha a kereszteszéket megjöljük. A mélységi keresés kudarcba fullad, ha a cselekvések nem visszafordíthatók. Az **Euler-gráfok** (*Eulerian graphs*) (azaz olyan gráfok, ahol minden csomópontban ugyanannyi él megy be, mint ki) feltárásnak általánosabb problémáját Hierholzer (Hierholzer, 1873) algoritmusával oldotta meg. A tetszőleges gráfok feltárásnak első részletes algoritmikus vizsgálatát Deng és Papadimitriou (Deng és Papadimitriou, 1990) végezték, akik egy teljesen általános algoritmushoz jutattak el, azonban kimutatták, hogy az általános gráf feltárására nem adható korlátos kompetitív arány. Papadimitriou és Yannakakis (Papadimitriou és Yannakakis, 1991) azt a kérdést vizsgálták, hogy hogyan található meg a célhoz vezető út geometriai úttervező környezetben (ahol minden cselekvés visszafordítható). Kimutatták, hogy négyzet alakú akadályok esetén kis kompetitív arány érhető el, az általános téglalap alakú akadályok esetén azonban korlátos arány nem érhető el (lásd 4.19. ábra).

A TRTA\* algoritmust, a valós idejű keresés (real-time search) vizsgálatának részeként olyan környezetekben, ahol az ágensnek cselekednie kell egy rögzített idejű keresést követően (a kétjátékos játékokban egy megszokottabb helyzet) Korf (Korf, 1990) fejlesztette ki. A TRTA\* valójában a megerősítéses tanulási algoritmus egy speciális esete sztochasztikus környezetekben (Barto és társai, 1995). Az optimizmus bizonytalanság mellett stratégiája – mindenkor legközelebbi még nem meglátogatott állapotok felé tartani – olyan feltárási mintát eredményezhet, amely a nem informált esetben kevésbé hatékony, mint az egyszerű mélységi keresés (Koenig, 2000). Dasgupta (Dasgupta és társai, 1994) kimutatta, hogy egy kiegyszúlyozott fában a cél megkeresésére, amikor heurisztikus információ nincs, az online iteratívan mélyülő keresés optimálisan hatékony.

A TRTA\* számos informált változatát is kifejlesztették, különböző módszereket alkalmazva, a gráf már ismert részein belüli keresésre és frissítésre (Pemberton és Korf, 1992). Egyelőre még nem sikertűlt megérteni, hogy heurisztikus információ alkalmazása esetén hogyan kell a célokot optimális hatékonysággal megkeresni.

A fejezet azért nem foglalkozott a **párhuzamos keresési (parallel search)** algoritmusokkal, mert ezek tárgyalásához a párhuzamos architektúrák terjedelmes ismertetésére lett volna szükség. A párhuzamos keresés egyre fontosabb témakörré válik minden az MI, minden pedig az elméleti számítógép-tudomány területén. Egy az MI idevágó irodalmába történő rövid bevezető Mahanti és Daniels cikkében (Mahanti és Daniels, 1993) található.

## Feladatok

- 4.1. Kövesse végig az A\* működését a Lugasból Bukarestbe utazás problémájára alkalmazva, légvonalban mért távolság heurisztikát alkalmazva. Azaz adja meg az algoritmus által megvizsgált csomópontok szekvenciáját az  $f$ -,  $g$ - és  $h$ -értékükkel együtt.
- 4.2. A **heurisztikus útalgoritmus (heuristic path algorithm)** egy olyan legjobbat-először keresés, ahol a célfüggvény  $f(n) = (2 - w)g(n) + wh(n)$ . Milyen  $w$  értékekre garantálható az algoritmus optimalitása? (Feltételezheti, hogy  $h$  elfogadható.) Milyen keresésről van szó, amikor  $w = 0$ ? Amikor  $w = 1$ ? És amikor  $w = 2$ ?
- 4.3. Bizonyítsa be az alábbi állításokat:
  - (a) A szélességi keresés az egyenletes költségű keresés egy speciális esete.
  - (b) A szélességi keresés, a mélységi keresés és az egyenletes költségű keresés a legjobbat-először keresés speciális esetei.
  - (c) Az egyenletes költségű keresés az A\* keresés egy speciális esete.
- 4.4. Tervezzen meg egy olyan állapotteret, ahol az A\* a GRÁF-KERESÉS felhasználásával egy szuboptimális megoldással tér vissza egy elfogadható, de nem konzisztens  $h(n)$  függvény mellett.
- 4.5. A 138–139. oldalon láttuk, hogy a légvonalban mért távolságheurisztika a mohó legjobbat-először keresést félrevezeti, amikor Iași-ról Fogarasra szerettünk volna eljutni. Az ellenkező problémára azonban, vagyis amikor Fogarastól Iașira szeretnénk eljutni, a heurisztika tökéletes. Léteznek olyan problémák, amelyek esetében a heurisztika minden irányban félrevezet bennünket?
- 4.6. Találjon ki a 8-as kirakójátéakra olyan heurisztikus függvényt, ami néha túlbecsül, és mutassa meg, hogy egy konkrét probléma esetén ez hogyan vezethet szuboptimális megoldáshoz (használhat számítógépet, ha ez segít). Bizonyítsa be, hogy ha  $h$  soha nem becsül túl  $c$ -nél jobban, az A\* algoritmus,  $h$ -t felhasználva, olyan megoldással tér vissza, melynek költsége az optimális megoldás költségét legfeljebb  $c$ -vel haladja meg.

- 4.7. Bizonyítsa be, hogy ha a heurisztikus függvény konzisztens, akkor elfogadható. Szerkesszen egy elfogadható heurisztikus függvényt, amely nem konzisztens.
- 4.8. Az utazó ügynök problémát (TSP) meg lehet oldani a minimális feszítő fa (MFF) heurisztika alkalmazásával, ami feltételezve, hogy már egy részútvonalat megtérvezünk, megbecsüli az útvonal befejezésének költségét. A városok egy halma-zának MFF-költsége az összes várost összekötő fák minimális élköltségösszege.
- (a) Mutassa meg, hogy ez a heurisztikus függvény hogyan származtattható a TSP relaxált változatából!
  - (b) Mutassa meg, hogy az MFF-heurisztika dominálja a légvonalban mért távolságheurisztikát!
  - (c) Írjon egy problémagenerátort a TSP-problémára, amely olyan probléma-egyedeket generál, amelyben a városokat az egységnégyzeten belüli véletlen módon generált pontok reprezentálják!
  - (d) Keressen hatékony módszert az irodalomban az MFF megszerkesztésére, és használja azt egy elfogadható keresési algoritmussal a TSP-probléma egyedeinek a megoldására!
- 4.9. A 150. oldalon a 8-as kirakójáték olyan relaxációját definiáltuk, ahol a lapka az A mezőről a B mezőre mozdulhat el, ha B üres. E probléma egzakt megoldása az ún. **Gaschnig-heurisztikát** (**Gaschnig's heuristics**) definiálja (Gaschnig, 1979). Magyarázza meg, hogy miért lesz a Gaschnig-heurisztika legalább olyan pontos, mint  $h_1$  (a rossz helyen lévő lapkák száma), és mutasson olyan eseteket, amikor mind  $h_1$ -nél mind  $h_2$ -nél pontosabb (Manhattan-távolság). Tud ajánlani a Gaschnig-heurisztikára egy hatékony számítási módszert?
- 4.10. A 8-as kirakójátékra adtunk két egyszerű heurisztikus függvényt: a Manhattan-távolságot és a rossz helyen lévő lapkák számát. Az irodalomban számos heurisztikáról azt állítják, hogy azok minden előbbinél jobbak. Lásd például Nilsson (Nilsson, 1971), Mostow (Mostow és Prieditis, 1989) és Hansson (Hansson és társai, 1992) cikkét. Ellenőrizze ezeket az állításokat a heurisztikák implementálásával és a futási eredmények összehasonlításával.
- 4.11. Nevezze meg azokat az algoritmusokat, amelyek az alábbi speciális esetekből származnak:
- (a) Lokális nyaláb keresés  $k = 1$  mellett.
  - (b) Lokális nyaláb keresés egy kezdeti állapottal és korlátlan számú visszatartott állapottal.
  - (c) Szimulált lehűtés, ha  $T = 0$  minden időpillanatra (és kihagyva a leállási tesztet).
  - (d) Genetikus algoritmus  $N = 1$  nagyságú populációval.
- 4.12. Néha egy adott problémához nem létezik jó kiértékelő függvény, azonban létezik jó összehasonlítási módszer: vagyis el lehet döntenи, hogy egy csomópont egy másik csomópontról jobb-e anélkül, hogy valamilyen számértéket rendel-

nénk hozzájuk. Mutassa meg, hogy ez elégséges a legjobbat-először (best-first) keresés végrehajtásához. Létezik-e ilyen analógia az A\* esetére is?

**4.13.** Hozza kapcsolatba a TRTA\* időkomplexitását az algoritmus tárkomplexitásával.

**4.14.** Tegyük fel, hogy egy ágens a 4.18. ábrán látható  $3 \times 3$  labirintus környezetben tartózkodik. Az ágens tudja, hogy a kezdeti pozíciója az (1,1), a cél a (3,3)-n van, és hogy a négy *Fel*, *Le*, *Balra*, *Jobbra* cselekvésnek a szokásos hatása van, ha csak egy fal nincs útból. Az ágens *nem* tudja a belső falak helyét. minden egyes állapotban az ágens a legális cselekvések halmazt képes érzékelni. Azt is meg tudja mondani, hogy az állapotot meglátogatta-e már korábban, vagy pedig egy új állapotban van.

- (a) Magyarázza meg, hogy ezt az online keresési problémát hogyan lehet offline keresésnek tekinteni egy olyan hiedelmi állapotérben, ahol a kezdeti hiedelmi állapot a környezet minden lehetséges konfigurációját tartalmazza. Milyen nagy a kezdeti hiedelmi állapot? Milyen nagy a hiedelmi állapottér?
- (b) Hányfélé különböző megfigyelés lehetséges a kezdeti állapotban?
- (c) Írja le e probléma esetén az eshetőségi terv néhány első ágát. Milyen nagy (nagyjából) az egész terv?

Jegyezze meg, hogy ez az eshetőségi terv az adott leírást kielégítő *minden lehetséges környezet* számára megoldás. A keresés és a végrehajtás átlapolódása tehát egy ismeretlen környezetben nem szigorúan szükséges.

**4.15.** Ebben a feladatban a 4.8. feladatban leírt típusú TSP-probléma megoldására lokális keresési módszerekkel kísérletezzünk.

- (a) Tervezze meg a TSP megoldására a hegymászó keresést. Az eredményeket az MFF-heurisztikával (4.8. feladat) dolgozó A\* algoritmus optimális megoldásaival hasonlítsa össze.
- (b) Tervezzen meg a TSP megoldására egy genetikus algoritmust. Az eredményeket más megközelítésekkel hasonlítsa össze. Esetleg szüksége lehet a (Larrañaga és társai, 1999) cikk egyes reprezentációs javaslataira.

**4.16.** Generáljon sok 8-as kirakójáték- és 8-királynő feladat-példányt, és oldja meg ezeket (ha lehetséges) a hegymászó kereséssel (legmeredekebb emelkedő és az első-választás változatok), véletlen újraindítású hegymászó kereséssel és szimulált lehűtéssel. Mérje meg a megoldási költségeket, és százalékosan adja meg a megoldások arányát. Mindezeket ábrázolja egy grafikonon az optimális költség függvényében. Fúzzon kommentárt az eredményekhez.

**4.17.** Ebben a feladatban a hegymászó keresést a robotnavigáció kontextusában fogjuk vizsgálni, példának használva a 3.22. ábrán látható környezetet.

- (a) Ismételten oldja meg a 3.16. feladatot a hegymászó keresés segítségével. Le fog-e ragadni az ágens valamikor egy lokális maximumban? *Lehetséges*, hogy leragadjon az ágens konvex akadályok esetén?
- (b) Hozzon létre olyan konkáv környezetet, amiben az ágens leragad!

- (c) Módosítsa úgy a hegymászó algoritmust, hogy az az 1 mélységű keresés helyett  $k$  mélységű keresést használjon annak eldöntésére, hogy a következő lépéshoz hová menjen. Meg kell keresnie a legjobb  $k$  lépéses utat, e mentén egy lépést megtenni, majd az eljárást megismételni.
- (d) Létezik  $k$ -nak olyan értéke, ami mellett az új algoritmus garantáltan elkerüli a lokális maximumokat?
- (e) Magyarázza meg, hogy ebben az esetben a TRTA\* algoritmus hogyan teszi lehetővé az ágens számára, hogy a lokális minimumokból kimeneküljön.

- 4.18.** Hasonlítsa össze az A\* és a RLEK keresés hatékonyságát véletlen módon generált problémahalmazokon a 8-as kirakójáték (Manhattan-távolsággal) és a TSP-(MFH-vel, lásd 4.8. feladat) problémakörben. Értelmezze az eredményeket. Mi történik az RLEK hatékonyságával, ha a 8-as kirakójáték problémakörben a heurisztika értékeihez egy kis véletlen számot adunk hozzá?

# 5. KÉNYSZERKIELÉGÍTÉSI PROBLÉMÁK

Ebben a fejezetben megmutatjuk azt, hogy ha az állapotokat nem egyszerűen kis fekete dobozoknak tekintjük, akkor új hatékony keresési módszerek egész sorához jutunk, valamint jobban megértjük a probléma struktúráját és komplexitását.

A 3. és a 4. fejezetben azt az elkövetést jártuk körbe, hogy a problémákat **állapottérbeli** kereséssel lehet megoldani. A tárgyterületre jellemző heurisztikák segítségével értékelhetjük ezeket az állapotokat és ellenőrizhetjük, hogy nem járunk-e célállapotban. A keresési algoritmus szempontjából azonban mindegyik állapot egy megkülböztetés nélküli belső struktúrájú **fekete doboz** (black box). Az állapotokat egy tetszőleges adatstruktúra reprezentálja, amelyhez a problémára *jellemző* rutinokkal lehet hozzáérni: az állapotátmenet- és a heurisztikafüggvényel, valamint a célállapotteszettel.

A fejezetben a **kényszerkielégítési problémákkal** (*constraint satisfaction problems*) foglalkozunk, amikor az állapotok és a céltesztszámításnak egy szabályos, strukturált és elég egyszerű **reprezentációhoz** (lásd 5.1. alfejezet). Az állapotstruktúra segítségével keresési algoritmusokat lehet definálni, és nem *problémáspecifikus*, hanem *általános célú* heurisztikák használatával nagy problémák megoldása is lehetővé válik (lásd 5.2. és 5.3. alfejezet). Talán a legfontosabb azonban az, hogy a céltesztszámításnak a problémának a struktúráját (lásd 5.4. alfejezet). Ez pedig módszereket ad a kezünkbe a probléma dekompozíciójára, és lehetővé teszi, hogy megértsük a probléma struktúrájára és a megoldás nehézsége közötti szoros kapcsolatot.

## 5.1. KÉNYSZERKIELÉGÍTÉSI PROBLÉMÁK

A kényszerkielégítési problémák (angol rövidítéssel CSP) formális definícióját változók (**variables**),  $X_1, X_2, \dots, X_n$  és kényszerek (**constraints**),  $C_1, C_2, \dots, C_m$  halmazaival adhatjuk meg. minden egyes  $X_i$  változó esetén adott a lehetséges **értékek** egy nem-türes  $D_i$  tartománya. minden egyes  $C_i$  kényszer a változók valamely részhalmazára vonatkozik, és meghatározza a részhalmaz megengedett értékkombinációt. Egy problémaállapotot az definiál, hogy vagy néhány, vagy mindegyik változóhoz értékeket **rendelünk hozzá**  $\{X_i = v_i, X_j = v_j, \dots\}$ . Egy hozzárendelést **konzisztsnek** (vagy megengedettnek) nevezünk, ha egyetlen kényszert sem sért meg. Teljes az a hozzárendelés, amelyben minden változó szerepel, és egy teljes hozzárendelés a kényszerkielégítési problémának **megoldása**, ha minden kényszert kielégíti. Néhány kényszerkielégítési probléma azt is igényli, hogy a megoldás egy **célfüggvényt** maximalizáljon.

Mit is jelent minden? Tegyük fel, hogy Romániába beleunva Ausztrália térképét tanulmányozzuk, és a térképen az 5.1. (a) ábrán feltüntetett államokat és területeket

látjuk, feladatunk pedig az, hogy vörössel, zölddel és kékkel minden egyes részt színezünk ki úgy, hogy a szomszédos részeknek ne legyen azonos a színe. Ahhoz, hogy ezt kényszerkielégítési problémaként fogalmazhassuk meg, változókat kell bevezetnünk az egyes részekhez:  $NyA$ ,  $\bar{E}T$ ,  $Q$ ,  $\bar{U}DW$ ,  $V$ ,  $DA$  és  $T$ . Mindegyik változó tartománya a {vörös, zöld, kék} halmaz. Kényszerek írják elő, hogy a szomszédos részek különböző színűek legyenek, például  $NyA$  és  $\bar{E}T$  esetén a megengedhető kombinációk az alábbi párok lehetnek:

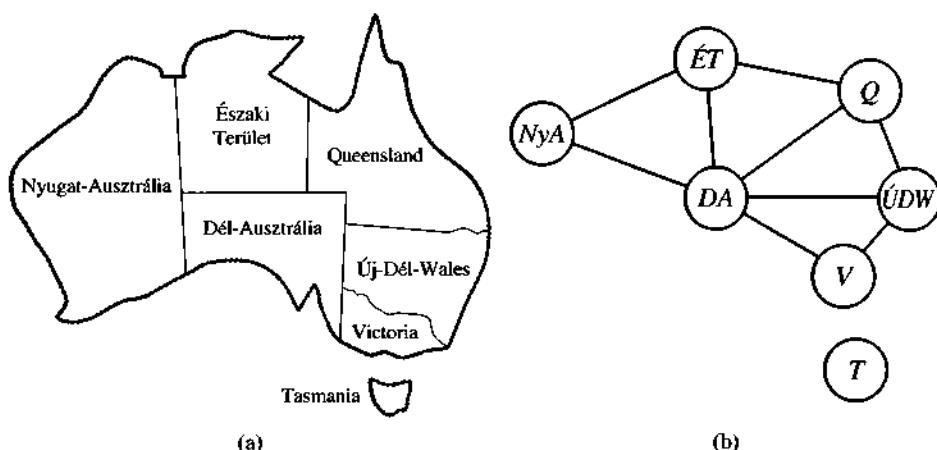
$$\{(vörös, zöld), (vörös, kék), (zöld, vörös), (zöld, kék), (kék, vörös), (kék, zöld)\}$$

(A kényszert tömörebben is ki lehet fejezni a  $NyA = \bar{E}T$  egyenlőtlenséggel, feltéve persze, hogy a kényszermegoldó algoritmus valamiképpen ki tudja értékelni az ilyen kifejezéseket.) Sok lehetséges megoldás adódik, mint például az

$$\{NyA = vörös, \bar{E}T = zöld, Q = vörös, \bar{U}DW = zöld, V = vörös, DA = kék, T = vörös\}$$

Gyakran hasznos lehet, ha felrajzoljuk a kényszerkielégítési probléma **kényszergráfját** (**constraint graph**), amint az az 5.1. (b) ábrán is látható. A gráf csomópontjai a probléma változóinak, élei pedig a kényszereknek felelnek meg.

Sok előnyvel járhat, ha egy problémát kényszerkielégítési problémaként tekintünk. Az állapotok reprezentációja miatt a kényszerkielégítési problémák egy standard min-tára illeszkednek (ez a minta a hozzárendelt értékekkel rendelkező változók halmaza), az állapotámenet-függvényt és a célállapottsetet pedig az összes kényszerkielégítési problémára érvényes általános módon meg lehet írni. Sőt létrehozhatók hatékony, általános heurisztikák minden kiegészítő, tárgyterület-specifikus szakértelem nélkül. Végül pedig a kényszergráf struktúrájának segítségével lerövidíthető a megoldási folyamat, és ez néhány esetben exponenciálisan csökkeneti a probléma komplexitását. A kényszerkielégítési probléma reprezentációja a legelső (és a legegyszerűbb) a könyv menete során bemutatott reprezentációs sémák sorában.



**5.1. ábra.** (a) Ausztrália államai és területei. A térkép kiszínezése tekinthető kényszerkielégítési problémának is. A cél az, hogy minden egyes részhez olyan színt találunk, amely nem azonos egy szomszédos rész színével sem. (b) A térképszínezési probléma kényszergráfjának reprezentálása.

Könnyű észrevenni, hogy egy kényszerkielégítési problémára adható egy **inkrementális megfogalmazás (incremental formulation)**, amely a kényszerkielégítési problémát szabályos keresési problémaként tekinti:

- **Kiinduló állapot (initial state):** az üres hozzárendelés {}, ahol egyik változónak sincs értéke.
- **Állapotátmenet-függvény (successor function):** bármelyik hozzárendelés nélküli változó értékét kaphat, amennyiben ez nem ütközik a korábbi értékadásokkal.
- **Célteszt (goal test):** az aktuális hozzárendelés teljes.
- **Az út költsége (path cost):** egy konstans költség (például 1) minden egyik lépésre.

Mindegyik megoldásnak egy teljes hozzárendelésnek kell lennie, tehát  $n$  változó esetén az  $n$ -edik mélységi szinten jelenik meg. A keresési fa pedig csak  $n$  mélységű. Emiatt a korlátkielégítési problémák esetén a mélységi keresési algoritmusok népszerűek (lásd 5.2. alfejezet). Az is igaz, hogy a megoldáshoz vezető út nem lényeges. Ezért használható a **teljes állapotleírás (complete-state formulation)** is, amelyben minden egyes állapot egy teljes változó-hozzárendelés, akár kielégíti a kényszereket, akár nem. Ebben a felírásban a lokális keresési eljárások is használhatók.

A kényszerkielégítési problémák legegyszerűbb esetében a változók **diszkrétek** és **véges tartományúak**. A térképszínezési problémák is ehhez az esethez tartoznak. A 3. fejezetben bemutatott 8-királynő problémát is felfoghatjuk véges tartományú kényszerkielégítési problémaként, ahol a  $Q_1, \dots, Q_8$  változók a királynők 1, ..., 8 oszlopokban betöltött pozícióit jelölik, és minden egyik változó tartománya {1, 2, 3, 4, 5, 6, 7, 8}. Amennyiben maximum  $d$  a kényszerkielégítési probléma bármely változójához tartozó tartomány mérete, akkor a lehetséges teljes hozzárendelések száma  $O(d^n)$ , azaz a változók számának exponenciális függvénye. A véges tartományú kényszerkielégítési problémák közé tartoznak a Boole-problémák is, ahol a változók értéke vagy *igaz*, vagy *hamis*. A **Boole kényszerkielégítési problémák** speciális esetként tartalmaznak néhány NP-teljes problémát, például a 3SAT-ot (lásd 7. fejezet). Legrosszabb esetben tehát nem lehetséges, hogy exponenciálisan kevesebb idő alatt meg tudunk oldani egy véges tartományú kényszerkielégítési problémát. A legtöbb gyakorlati alkalmazásban azonban az általános célú CSP algoritmusok megbirkóznak a 3. fejezetben bemutatott általános célú keresési algoritmusokkal megoldhatóknál több nagyságrenddel nagyobb problémákkal is.

A diszkrét változók lehetnek **végtelen tartományúak** is: például ilyen az egész számok halmaza vagy a füzerek halmaza. Például amikor egy építkezési munka naptárát ütemezzük, minden feladat kezdési ideje egy változó, amelyeknek lehetséges értékei az aktuális dátum napjait megadó egész számok. Végtelen tartományok esetén már nem lehet a kényszereket a megengedett értékkombinációk felsorolásával leírni. Ehelyett egy **kényszernyelvet (constraint language)** kell használni. Például ha  $Munka_1$ öt napig tart, és meg kell előznie  $Munka_3$ -t, akkor kényszernyelvként az algebrai egyenlőtlenségek nyelvére van szükségünk, mint például  $KezdMunka_1 + 5 \leq KezdMunka_3$ . Továbbá az ilyen kényszereket többé már nem lehet megoldani az összes lehetséges hozzárendelés felsorolásával, mert végtelen sok van belőlük. Különleges megoldó algoritmusok (melyekre itt nem térünk ki) léteznek egész értékű változók **lineáris kényszereire**, azaz olyan kényszerekre, mint a fenti, ahol minden egyik változó csak lineáris műveletekben jelenik meg. Megmutatható, hogy nem létezhet algoritmus egész értékű változók általános



**nemlineáris kényszereinek megoldására.** Néhány esetben az egész értékű kényszerproblémák egyszerűen azzal visszavezethetők véges tartományúakra, hogy korlátozzuk az összes változó értékeit. Egy ütemezési problémában például egy felső korlátot szabhatunk az összes ütemezendő munka teljes hosszára.

A folytonos tartományú kényszerkielégítési problémák elég gyakoriak a valós alkalmazásokban, és az operációkutatáson belül is tanulmányozták ezeket a problémákat. Például a Hubble űrteleszkóp kísérleteinek ütemezése a megfigyelések nagyon pontos időzítését igényli, a megfigyelések és a manőverek kezdeti és befejezési időpontjai folytonos értékű változók és csillagászati, precedencia-, valamint teljesítménnyük kényszerek sokasága vonatkozik rájuk. A folytonos tartományú kényszerkielégítési problémák legismertebb válfaja a **lineáris programozási** problémák csoportja, ahol a kényszerek *konvex* tartományt alkotó lineáris egyenlőtlenségek. A lineáris programozási feladatokat meg lehet oldani a változók száma szerinti polinomiális időben. Más típusú kényszerekkel és célfüggvényekkel rendelkező problémákat is tanulmányoztak, például ilyennel foglalkozik a kvadratikus programozás, a másodrendű kónikus programozás és hasonló társaik.

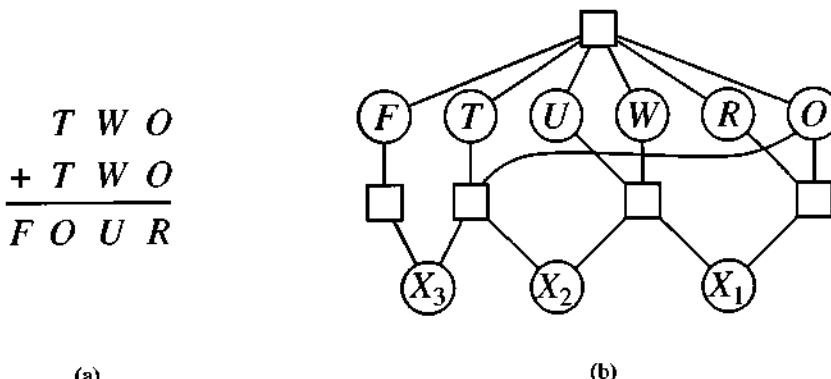
Azon túl, hogy megvizsgálhatjuk a kényszerkielégítési problémákban megjelenő változók típusát, hasznos lehet a kényszerfajtákat is tanulmányozni. A legegyszerűbb fajta az **unáris kényszer** (**unary constraint**), amely egy változó értékére tesz megkötést. Elképzelhető például, hogy a délausztrálok élénken tiltakoznak a zöld szín ellen. A hatvatzott változó tartományának előfeldolgozásával mindenki unáris kényszer kikiüszőbölhető: ki kell venni a kényszeri sértő összes értékét. A **bináris kényszer** (**binary constraint**) két változót köt össze. Például a  $DA \neq \bar{U}W$  egy bináris kényszer. A csak bináris kényszerekkel rendelkező kényszerkielégítési problémát binárisnak nevezzük, amit az 5.1. (b) ábrán láthatóhoz hasonló kényszergráftal lehet reprezentálni.

A magasabb rendű kényszerek három vagy több változóra vonatkoznak. Ennek egy ismerős példája a **betűrejtvény** (lásd 5.2. (a) ábra). A betűrejtvényben általában minden egyik betű különböző számot jelöl. Ez a megkötés az 5.2. (a) ábra betűrejtvénye esetén például a hatváltozós *MindKül(F,T,U,W,R,O)* kényszerrel adható meg. Egy másik megoldás lehet, ha a megkötést bináris kényszerek (mint például  $F \neq T$ ) gyűjteményével adjuk meg. A rejtvény négy oszlopában az összeadás kényszerek több változót is tartalmazhatnak, amelyek a következőképpen írhatók fel:

$$\begin{aligned} O + O &= R + 10 \cdot X_1 \\ X_1 + W + W &= U + 10 \cdot X_2 \\ X_2 + T + T &= O + 10 \cdot X_3 \\ X_3 &= F \end{aligned}$$

ahol  $X_1, X_2$  és  $X_3$  a következő oszlopba átvitt számjegyet (0 vagy 1) reprezentáló **segédváltozók** (**auxiliary variables**). A magasabb rendű kényszereket egy **kényszerhipergráffal** (**constraint hypergraph**) lehet ábrázolni, amelynek egy példája az 5.2. (b) ábrán látható. Az éles szemű olvasónak feltűnhet, hogy a *MindKül* kényszer felbontható bináris kényszerekre:  $F \neq T$ ,  $F \neq U$  stb. Az 5.11. feladat éppen annak bizonyítását kéri, hogy elegendő segédváltozó bevezetésével mindenki magasabb rendű véges tartományú kényszer átírható bináris kényszerek halmazává. Ebből kiindulva ebben a fejezetben csak bináris kényszerekkel foglalkozunk.

Az eddig említett kényszerek mind **abszolút kényszerek** voltak: egy kényszer megszegése kizár egy megoldásjelöltet. A valós alkalmazások során azonban sok kényszer-



**5.2. ábra.** (a) Egy betűrejtvény. Mindegyik betű különböző számjegyet jelöl, a rejtvény célja olyan számjegyeket helyettesíteni a betűk helyére, amelyekkel a kiadódó összeg aritmetikailag helyes lesz (azzal a külön megkötéssel, hogy nem kezdő nullák nem megengedettek). (b) A betűrejtvény kényszerhipergráfja a *MindKit* kényszer és az oszlopunkénti összeadási kényszerek feltüntetésével. A kényszereket négyzet alakú dobozok jelölik. minden doboz azzal a változóval van összekötve, amelyre a doboz által jelzett kényszer vonatkozik.

kielégítési probléma tartalmaz **preferenciakényszereket**, melyek jelzik, hogy mely megoldások preferáltak. Egy egyetemi órarend meghatározásakor például X professzor talán inkább reggel tanítana, míg Y professzor a délutáni órákat részesítene előnyben. Egy olyan órarend, amely szerint X professzornak délután kettőkor kellene tanítania, még megoldás lenne (persze csak ha X professzor éppenséggel nem a tanszékvezető), de ez nem lenne optimális megoldás. A preferenciakényszerek gyakran az egyedi változó-hozzárendelések költségeként ábrázolhatók: például ha X professzor egy délutáni időpontot kap, az két pontot ront az általános célfüggvényen, míg a reggeli óraidőpont csak egy pontba kerül. Így megfogalmazva a preferenciákkal kiegészített kényszerielégítési problémák (útvonalalapú vagy lokális) optimalizációs keresési módszerekkel megoldhatók. Az ilyen kényszerielégítési problémákkal a fejezetben többet nem foglalkozunk, de az irodalomjegyzékben megadunk néhány kiindulási pontként használható hivatkozást.

## 5.2. A VISSZALÉPÉSES KERESÉS ALKALMAZÁSA KÉNYSZERIELÉGÍTÉSI PROBLÉMÁKRA

Az előző alfejezetben keresési problémaként fogalmaztuk meg a kényszerielégítési problémákat. Ezt a megfogalmazást felhasználva a 3. és a 4. fejezetben bemutatott bár-mely keresési eljárással megoldhatók a kényszerielégítési problémák. Tegyük fel, hogy alkalmazzuk a szélességi keresést az előző fejezetben megadott általános CSP-megfogalmazásra. Hamarosan valami szörnyűség tünik fel: a fa tetején az elágazási tényező  $nd$ , mert a  $d$  érték bármelyike hozzárendelhető az  $n$  változó bármelyikéhez. A következő szinten az elágazási tényező  $(n-1)d$  és így tovább az  $n$  szint során. Egy  $n! \cdot d^n$  levelű fát építünk fel, noha csupán  $d^n$  lehetséges teljes hozzárendelés van!

Látszólag racionális, valójában naiv problémamegfogalmazásunk elhanyagolta az összes kényszerielégítési probléma egy döntő közös tulajdonságát: a **kommutativitást (commutativity)**. Egy probléma akkor kommutatív, ha a végeredmény szempontjából közömbös,

hogy cselekvések egy adott sorozatát milyen sorrendben alkalmazzuk. Ilyenek a kényszerkielégítési problémák is, mert a változók hozzárendelése során a sorrendtől függetlenül ugyanazt a parciális hozzárendelést kapjuk. Ezért tehát *mindegyik kényszerkielégítési problémamegoldó a következő állapot generálásakor a keresési fa minden csomóponijában csak egyetlen változó lehetséges hozzárendeléseit veszi tekintetbe*. Az ausztráliai térképszínezési probléma keresési fájának gyökérkörömpontjában választhatunk a  $DA = \text{vörös}$ , a  $DA = \text{zöld}$  és a  $DA = \text{kék}$  közül, de sohasem merül fel választási lehetőséggé a  $DA = \text{vörös}$  vagy az  $NyA = \text{kék}$ . A levelek száma ezzel a megszorítással már a remélte  $d^n$ .

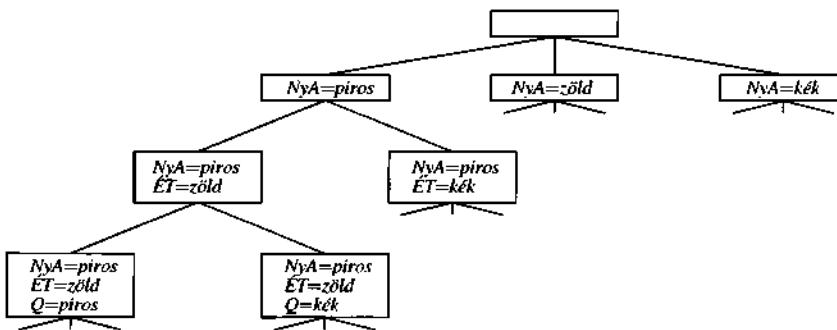
```

function VISSZALEPÉSÉS-KERESÉS(csp) returns egy megoldás vagy kudarc
    return REKURZÍV-VISSZALÉPÉSES({ }, csp)

function REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp) returns egy megoldás vagy kudarc
    if hozzárendelések teljes then return hozzárendelések
    var  $\leftarrow$  HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA(VÁLTOZÓK[csp], hozzárendelések, csp)
    for each érték in TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE(var, hozzárendelések, csp) do
        if érték konziszens a hozzárendelések-kel KÉNYSZEREK(csp) szerint then
            hozzáad {var = érték} hozzárendelések-bez
            eredmény  $\leftarrow$  REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp)
            if eredmény  $\neq$  meghiujsulás then return eredmény
            kivesz {var = érték} hozzárendelések-ból
    return kudarc

```

**5.3. ábra.** Egy egyszerű visszalépéses keresési algoritmus kényszerkielégítési problémához. Az algoritmus a 3. fejezet rekurzív mélységi keresőjén alapul. A HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA és a TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE függvények használhatók a szövegben bemutatott általános célú heurisztikák megvalósítására.



**5.4. ábra.** Részlet a keresési fából, amelyet az egyszerű visszalépéses keresés generált az 5.1. ábrán bemutatott térképszínezési problémához

A visszalépéses keresés (backtracking search) kifejezést olyan mélységi keresésekre használjuk, melyek egyszerre csak egy változóhoz rendelnek értéket, és visszalépnek, ha már nincs megengedett hozzárendelési lehetőség. Az algoritmus az 5.3. ábrán látható. Vegyük észre, hogy az algoritmus lényegében a 115–116. oldalon leírt

egyenkénti módszert használja az inkrementális következő állapot generálásra. Továbbá a következő állapot generálásakor kifejtí az aktuális állapotot, nem pedig egyszerűen másolja. A kényszerkielégítési problémák szokásos reprezentációja miatt nincs szükség a VISSZALÉPÉSES-KERESÉS algoritmus kiegészítésére tárgyterület-specifikus kezdeti állapottal, állapotátmennet-függvényel vagy céteszettel. Az 5.4. ábrán látható egy részlet az ausztráliai színezési problémához generált keresési fából (a változó-hozzárendelések sorrendje *NyA, ÉT, Q, ...*).

Az egyszerű visszalépéses keresés a 3. fejezet terminológiája szerint nem informált algoritmus, tehát nem is vártuk el, hogy nagy problémák esetén is hatékony legyen. Az 5.5. ábra első oszlopában bemutatott néhány eredmény megerősíti várakozásainkat.

A probléma	Visszalépéses	BT+MRV	Előrenéző ellenőrzés	FC+MRV	Min-Konfliktusok
USA	(> 1000K)	(> 1000K)	2K	60	64
<i>n</i> -királynő	(> 40000K)	13500K	(> 40000K)	817K	4K
Zebra	3859K	1K	35K	0,5K	2K
Véletlen (1.)	415K	3K	26K	2K	
Véletlen (2.)	942K	27K	77K	15K	

**5.5. ábra.** Az egyes CSP-algoritmusok teljesítménye különböző problémák megoldásában. Az algoritmusok balról jobbra haladva a következők: egyszerű visszalépéses, MRV-heurisztikával bővített visszalépéses, előrenéző ellenőrzés. MRV-heurisztikával kiegészített előrenéző ellenőrzés és a minimális konfliktusok lokális keresése. Az egyes cellákban a probléma megoldása során elvégzett konziszencia-ellenőrzések száma található (öt futtatás mediánértekéből meghatározva): figyeljünk fel arra, hogy a két jobb felső cella kivételével mindenkor érték ezrekben van megadva (ezt jelzi a K betű). Az első probléma az Amerikai Egyesült Államok 50 államát tartalmazó térkép négy színnel történő színezése. A többi probléma a (Bacchus és van Run, 1995)-ből származik (1. táblázat). A második probléma az *n*-királynő megoldásához szükséges ellenőrzések száma (*n* értékeire 2-től 50-ig összegezve). A harmadik az 5.13. feladatban is szereplő „zebrarejtvény”. A maradék kettő mesterséges véletlenszerűen generált problémák. (A min-konfliktusok algoritmus ezeken nem futott.) Az eredmények azt érzékelhetik, hogy az MRV-heurisztikával kombinált előrenéző ellenőrzés mindenkor probléma esetén jobban teljesít, mint a többi visszalépéses algoritmus, de nem minden esetben jobb a min-konfliktusokat használó helyi keresésnél.

A 4. fejezetben a nem informált keresési algoritmusok gyenge teljesítményét a probléma ismeretéből származó tárgyterület-specifikus heurisztika bevezetésével orvosoltuk. Az derül azonban ki, hogy a kényszerkielégítési problémák hatékonyan megoldhatók ilyen tárgyterület-specifikus tudás nélkül is. Olyan általános célú eljárások alakíthatók ki, amelyek az alábbi kérdéseket vizsgálják:

1. A következő lépésekben melyik változóhoz rendeljünk értéket, és milyen sorrendben próbálkozzunk az értékekkel?
2. Milyen következményei vannak a jelenlegi változó-hozzárendeléseknek a még hozzárendeletlen változók számára?
3. Ha egy út sikertelennek bizonyul (azaz egy olyan állapothoz jutunk, ahol egy változónak nincs megengedhető értéke), a következő utak során el tudja-e kerülni a keresés ezt a hibát?

A következő pontokban megpróbáljuk sorra megválaszolni ezeket a kérdéseket.

## Változó- és értékrendezés

A visszalépéses algoritmus tartalmazza az alábbi sort:

```
var ← HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁS(VÁLTOZÓK[csp], hozzárendelések, csp)
```

Alapértelmezés szerint a HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁS egyszerűen ki-választja a VÁLTOZÓK[*csp*] lista sorrendje szerinti következő hozzárendeletlen változót. A statikus változó-hozzárendelés ritkán vezet a leghatékonyabb kereséshez. Például miután kiválasztottuk az  $NyA = \text{vörös}$  és az  $\bar{E}T = \text{zöld}$  hozzárendeléseket, csak egyetlen értéket rendelhetünk *DA*-hoz, tehát ésszerű lenne inkább a *DA* = *kék* hozzárendelést elvégezni, mintsem *Q* számára keresni értékét. Sőt miután *DA* értéket kapott, kényszerítve vagyunk *Q*, *ÜDW* és *V* értékeinek kiválasztásakor. Ezt az intuitív ötletet (a legkevesebb „megengedett” értékkel rendelkező változó kiválasztását) nevezik **legkevesebb fennmaradó érték (minimum remaining values, MRV)** heurisztikának. Neveztek már „legkorlátozottabb változó” vagy „meghiúsulási előre” heurisztikának is (az utóbbi névnek az a magyarázata, hogy azt a változót emeli ki, amelyik legvalószínűbben fog hamarosan hibához vezetni, ezáltal megnyesve a keresési fát). Ha van egy *X* változó, amelynek egyetlen megengedett értéke sincs, akkor az MRV-heurisztika ki fogja választani *X*-et, és azonnal kideríti a hibát, elkerülve ezzel a többi változó közötti értelmetlen keresgést, ami minden kudarcra vezet, amikor *X* végül is kiválasztódik. Az 5.5. ábra második, BT+MRV című oszlopa ennek a heurisztikának a teljesítményét mutatja. Az egyszerű visszalépéses keresésnél a teljesítmény a problémától függően mintegy 3...3000-szer jobb lehet. Vegyük észre, hogy a teljesítménymértekünk nem tartalmazza a heurisztikus értékek kiszámításának többletköltségét. A következő pontban bemutatunk egy módszert, amellyel kezelhetővé válik ez a költség.

Az MRV-heurisztika semmit sem segít abban, hogy melyik régiót válasszuk ki elsőként Ausztrália kiszínezésekor, mert a kiinduláskor minden egyik régiónak három megengedett színe van. Ebben az esetben segíthet a **fokszám-heurisztika (degree heuristics)**. Ez azáltal kísérli meg a későbbi választások elágazási tényezőjét csökkenteni, hogy azt a változót választja ki, amely a legtöbbször szerepel a hozzárendeletlen változókra vonatkozó kényszerekben. Az 5.1. ábrán *DA* a legnagyobb fokszámú változó (fokszáma 5), az összes többi változó fokszáma 2 vagy 3, leszámítva *T*-t, amelyé nulla. Valójában, ha már egyszer *DA*-t kiválasztottuk, a fokszám-heurisztika alkalmazása a problémát egyetlen hibás lépés nélkül oldja meg: minden egyik választási pontnál bármelyik konziszens színt is választjuk, mindenéppen visszalépés nélkül jutunk egy megoldáshoz. A legkevesebb fennmaradó érték heurisztika általában jobban irányít, de a patthelyzetek előtétiséhez hasznos segítség lehet a fokszám-heurisztika.

Miután az algoritmus már kiválasztott egy változót, el kell döntenie, hogy milyen sorrendben vizsgálja meg ennek értékeit. Erre néha a **legkevésbé-korlátozó-érték (least-constraining-value)** heurisztika lehet hasznos. Ez a heurisztika előnyben részesíti azt az értéket, amely a legkevesebb választást zárja ki a kényszergráfban a szomszédos változóknál. Például tegyük fel, hogy az 5.1. ábra esetében már létrehoztuk az  $NyA = \text{vörös}$  és az  $\bar{E}T = \text{zöld}$  parciális hozzárendelést, és most *Q* számára akarunk értéket találni. A kék rossz választás volna, mert ez kizáraja *Q* szomszédjának, *DA*-nak utolsó megengedett értékét. A legkevésbé korlátozó heurisztika ezért a kékkel szemben a vöröset részesít előnyben. Általában véve a heurisztika megpróbálja a későbbi változó-

hozzárendelések számára a lehető legnagyobb szabadságot meghagyni. Természetesen ha egy probléma összes megoldását meg szeretnénk találni, és nem elégünk meg csupán egyet, akkor a sorrend közömbös, hiszen úgyis minden értéknek sorra kell kerülnie. Ugyanez igaz akkor is, ha a problémának nincsen megoldása.

## Az információ terjesztése a kényszereken keresztül

A keresési algoritmusunk eddig csak akkor foglalkozott egy változóra vonatkozó kényszerttel, ha a változót a HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁS kiválasztotta. De ha néhány kényszert a keresés folyamán korábban vagy akár még a keresés megkezdése előtt megvizsgálunk, akkor drasztikusan csökkenhető a keresési tér mérete.

### Előrenéző ellenőrzés

A kényszerek keresés közbeni jobb felhasználásának egyik módszerét **előrenéző ellenőrzésnek (forward checking)** nevezzük. Az előrenéző ellenőrzési folyamat minden egyes alkalommal, amikor egy  $X$  változó értéket kap, minden, az  $X$ -hez kényszerrrel kötött, hozzárendeletlen  $Y$ -t megvizsgál, és  $Y$  tartományából törli az  $X$  számára választott értékkel inkonzisztens értékeket. Az 5.6. ábra bemutatja az előrenéző ellenőrzéses keresés menetét a térképsínezési probléma során. Két fontos dolgot kell megemlítenünk ezzel a példával kapcsolatban. Először is vegyük észre, hogy miután elvégeztük az  $NyA = \text{vörös}$  és a  $Q = \text{zöld}$  hozzárendeléseket,  $\bar{ET}$  és  $DA$  tartományai egyetlen elemre szűkültek, és azzal, hogy az  $NyA$ -ból és a  $Q$ -ból származó információt terjesztettük, teljesen megszüntettük az ezen változók szerinti elágazást. Az MRV-heurisztika, amely az előrenéző ellenőrzés nyilvánvaló partnere, automatikusan  $DA$ -t és  $\bar{ET}$ -t választja következőnek. (Tulajdonképpen még hatékonyabb eljárássá tehetjük az előrenéző ellenőrzést, ha az MRV-heurisztika munkájához szükséges információt inkrementálisan számítjuk.) A másik dolog, amire érdemes felfigyelni, az az, hogy a  $V = \text{kék}$  hozzárendelés után  $DA$  tartománya üres lett. Ezért az előrenéző ellenőrzés megállapítja, hogy a  $\{NyA = \text{vörös}, Q = \text{zöld}, V = \text{kék}\}$  részleges hozzárendelés inkonzisztens a probléma kényszereivel, és az algoritmus azonnal visszalép.

### A kényszerek terjesztése

Az előrenéző ellenőrzés ugyan sok inkonziszenciát észrevevesz, de nem minden. Például nézzük csak az 5.6. ábra harmadik sorát. Az látható itt, hogy amikor  $DA$  vörös és  $Q$  zöld, mind  $\bar{ET}$ -nek, mind  $DA$ -nak kéknak kell lennie. De ezek szomszédosak, és nem lehet azonos az értékük. Az előrenéző ellenőrzés nem veszi észre ezt az inkonziszenciát, mert nem néz elég messze előre. A kényszerek terjesztése (**constraint propagation**) általános kifejezés jelöli azt, ha az egyik változó kényszerének a többi változót érintő következményeit terjesztjük: esetünkben  $NyA$ -ról  $Q$ -ra és  $\bar{ET}$ -ről  $DA$ -ra kell terjeszteni (ahogy ezt már az előrenéző ellenőrzés is megtette), majd az  $\bar{ET}$  és  $DA$  közti kényszerre, hogy észrevehessük az inkonziszenciát. Ezt pedig gyorsan szeretnénk tenni: semmi értelme

	NyA	ÉT	Q	ÚDW	V	DA	T
Kezdeti tartományok	V Z K	V Z K	V Z K	V Z K	V Z K	V Z K	V Z K
Miután NyA = vörös	(V)	Z K	V Z K	V Z K	V Z K	Z K	V Z K
Miután Q = zöld	(V)	K	(Z)	V	K	V Z K	K
Miután V = kék	(V)	K	(Z)	V		(K)	V Z K

**5.6. ábra.** A térképsínezés folyamata előrenéző ellenőrzéses keresés esetén. Az első hozzárendelés az NyA = vörös, majd az előrenéző ellenőrzés törli a vörös értéket a szomszédos ÉT és DA változók tartományáiból. A Q = zöld hozzárendelés után a zöld is kikerül az ÉT, DA és ÚDW változók tartományaiából. A V = kék-et követően a kék is törölődik a DA és ÚDW változók tartományaiából, és DA számára nem marad megengedett érték.

a keresés méretét csökkenteni, ha több időt töltünk a kényszerek terjesztésével, mint az egyszerű kereséssel tennénk.

Az **élkonzisztencia** (arc consistency) alapul szolgálhat egy gyors, az előrenéző ellenőrzésnél lényegesen erősebb kényszerterjesztéshez. Az „él” itt a kényszergráf irányított éleit jelenti, amilyen például a DA-ból az ÚDW-be mutató él. Ha adott DA és ÚDW aktuális tartománya, akkor ez az él konzisztens, ha DA mindegyik x értékéhez található egy x-szel konzisztens valamely y érték ÚDW-ben. Az 5.6. ábra harmadik sorában DA és ÚDW pillanatnyi tartománya rendre a {kék}, illetve a {vörös, kék}. A DA = kék esetén található egy konzisztens hozzárendelés ÚDW-hez, nevezetesen az ÚDW = vörös; ezért a DA-ból ÚDW-be mutató él konzisztens. Viszont az ÚDW-ból a DA-ba mutató él nem konzisztens, mert az ÚDW = {kék} hozzárendelésnél a DA-hoz nem tudunk szint találni. Az él úgy tehető konzisztenssé, hogy az ÚDW tartományából töröljük a kék értéket.

A DA-ból ÚT-be mutató él mentén is alkalmazhatjuk a keresési folyamat ugyanazon fázisában az élkonzisztenciát. Az 5.6. ábra harmadik sora szerint minden változó értéktartománya {kék}. Ez azt eredményezi, hogy törölünk kell a kék-et DA tartományából, amely ezzel kiürül. Az élkonzisztencia ellenőrzés tehát lehetővé teszi, hogy korábban észrevegyük az egyszerű előrenéző ellenőrzés által fel nem fedett inkonzisztenciát.

Az élkonzisztencia-ellenőrzés alkalmazható előfeldolgozó lépésként a keresés megkezdése előtt, vagy a keresési folyamat minden egyes hozzárendelést követő terjesztési lépésként (az előrenéző ellenőrzéshez hasonlóan). (Az utóbbi algoritmust **Élkonzisztencia fenntartásának** is, angol rövidítéssel MAC-nak nevezik.) Mindkét esetben addig kell ismételte alkalmazni a folyamatot, amíg nem marad inkonzisztencia. Erre azért van szükség, mert amikor egy élkonzisztenciát eltávolítandó egy érték kikerül egy változó tartományából, az ehhez a változóhoz mutató éleknek új inkonzisztencia jöhet létre. Az AC-3, az élkonzisztencia teljes algoritmusára egy sort használ annak nyilvántartására, hogy mely élek inkonzisztenciáját kell még ellenőriznie (lásd 5.7. ábra). minden egyes  $(X_i, X_j)$  élet sorban egyenként levessük a tennivalók listájáról és ellenőrizzük; ha pedig  $X_i$  tartományának bármely változóját törölni kell, akkor minden  $X_i$ -be mutató  $(X_k, X_i)$  élet visszateszünk ellenőrzésre a sorba. Az élkonzisztencia-ellenőrzés komplexitása az alábbiak szerint vizsgálható: egy bináris kényszerkielégítési problémában legfeljebb  $O(n^2)$  él van; minden egyes  $(X_k, X_i)$  él csak  $d$  alkalommal kerülhet napirendre, mert  $X_i$ -ben összesen  $d$  törölhető érték van; egy él konzisztenciájának ellenőrzése elvégezhető  $O(d^2)$

```

function AC-3(csp) returns a CSP-t, lehetőleg redukált tartományokkal
  inputs: csp, egy bináris CSP  $\{X_1, X_2, \dots, X_n\}$  változókkal
  local variables: sor, a sorra következő élek, kiindulásként csp összes éle

  while sor nem üres do
     $(X_i, X_j) \leftarrow \text{ELŐT-KIVESZ}(sor)$ 
    if INKONZISZTENS-ÉRTÉKEKET-KIVESZ( $X_i, X_j$ ) then
      for each  $X_k$  in SZOMSZÉDAL( $X_i$ ) do
        az  $(X_k, X_i)$  belekerül a sor-ba

function INKONZISZTENS-ÉRTÉKEKET-KIVESZ( $X_i, X_j$ ) returns igaz, a.cs.a. ha kiveszünk egy értéket
  eltávolítva  $\leftarrow$  halmis
  for each x in TARTOMÁNY[ $X_i$ ] do
    if a TARTOMÁNY[ $X_j$ ] egyetlen y értéke esetén sem elégít ki  $(x, y)$  az  $X_i$  és  $X_j$  közti kényszert
    then x törlése TARTOMÁNY[ $X_i$ ]-ból: eltávolítva  $\leftarrow$  igaz
  return eltávolítva

```

**5.7. ábra.** Az AC-3 élkonzisztenzia algoritmus. Az AC-3 alkalmazását követően vagy minden egyik él élkonzisztenst, vagy néhány változó tartománya üres, azaz a kényszerkielégítési probléma nem hozható élkonzisztenst alakra (tehát nem oldható meg). Az „AC-3” nevet az algoritmus kitalálója (Mackworth, 1977) vezette be, mert ezt az algoritmust mutatta be cikkében harmadikként.

időben; tehát a legrosszabb esetben vett teljes idő  $O(n^2d^3)$ . Ez ugyan lényegesen költségesebb, mint az előretekintő ellenőrzés, de az extra költség általában kifizetődik.<sup>1</sup>

Mivel a kényszerkielégítési problémák speciális esetként tartalmazzák a 3SAT-ot, ezért nem várhatjuk, hogy polinom idejű algoritmust találunk annak előntésére, hogy egy adott kényszerkielégítési probléma konziszts-e. Ebből arra következtethetünk, hogy az élkonzisztenzia nem tár fel minden lehetséges inkonziszenciát. Például az 5.1. ábrán az  $\{NyA = \text{vörös}, \text{ÚDW} = \text{vörös}\}$  részleges hozzárendelés inkonziszts, de az AC-3 nem találja meg. Definiálható a terjesztés erősebb formája is, amit *k-konziszenciának* (*k-consistency*) nevezünk. Egy kényszerkielégítési probléma akkor *k-konziszts*, ha bármely  $k - 1$  változóból álló halmaz és ezen változók bármely konziszts hozzárendelése esetén minden lehet a *k*-adik változónak konziszts értéket találni. Az 1-konzisztenzia például azt jelenti, hogy minden individuális változó önmagában véve is konziszts; ezt **csomópont-konziszenciának** (**node consistency**) is nevezik. A 2-konzisztenzia ugyanaz, mint az élkonzisztenzia. A 3-konzisztenzia azt jelenti, hogy egymás melletti változók bármely párosa minden kiterjeszhető egy szomszédos változóra; ezt **útvonal-konziszenciának** (**path consistency**) is nevezik.

Egy gráf erősen *k-konziszts* akkor, ha *k*-koniszts, valamint  $(k - 1)$ -koniszts,  $(k - 2)$ -koniszts és így tovább le egészen az 1-konziszenciáig. Tegyük fel most, hogy van egy  $n$  csomópontból álló kényszerkielégítési problémánk, és tegyük erősen  $n$ -konisztenssé (azaz *k*-konisztenssé  $k = n$  esetében). Ekkor visszalépés nélkül megoldhatjuk a problémát. Először egy konziszts értéket választunk  $X_1$  számára. Ezután garanciánk van arra, hogy tudunk  $X_2$  számára értéket választani, mert a gráf 2-koniszts,  $X_3$  számára, mert a gráf 3-koniszts és így tovább. minden egyes  $X_i$  változó

<sup>1</sup> A Mohr és Henderson (Mohr és Henderson, 1986) bemutatta AC-4 algoritmus  $O(n^2d^2)$  alatt fut. (Lásd 5.10. feladat.)

esetén csak  $d$  értéket kell ellenőriznünk a tartományból, hogy egy  $X_1, \dots, X_{i-1}$ -gyel konzisztens értéket találunk. Garanciánk van tehát arra, hogy  $O(nd)$  időben megoldást találunk. Persze semmit sem adnak ingyen: az  $n$ -konisztenziát biztosító bármely algoritmus legrosszabb esetben  $n$ -ben exponenciális lesz.

Az  $n$ -konisztenzia és az élkonisztencia között egy széles köztes mező húzódik: az erősebb konisztenzia-ellenőrzések futtatása tovább fog tartani, de több eredménnyel jár az elágazási tényező csökkentésében és az inkonisztenzs részleges hozzárendelések feldejrítésében. Ki lehet számolni a legkisebb olyan  $k$  értéket, melyre a  $k$ -konisztenzia ellenőrzésének futtatása visszalépés nélküli problémamegoldást tesz lehetővé (lásd 5.4. alfejezet), de ez gyakran nem célszerű. A legmegfelelőbb konisztenzia-ellenőrzési szint kiválasztása valójában leginkább empirikus tudomány.

### Speciális kényszerek kezelése

A kényszerek bizonyos típusai gyakran fordulnak elő a valós problémákban és speciális célú algoritmusokkal jóval hatékonyabban kezelhetők, mint az eddig leírt általános célú módszerekkel. Például a *MindKül* kényszer azt állítja, hogy a benne szereplő összes változóknak minden különböző értéket kell felvennie (ahogyan a betűrejtvényes példában láttuk). A *MindKül* kényszer inkonisztenzia-ellenőrzésére egy egyszerű módszer az, hogy ha  $m$  változó szerepel a kényszerben, és ha ezeknek együttesen  $n$  különböző értéke lehet, akkor  $m > n$  esetén a kényszert nem lehet kielégíteni.

Ez a következő egyszerű algoritmushoz vezet: először vegyünk ki a kényszerből minden változót, amelynek egyelemű tartománya van, és vegyük ki ezeknek a változóknak az értékeit a megmaradó változók tartományiból. Ismételjük ezt a lépést mindenkor, amíg van ilyen változónk. Ha bármely ponton egy üres tartomány jön létre, vagy több változónk marad, mint értékünk a tartományban, akkor egy inkonisztenciát derítettünk fel.

Ezt a módszert használhatjuk arra, hogy az 5.1. ábra ( $NyA = vörös$ ,  $\bar{U}DW = vörös$ ) részleges hozzárendelésében észrevegeyük az inkonisztenciát. Figyeljünk fel arra, hogy a  $DA$ , az  $\bar{U}T$  és a  $Q$  változók lényegében egy *MindKül* kényszerrel vannak összekötve, mert mindenkorik párnak különböző színűnek kell lennie. Miután alkalmaztuk az AC-3-at a részleges hozzárendelésre, az egyes változók tartománya a  $\{zöld, kék\}$  tartományra szűkült le. Azaz három változónk van és csupán két színünk, tehát megszegtük a *MindKül* kényszert. Tehát a magasabb szintű kényszerekre alkalmazott egyszerű konisztenzia-ellenőrzések néha hatékonyabbak, mintha az ekvivalens bináris kényszerek halmazára vizsgáltuk volna az élkonisztenciát.

A legfontosabb magasabb rendű kényszer talán az **erőforráskényszer** (resource constraint), amit néha *legfeljebb* kényszernak is neveznek. Jelölje például  $PA_1, \dots, PA_4$  rendre négy feladathoz hozzárendelt személyzet számát. Azt a kényszert, miszerint összesen nem lehet 10-nél több embernek feladatot adni, így írhatjuk fel: *legfeljebb*(10,  $PA_1, PA_2, PA_3, PA_4$ ). Az inkonisztenciát egyszerűen ki lehet mutatni az aktuális tartományok minimális értékeinek összegzésével: ha például mindenkorik változó tartománya a  $\{3, 4, 5, 6\}$ , akkor a *legfeljebb* kényszert nem lehet kielégíteni. A konisztenciát azáltal is érvényesíthetjük, hogy töröljük bármely tartomány maximális értékét, amely nem konisztenzs az összes többi tartomány minimális értékével. Így ha példánkban

mindegyik változónak  $\{2, 3, 4, 5, 6\}$  a tartománya, akkor az 5 és a 6 értékeket minden egyik tartományból törölni lehet.

A nagy erőforrás-korlátozott egész értékű problémáknál – ilyenek például az emberek ezreit és járművek százait mozgató logisztikai problémák – általában nem tehető meg az, hogy minden változó tartományát egy nagy egész halmazzal ábrázoljuk, majd ezt a halmazt a konziszencia-ellenőrző módszerek segítségével fokozatosan csökkentjük. Ehelyett a tartományokat az alsó és a felső határaikkal ábrázoljuk, és határterjesztés segítségével kezeljük. Tegyük fel például, hogy két repülőjárat van, a 271-es és a 272-es, amelyekre a repülőgépek kapacitása rendre 165 és 385. A repülőjárat utasszámának kezdeti tartománya tehát

$$\text{Járat271} \in [0, 165] \text{ és } \text{Járat272} \in [0, 385]$$

Tegyük fel most, hogy van egy külön korlát, miszerint a két járatnak együtt 420 embert kell szállítania.  $\text{Járat271} + \text{Járat272} \in [420, 420]$ . A határkényszereket terjesztve a tartományokat a következőre csökkentjük:

$$\text{Járat271} \in [35, 165] \text{ és } \text{Járat272} \in [225, 385]$$

Akkor nevezünk egy kényszerkielégítési problémát határkonziszensnek, ha minden  $X$  változóra ennek a változónak mind az alsó, mind a felső határértékére található minden  $Y$  változóhoz olyan érték, amely kielégíti az  $X$  és  $Y$  közti kényszereket. Ezt a fajta **kényszerterjesztést (bound propagation)** széles körben használják a gyakorlati kényszerkielégítési problémák során.

## Intelligens visszalépés: visszanézni

Az 5.3. ábra VISSZALÉPÉSES-KERESÉS algoritmusára egy elég egyszerű intézkedést alkalmaz akkor, amikor egy keresési ág meghiúsul: visszalép az előző változóra, és megpróbál számára egy másik értéket találni. Ezt **időrendi visszalépésnek (chronological backtracking)** nevezik, mert a *legutolsó* döntési pontot keresi fel újra. Ebben az alrészben látni fogjuk, hogy adódik erre sokkal jobb módszer is.

Nézzük meg, mi történik, amikor az egyszerű visszalépéses algoritmust az 5.1. ábrán látható problémára alkalmazzuk rögzített változósorrenddel ( $Q, \text{ÚDW}, V, T, DA, NyA, UT$ ). Tegyük fel, hogy már létrehoztuk a  $\{Q = \text{vörös}, \text{ÚDW} = \text{zöld}, V = \text{kék}, T = \text{vörös}\}$  hozzárendelést. Amikor a következő változóval,  $DA$ -val próbálkozunk, azt látjuk, hogy minden egyik érték sérti a kényszert. Visszalépünk  $T$ -re, és egy új szint keresünk Tasmania számára. Ez nyilvánvalóan butaság: Tasmania átszínezése nem oldja meg a Dél-Ausztráliával kapcsolatos problémát.

A visszalépéses megközelítés intelligens alkalmazása lenne, ha ahhoz a változóhalmazhoz mennénk vissza, amely a *meghiúsulást okozta*. Ezt a halmazt **konfliktushalmaznak (conflict set)** nevezik; esetünkben a  $DA$ -hoz tartozó konfliktushalmaz a  $\{Q, \text{ÚDW}, V\}$ . Általános esetben egy  $X$  változó konfliktushalmaza a korábban értéket kapott változók egy halmaza, melyeket  $X$ -hez kényszerek kötnek. A **visszaugrás (backjumping)** a konfliktushalmaz-beli legutolsó változóhoz lép vissza; esetünkben a visszaugrás átugorná Tasmaniát, és  $V$  számára próbálna új értéket keresni. Ezt egyszerű implementálni a VISSZALÉPÉSES-KERESÉS olyan módosításával, hogy az összegyűjtse a konfliktushalmazt,

miközben hozzárendelhető értékeket keres. Ha nem talál ilyen értéket, akkor a konfliktushalmaz legutolsó elemét kell visszaadnia (a meghíúsulás jelzésével együtt).

Az éles szemű olvasónak feltűnhetett, hogy az előrenéző ellenőrzés a konfliktushalmaz minden külön munka nélkül előállíthatja: minden amikor az előrenéző ellenőrzés  $X$  egy hozzárendelésén alapulva  $Y$  tartományából kitöröl egy elemet,  $X$ -et hozzá kell adnia  $Y$  konfliktushalmazához. Ezenkívül még minden esetben, amikor az utolsó értéket törli  $Y$  tartományából, az  $Y$  konfliktushalmazában szereplő változókat hozzá kell adnia  $X$  konfliktushalmazához. Ezek után mihelyt  $Y$ -hoz jutunk, már azonnal ismerni fogjuk, hogy hova lépjünk vissza, ha erre szükség van.

A sasszemű olvasónak valami furcsaság is feltűnhetett: a visszaugrás akkor következik be, amikor egy tartomány minden értéke konfliktusban van az aktuális hozzárendelésekkel, de az előrenéző ellenőrzés felismeri ezt az eseményt, és megakadályozza, hogy valaha is ilyen csomóponthoz jussunk! Valóban, meg lehet mutatni, hogy a visszaugrás által levágott minden ágat az előrenéző ellenőrzés is levágja. Tehát az egyszerű visszaugrás redundáns az előrenéző ellenőrzés során, vagy pontosabban szólva, egy olyan keresésnél, amely erősebb konzisztszencia-ellenőrzést, például MAC-ot használ.

Az előző bekezdés megfigyelése ellenére a visszaugrás mögötti ötlet továbbra is használható marad: lépjünk vissza a hiba helyére. A visszaugrás akkor veszi észre a kudarcot, amikor egy változó értéktartománya üressé válik, de sok esetben egy ág már sokkal korábban levágásra lett ítélt. Tekintsük ismét az  $\{NyA = \text{vörös}, \text{ÚDW} = \text{vörös}\}$  részleges hozzárendelést (amelyik, korábbi megjegyzésekben, inkonzisztens). Tegyük fel, hogy a  $T = \text{vörös}-t$  próbáljuk, majd az  $\text{ÚT}, Q, V, DA$ -kra alkalmazzunk hozzárendeléseket. Tudjuk, hogy az utóbbi négy változó esetén semmilyen hozzárendelés sem lesz jó, így végül  $\text{ÚT}$  próbálhatásánál kifogyunk az értékekből. A kérdés most az, hogy hova lépjünk vissza? A visszaugrás nem fog működni, mert  $\text{ÚT}$ -nek *igenis* van az előző hozzárendelésekkel konzisztens értéke:  $\text{ÚT}$  esetén nincsen teljes konfliktushalmaz a megelőző változók között, amely felelős lenne a kudarcért. Tudjuk azonban, hogy a négy változó,  $\text{ÚT}, Q, V$ , és  $DA$  együttvéve a megelőző változók egy halmaza miatt hiúsul meg, és ezek azok a változók, amelyek közvetlenül konfliktusban állnak a kérdéses négygyel. Ennek alapján  $\text{ÚT}$  konfliktushalmazának egy mélyebb megértéséhez jutottunk: a konfliktushalmaz a megelőző változók azon halmaza, ami *bármely következő változóval együtt* felelős  $\text{ÚT}$  konzisztens megoldásának kudarcáért. Esetünkben ez a halmaz az  $NyA$  és az  $\text{ÚDW}$ , tehát az algoritmusnak  $\text{ÚDW}$ -hez kell visszalépnie átugorva Tasmaniát. A konfliktushalmaz ilyen definíciójára támaszkodó visszaugrási algoritmust nevezük **konfliktusvezérelt visszaugrásnak (conflict-directed backjumping)**.

Magyarázatot kell most adnunk arra, hogy miként lehet kiszámítani ezeket az újfajta konfliktushalmazokat. A módszer valójában nagyon egyszerű. Egy keresési ág „terminálási” kudarca minden azért következik be, mert egy változó tartománya kiürül, ennek a változónak pedig egy szokásos konfliktushalmaza van. Példánkban  $DA$  hiúsul meg, és konfliktushalmaza (mondjuk) az  $\{NyA, \text{ÚT}, Q\}$ . Visszaugrunk  $Q$ -ra, és  $Q$  beépíti saját közvetlen konfliktushalmazába (ami az  $\{\text{ÚT}, \text{ÚDW}\}$ )  $DA$  konfliktushalmazát (leszámítva persze  $Q$ -t magát), és így előáll az  $\{NyA, \text{ÚT}, \text{ÚDW}\}$  konfliktushalmaz. Azaz  $Q$ -től számítva tovább már nincsen megoldás feltételezve  $\{NyA, \text{ÚT}, \text{ÚDW}\}$  korábbi hozzárendeléseit. Tehát visszalépünk  $\text{ÚT}$ -re, amelyik a legutóbbi ezek közül.  $\text{ÚT}$  felveszi  $\{NyA, \text{ÚT}, \text{ÚDW}\} - \{\text{ÚT}\}$ -t a saját közvetlen konfliktushalmazába,  $\{NyA\}$ -ba, és előáll az  $\{NyA, \text{ÚDW}\}$  konfliktushalmaz (ahogyan az előző bekezdésben állítottuk). Foglaljuk

össze az eddig mondottakat: jelölje  $X_j$  az aktuális változót és  $conf(X_j)$  a saját konfliktushalmazát. Ha  $X_j$  minden lehetséges értéke meghiúsul, ugorunk vissza a  $conf(X_j)$ -beli legutolsó  $X_i$  értékre és legyen

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$$

A konfliktusvezérelt visszaugrás a keresési fa megfelelő helyére visz vissza, de nem akadályozza meg, hogy ugyanazt a hibát elkövessük a fa egy másik ágán. A kényszer-tanulás (**constraint learning**) a konfliktusból kiemelt új kényszer megtanulásával módosítja a kényszerkielégítési problémát.

## 5.3. LOKÁLIS KERESÉS KÉNYSZERKIELÉGÍTÉSI PROBLÉMÁKNÁL

A lokális keresési algoritmusok (lásd 4.3. alfejezet) nagyon hatékonyak bizonyulnak sok kényszerkielégítési probléma megoldásában: a kiinduló állapot minden változóhoz értéket rendel, és az állapotátmennet-függvény működése során általában egyszerre csak egy változó értékét módosítja. A 8-királynő problémában például a kezdeti állapot lehet a 8 királynő véletlenszerű elhelyezése a 8 oszlopban, az állapotátmennet-függvény pedig kiemeli egy királynőt, és megpróbálja a saját oszlopán belül máshova helyezni. Egy másik lehetőség lehetne, ha úgy indulnánk, hogy a 8 királynő mindegyikét egy külön oszlopba a nyolc sor permutációjával helyezzük el, és az a következő állapotokat pedig két királynő sorának felcseréléssel generálnánk.<sup>2</sup> Tulajdonképpen már láttunk egy példát lokális keresés alkalmazására egy CSP-probléma megoldásánál: ilyen volt a hegymászás alkalmazása a 8-királynő probléma megoldására (lásd 155. oldal). Egy másik alkalmazás a WALKSAT (lásd 278. oldal) a kielégíthetőségi probléma megoldására, mely szintén egy speciális esete a kényszerkielégítési problémáknak.

Amikor egy változónak új értéket választunk, a legnyilvánvalóbban kínálkozó heurisztika annak az értéknek kiválasztása, amelyik a legkevesebb konfliktust eredményezi más változókkal. Ezt nevezik **min-konfliktusok** (**min-conflicts**) heurisztikának. Az algoritmus az 5.8. ábrán látható, a 8-királynő problémára történő alkalmazását az 5.9. ábrán mutatjuk be, a kiértékelés eredménye pedig az 5.5. ábrán szerepel.

A min-konfliktusok heurisztika meglepően hatékony sok kényszerkielégítési probléma megoldásában, különösen amikor adott egy viszonylag jó kezdeti állapot. Teljesítményét az 5.5. ábra utolsó oszlopa mutatja. Meglepő módon az  $n$ -királynő probléma esetén a min-konfliktusok algoritmus futási ideje a kezdeti elhelyezéseket nem számítva nagyjából független a probléma méretétől. Akár a millió-királynő problémát is megoldja átlagosan ötven lépében (a kezdeti értékkedést követően). Ez a figyelemre méltó megfigyelés a kilencvenes években komoly kutatómunkához vezetett a lokális keresés, valamint a könnyű és nehéz problémák megkülönböztetése terén (ezzel a 7. fejezetben még foglalkozunk). Durván szólva az  $n$ -királynő azért könnyű a lokális keresés számára, mert a megoldások az állapottérben sűrűn helyezkednek el. A min-konfliktusok

<sup>2</sup> A lokális keresés a célfüggvény bevezetésével könnyen kiterjeszthető egy kényszerkielégítési problémává. Ebben az esetben a hegymászásra és a szimulált lehűtéstre használt összes technika alkalmazható a célfüggvény optimalizálására.

```

function MIN-KONFLIKTUSOK(csp, max_lépések) returns egy megoldás vagy kudarc
  inputs: csp, egy kényszerkielégítési probléma
    max_lépések, a lépések megengedett száma, mielőtt feladnánk

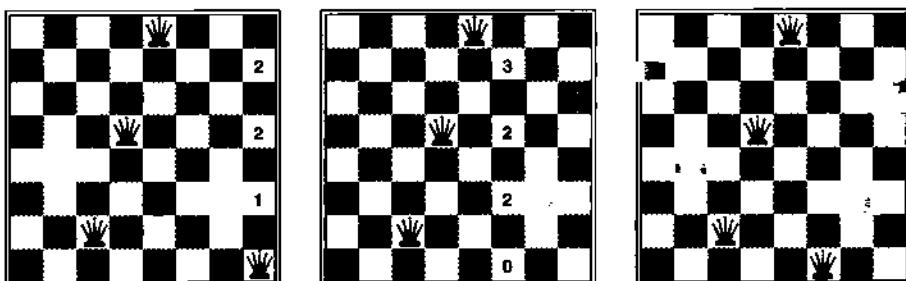
  aktuális  $\leftarrow$  csp egy kezdeti teljes hozzárendelése
  for i = 1 egészen max_lépések-ig do
    if aktuális egy megoldása csp-nek then return aktuális
    vált  $\leftarrow$  egy véletlenszerűen kiválasztott, konfliktusban lévő változó VÁLTOZÓK[csp]-ból
    érték  $\leftarrow$  azon v értéke vált-nak, amely minimalizálja KONFLIKTUSOK(vált, v, aktuális, csp)-t
    aktuális-ban állítsa vált-ot érték-re
  return kudarc

```

**5.8. ábra** A MIN-KONFLIKTUSOK algoritmus kényszerkielégítési problémák megoldására lokális kereséssel. A kezdeti állapotot véletlenszerűen választjuk ki, vagy egy mohó hozzárendelési folyamat segítségével, amely minden egyes sorra kerülő változónak egy minimális-konfliktus értéket választ. A KONFLIKTUSOK függvény megszámolja az adott érték által megsérült kényszerek számát (adottnak tekintve az aktuális hozzárendelések fennmaradó részét).

algoritmus nehéz problémák esetén is működik. Alkalmazták például a Hubble-űrtávcső megfigyeléseinek ütemezésére, és az egy hétre tervezett megfigyelések ütemezésének elkészítéséhez szükséges három hetet (!) körülbelül tíz percet rövidítette le.

A lokális keresés másik előnye az, hogy alkalmazható online elrendezésben is, amikor a probléma változik. Ez különösen fontos az ütemezési problémákban. Egy légitársaság heti ütemezése járatok ezreit és emberekhez rendelt feladatok tízezreit tartalmazhatja, de egy rossz időjárás az egyik repülőtéren lehetetlenné teheti az ütemezést. Ezt a lehető legkevesebb változtatással szeretnénk helyrehozni, amit a jelenlegi ütemezésből kiinduló lokális keresési algoritmussal végezhetünk el könnyen. Egy visszalépéses keresés a kényszerek egy új halmazzal általában jóval több időbe telik, és olyan megoldást is találhat, amely jelentősen különbözik az aktuálisról.



**5.9. ábra** Egy kétrépéses megoldás a 8-királynő problémára a MIN-KONFLIKTUSOK felhasználásával. Mindegyik fázisban egy királynőnek keresünk új oszlopot. A konfliktusok számát (esetünkben a támadó pozícióban lévő királynők számát) minden egyik négyzetben feltüntettük. Az algoritmus a királynőt a MIN-KONFLIKTUSOK négyzetre viszi, véletlenszerűen törve fel ezzel a kötésekkel.

## 5.4. A PROBLÉMÁK STRUKTÚRÁJA

Ebben az alfejezetben azt vizsgáljuk meg, miként lehet a probléma *struktúráját*, ahogy azt a kényszergráf megmutatja, felhasználni a megoldások gyors keresésére. Az itt tárgyalt megközelítések java része nagyon általános és a kényszerkielégítési problémákon kívül más eseteknél is alkalmazható, például a valószínűségi következtetésnél. Végül is a részproblémákká történő dekompozíció az egyetlen mód, mellyel remélhetjük, hogy megbirkózhatunk a valósvilág-beli problémákkal. Ismét ránézve az 5.1. (b) ábrára, miközben a probléma struktúráját keressük, egy tény ötlik a szemünkbe ki: Tasmania nincs összeköttetésben a nagy szárazfölddel.<sup>3</sup> Napnál világosabb, hogy Tasmania színezése és a szárazföld színezése független részproblémák (**independent subproblems**): a szárazföld színezésének bármely megoldása és Tasmania színezésének bármely megoldása kombinálva egyben a teljes térkép színezésének is megoldása lesz. A függetlenségről könnyű megbizonyosodni a kényszergráf **összefüggő komponenseit** (**connected components**) vizsgálva. Mindegyik komponens egy  $CSP_i$  részproblémának felel meg. Ha az  $S_i$  hozzárendelés egy megoldás a  $CSP_i$  részproblémára, akkor az  $\cup_i S_i$  hozzárendelés megoldása lesz az  $\cup_i CSP_i$ -nek. Miért fontos ez? Gondolunk bele a következőbe: tegyük fel, hogy minden egyes  $CSP_i$ -nek  $c$  változója van az  $n$ -ból, ahol  $c$  konstans. Ekkor  $n/c$  részproblémánk van, amelyek mindegyike legfeljebb  $d^c$  komplexitású. Tehát a teljes komplexitás  $O(d^c n/c)$ , ami lineáris  $n$ -ben, még dekompozíció nélkül  $O(d^n)$ , ami exponenciális  $n$ -ben. Tegyük ezt még konkrétabbá: egy  $n = 80$ -nal jellemzett Boole CSP-t négy részproblémává osztva ( $c = 20$ ) a megoldás futási idejét a legrosszabb esetben is az univerzum élethosszáról a másodperc törtrészére rövidítjük le.

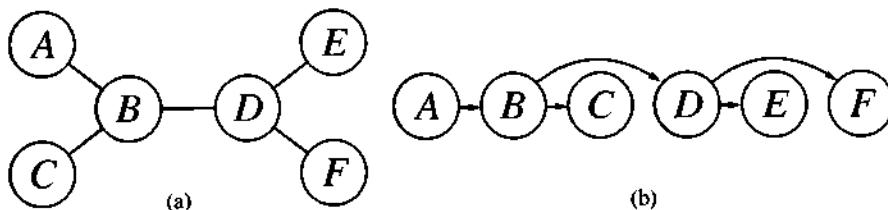
A teljesen független részproblémák ínyencfalaik, de ritkák. A legtöbb esetben a kényszerkielégítési problémák részproblémái kapcsolatban vannak egymással. A legegyszerűbb eset az, amikor a kényszergráf egy fát alkot: bármely két változót legfeljebb egy út köt össze. Az 5.10. (a) ábra egy sematikus példát mutat erre.<sup>4</sup> Meg fogjuk mutatni, hogy bármely *fastruktúrájú kényszerkielégítési probléma megoldható a változók száma szerinti lineáris időben*. Az algoritmusnak a következő lépései vannak:

1. Válasszuk ki bármelyik változót a fa gyökércomópontjául, és rendezzük a többi változót a gyökértől a levelekig úgy, hogy minden csomópontot a sorrendezésben megelőzzön a szülője (lásd 5.10. (b) ábra). Címkézzük ezeket a változókat sorban  $X_1, \dots, X_n$ -nel. Most, a gyökércomópontot leszámítva, minden változónak pontosan egy szülője van.
2. Alkalmazzuk az élkonziszenciát  $(X_i, X_j)$ -re, ahol  $X_i$  szülője  $X_j$ -nek ( $j$  pedig fusson visszafelé  $n$ -től 2-ig), és szükség esetén vegyük ki értékeit a  $TARTOMÁNY[X_i]$ -ből.
3. Adjunk  $X_j$ -nek bármilyen,  $X_i$  hozzárendelt értékkel konzisztenst értéket ahol  $X_i$  szülője  $X_j$ -nek, és  $j$  1-től  $n$ -ig halad.

<sup>3</sup> Egy nagyon gondos térképész vagy egy tasmaniai patriota ellenvéhetné, hogy Tasmaniát nem lehet ugyanolyan színnel színezni, mint a legközelebbi szárazföldi szomszédját, nehogy úgy tűnjön. mintha annak az államnak a része lenne.

<sup>4</sup> Sajnos nagyon kevés olyan területe van a világnak (talán Celebesz ilyen), amelynek faszerkezetű térképe van. (Celebesz egy közelítőleg csillagstruktúrájú sziget Közép-Indonéziában – A ford.)





**5.10. ábra.** (a) Egy faszerkezetű kényszerkielégítési probléma kényszergráfja. (b) Az A csomópont gyökérnek tekintésével konzisztens változók egy lineáris rendezése.

Két dolog érdemes említésre. Egyrészt a 2. lépés után a kényszerkielégítési probléma irány szerint élkonzisztens, tehát a 3. lépés hozzárendeléseiben nincsen szükség vissza-lépésre (lásd a  $k$ -konzisztencia tárgyalását a 193. oldalon). Másrészt, miután a 2. lépében fordított sorrendben alkalmaztuk az élkonzisztencia-ellenőrzéseket, elérült az algorit-mussal, hogy a törölt értékek ne veszélyeztessék a már feldolgozott élek konzisztenciáját. A teljes algoritmus  $O(nd^2)$  időben fut.

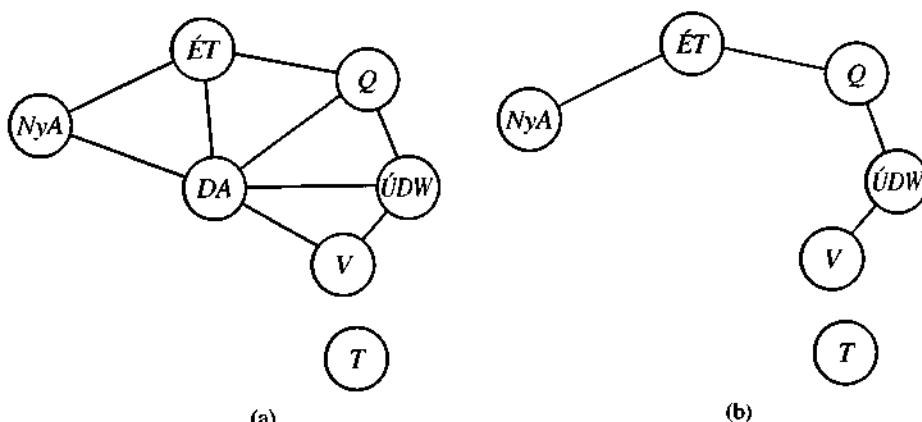
Most, hogy fákra már van egy hatékony algoritmusunk, megvizsgálhatjuk, miként lehet az általánosabb kényszergráfokat valahogyan fákra visszavezetni. Alapvetően két mód van erre: az egyik a csomópontok eltávolításán, a másik a csomópontok összevonásán alapul.

Az első megközelítés úgy jár el, hogy néhány változónak értéket ad, a maradékok pedig fát fognak alkotni. Vegyük ismét az 5.11. (a) ábrán látható kényszergráfot az Ausztrália-példához. Ha törölni tudnánk Dél-Ausztráliát, akkor a gráf fává válhatna (ahogy az ábra (b) részén látható). Szerencsére meg tudjuk tenni ezt (a gráfban, nem a kontinensen) azzal, hogy DA értékét rögzítjük, és a többi változó tartományából töröl-jük azokat az értéket, melyek inkonzisztensek a DA számára választott.

Most tehát hogy mind DA-t, mind a rá vonatkozó kényszereket eltávolítottuk, a kén-y-szerkielégítési probléma bármely megoldása konzisztens lesz a DA számára választott értékkel. (Ez bináris kényszerkielégítési problémák esetén működik; a helyzet jóval bonyolultabb magasabb rendű kényszerek esetén.) Tehát a keletkező fa a fenti algorit-mussal megoldható, és így az egész problémát is megoldottuk. Általános esetben persze (nem úgy, mint a térképszínezésnél) a DA számára választott érték lehet rossz is, és ekkor egyesével végig kell próbálgatni őket. Az általános algoritmus az alábbi:

1. Válasszunk ki egy  $S$  részhalmazt a  $\text{VÁLTOZÓK}[\text{csp}]$ -ből úgy, hogy a kényszergráf  $S$  eltávolítása után fa legyen.  $S$ -et **ciklikusság-vágóhalmaznak (cycle cutset)** nevezzük.
2.  $S$  minden egyes változójának minden egyes, az  $S$ -re vonatkozó összes kényszer-kielégítő lehetséges hozzárendelésére:
  - (a) vegyük ki a fennmaradó változók tartományaiból az  $S$  számára választott hozzá-rendeléssel inkonzisztens értékeket, és
  - (b) ha a fennmaradó kényszerkielégítési problémának van megoldása, akkor adjuk vissza ezt az  $S$  hozzárendelésével együtt.

Ha a ciklikusság-vágóhalmaz mérete  $c$ , akkor a teljes futási idő  $O(d^c \cdot (n - c)d^2)$  lesz. Ha a gráf „közel fa”, akkor  $c$  kicsi lesz, a megtakarítás pedig tetemes a gondolkodás nél-küli visszalépéses megoldáshoz képest. A legrosszabb esetben azonban  $c$  akár  $(n - 2)$  méretű is lehet. A *legkisebb* ciklikusság-vágóhalmaz megtalálása NP-nehéz probléma,



5.11. ábra. (a) Az 5.1. ábrán szereplő eredeti kényszergráf. (b) A kényszergráf DA eltávolítását követően.

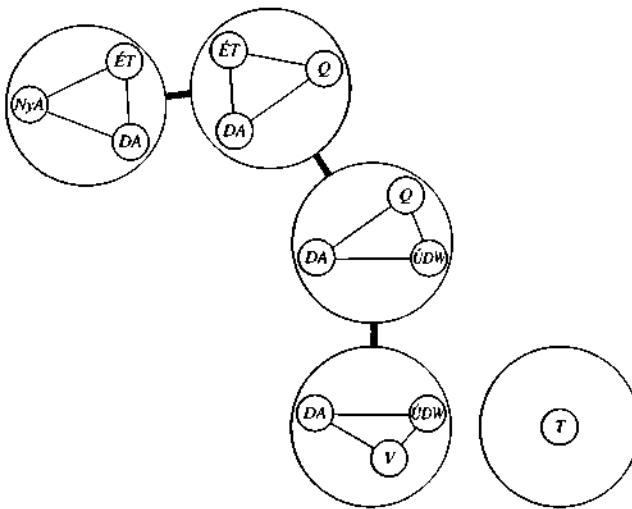
de sok hatékony algoritmust ismerünk erre a feladatra. Az általános algoritmikus megközelítés a vágóhalmaz-kondicionálás (cutset conditioning); még találkozni fogunk ezzel a 14. fejezetben, ahol a valószínűségekről történő következtetésekhez használjuk.

A második megközelítés a kényszergráf több, egymással kapcsolatban lévő részproblémából álló **fádekompozíció** (tree decomposition) előállításán alapul. Mindegyik részproblémát függetlenül oldjuk meg, és a keletkező megoldásokat összekapcsoljuk. A legtöbb „oszd meg és uralkodj” típusú megközelítéshez hasonlóan ez is akkor működik, ha egyik részprobléma sem túl nagy. Egy fádekompozíciónak az alábbi követelményeket kell kielégítenie:

- Az eredeti probléma mindegyik változója szerepeljen a részproblémák legalább egyikében.
- Ha bármely két változót az eredeti problémában egy kényszer köt össze, akkor a részproblémák legalább egyikében együtt is elő kell fordulniuk (és természetesen a kényszernek is).
- Ha egy változó a fa két részproblémájában is előfordul, akkor a változónak az ezeket a részproblémákat összekötő út minden részproblémájában is elő kell fordulnia.

Az első két kikötés biztosítja, hogy az összes változó és kényszer előforduljon a dekompozícióban. A harmadik kikötés eléggyé technikai ízűnek tűnik, de egyszerűen csak arról szól, hogy bármely adott változónak ugyanazzal az értékkel kell rendelkeznie az összes részproblémában, ahol előfordul – a részproblémákat a fában összekötő éllek kényszerítik ezt ki. Például DA az 5.12. ábra mindegyik összekötött részproblémájában előfordul. Az olvasó az 5.11. ábra alapján igazolhatja, hogy van értelme ennek a dekompozíciójának.

Külön-külön megoldhatjuk az egyes részproblémákat, és ha bármelyiknek is nincs megoldása, akkor az egész problémának nincs. Ha az összes részproblémát meg tudjuk oldani, akkor a következők szerint megpróbálhatunk összeállítani egy globális megoldást. Először is tekintsünk minden egyes változót egy „megaváltozónak”, amelynek tartománya a részprobléma összes megoldásának halmaza. Például az 5.12. ábra bal szélső részproblémája az a térképszínezési probléma, amelynek három változója és –



**5.12. ábra** Az 5.11. ábra (a) részén szereplő kényszergráf egy fa dekompozíciója

ezek szerint – hat megoldása van (ezek egyike az  $\{NyA = \text{vörös}, DA = \text{kék}, \bar{U}T = \text{zöld}\}$ ). Ezután a részproblémákat összekötő kényszereket meg tudjuk oldani a fákra adott fent bemutatott hatékony algoritmussal. A részproblémák közti kényszerek egyszerűen csak azért vannak, hogy a részproblémák megoldásai megegyezzenek az osztott változókban. Például, ha adott az  $\{NyA = \text{vörös}, DA = \text{kék}, \bar{U}T = \text{zöld}\}$  megoldás az első részproblémára, akkor a következő részprobléma egyetlen konzisztens megoldása a  $\{DA = \text{kék}, \bar{U}T = \text{zöld}, Q = \text{vörös}\}$  lehet.

Egy adott kényszergráfnak több dekompozíciója is lehetséges; a dekompozíció kiválasztásakor az a cél, hogy a részproblémák a lehető legkisebbek legyenek. Egy gráf fa-dekompozíciójának **faszélessége** (*tree width*) eggyel kisebb, mint a legnagyobb részprobléma mérete; magának a gráfnak a fazsélessége pedig definíció szerint a legkisebb fazsélesség az összes fadecompozíciója között. Ha egy gráfnak  $w$  a fazsélessége, és adott a megfelelő fadecompozíció, akkor a problémát meg lehet oldani  $O(nd^{w+1})$  időben. Tehát *egy felülről korlátos fazsélességű kényszergráfjal rendelkező kényszerkielégítési probléma polinomiális időben megoldható*. Sajnálatos módon egy minimális fazsélességű fadecompozíció megtalálása NP-nehéz probléma, de vannak olyan heurisztikus módszerek, amelyek jól működnek a gyakorlatban.

## 5.5. ÖSSZEFOGLALÁS

- A kényszerkielégítési problémák (CSP-k) változókból állnak, melyekre kényszerek vonatkoznak. Nagyon sok fontos valósvilág-beli probléma írható le kényszerkielégítési problémaként. A kényszerkielégítési problémák struktúrája egy kényszergráfjal reprezentálható.
- A **visszalépéses keresés** (*backtracking search*), a mélységi keresés egyik formája, a kényszerkielégítési problémák megoldásának gyakran alkalmazott eszköze.

- A legkevesebb fennmaradó érték (**least-remaining value**) heurisztika és a fokszám- (**degree-**) heurisztika tárgyterület-független módszerek annak eldöntésére, hogy a visszalépéses keresés során melyik változót válasszuk ki következőnek. A legkevésbé korlátozó érték (**least-constraining value**) heurisztika segítségül szolgálhat a változóértékek sorrendezésében.
- A visszalépéses algoritmus nagyban csökkenteni tudja a probléma elágazási tényezőjét a létrehozott részleges hozzárendelések következményeinek terjesztésével. Erre a legegyszerűbb módszer az előrenéző ellenőrzés (**forward checking**). Az élkonzisztencia (**arc consistency**) kikényszerítés egy jóval nagyobb teljesítőképeségű technika, de tovább is tart a futása.
- Visszalépésre akkor kerül sor, amikor egy változóhoz már nem találunk hozzárendelhető értéket. A konfliktusvezérelt visszaugrás (**conflict-directed backjumping**) közvetlenül a probléma okához ugrik vissza. A min-konfliktusok (**min-conflicts**) heurisztikát használó lokális keresést komoly sikkerrel alkalmazták a kényszerkielégítési problémákra.
- A kényszerkielégítési probléma komplexitása szorosan kötődik a saját kényszérgráfjának struktúrájához. A fastruktúrájú problémák megoldhatók lineáris időben. A vágóhalmaz-kondicionálás (**cutset conditioning**) az eredeti kényszerkielégítési problémát fastruktúrájúvá alakíthatja, és nagyon hatékonynak bizonyul, ha sikerül kis vágóhalmazt találnunk. A fádekompozíció (**tree decomposition**) technikák a kényszerkielégítési problémát részproblémák fájává alakíthatják, és hatékonyak, ha a kényszergráf faszélessége (**tree width**) kicsi.

## Irodalmi és történeti megjegyzések

A kényszerkielégítéssel kapcsolatos legkorábbi munka a numerikus kényszerekhez kötődik. Az egész értékű egyenlőségi kényszereket Brahmagupta indiai matematikus tanulmányozta a 7. században. Ezeket gyakran **diosfantoszi egyenleteknek** is nevezik Diofantosz görög matematikus (kb. 200–284) nyomán, aki valójában a pozitív racionális számok esetét vizsgálta. A lineáris egyenlőségek változóküküszöböléssel történő megoldásának szisztematikus módszereit Gauss tanulmányozta (Gauss, 1829), a lineáris egyenlőtlenség alakú kényszerek megoldása Fourier-ig vezethető vissza (Fourier, 1827).

A véges tartományú kényszerkielégítési problémáknak hosszú történetük van. A gráf-színezés például (amelynek a térképszínezés csak egy speciális esete) a matematika régi problémáinak egyike. Biggs és társai (Biggs és társai. 1986) szerint a négyszín-sejtést ( minden síkbeli gráf kiszínezhető legfeljebb négy színnel) először Francis Guthrie, de Morgan egyik tanítványa fogalmazta meg 1852-ben. A feladat ellenállt a megoldási kísérleteknek – annak ellenére, hogy néhányan publikációkban az ellenkezőjét állították –, míg nem Appel és Haken (Appel és Haken, 1977) előállt egy számítógépre is támászkodó bizonyítással.

A kényszerkielégítési problémák egyes osztályai gyakran felmerültek a számítógép-tudomány történetében. Az egyik legkorábbi nagy hatású példa a SKETCHPAD rendszer volt (Sutherland, 1963), amely geometriai kényszereket oldott meg diagramokban, és a modern rajzolóprogramok és CAD-programok előfutárának tekinthető. A kényszerkielégítési problémák általános problémaosztályként történő azonosítása Ugo Montanari (Montanari,

1974) nevéhez fűződik. A magasabb rendű kényszerkielégítési problémák visszavezetése segédváltozók felvételével tisztán bináris esetre (lásd 5.11. feladat) eredetileg a 19. századi logikushoz, Charles Sanders Peirce-hez fűződik. A CSP-irodalomba Dechter (Dechter, 1990b) vezette be, majd Bacchus és Van Beek (Bacchus és van Beek, 1998) dolgozták ki. A megoldásokra vonatkozó preferenciákkal kiegészített kényszerkielégítési problémákat széles körben tanulmányozza az optimalizáció irodalma; lásd (Bistarelli és társai, 1997)-et a CSP-kéretrendszer preferenciákat is megengedő általánosításáról. A vődör-eliminációs algoritmus (Dechter, 1999) szintén alkalmazható az optimalizációs problémáakra.

A kényszerkielégítési problémák visszalépéses keresése Bitnertől és Reingoldtől (Bitner és Reingold, 1975) származik, noha ők az alapalgoritmust a 19. századig követték vissza. Bitner és Reingold az MRV-heurisztikát is bevezették (ők ezt *leginkább-korlátozott-érték* heurisztikának nevezték). Az MRV-heurisztika utáni eldöntetlen helyzetek megoldására Brelaz (Brelaz, 1979) a fokszám-heurisztikát alkalmazta. Az így létrejövő algoritmus minden egyszerűsége ellenére, máig a leghatékonyabb tetszőleges gráfok  $k$ -színezésére. A *legkevésbé-korlátozó-érték* heurisztikát Haralick és Elliot javasolták (Haralick és Elliot, 1980).

A kényszerterjesztési módszereket Waltz (Waltz, 1975) sikere tette népszerűvé, amelyet a számítógépes látásnál felmerülő poliéder-élcímkézési problémán ért el. Waltz megmutatta, hogy sok probléma esetén a kényszerterjesztés teljesen kiküszöböli a visszalépést. Montanari (Montanari, 1974) bevezette a kényszerhálózat és az útvonalkonzisztenzia-terjesztés fogalmát. Alan Mackworth (Mackworth, 1977) javasolta az AC-3 algoritmust az élkonzisztenzia betartására, csakúgy, mint annak általános lehetőségét, hogy valamilyen fokú konzisztencia-ellenőrzést építsünk be a visszalépéses algoritmusba. Az AC-4, egy jóval hatékonyabb algoritmus, melyet Mohr és Henderson fejlesztettek ki (Mohr és Henderson, 1984). Nem sokkal Mackworth cikénék megjelenése után a kutatók elkezdték kísérletezni a konzisztencia-ellenőrzések költsége és a kereséslevágásban jelentkező előny csereviszonyának feltérképezésével. Haralick és Elliot (Haralick és Elliot, 1980) a McGregor által leírt (McGregor, 1979) minimális előrenéző ellenőrzés mellett álltak ki, míg Gaschnig (Gaschnig, 1979) minden egyes változó-hozzárendelés után élkonzisztenzia-ellenőrzéseket javasolt (ezt az algoritmust nevezte később Sabin és Freuder (Sabin és Freuder, 1994) MAC-nak). Az utóbbi cikk valamennyire meggyőző bizonyítékok hoz fel amellett, hogy nehezebb problémáknál kifizetődik a teljes élkonzisztencia-ellenőrzés. Freuder (Freuder 1978, 1982) megvizsgálta a  $k$ -konzisztenciát és ennek kapcsolatát a kényszerkielégítési problémák megoldásának komplexitásával. Apt (Apt, 1999) egy általános algoritmikus keretrendszer mutat be a konzisztenciaterjesztési algoritmusok vizsgálatára.

A magasabb rendű kényszerek kezelésének külön módszerei elsősorban a **kényszerlogikai-programozás (constraint logic programming)** keretein belül alakultak ki. Marriott és Stuckey (Marriott és Stuckey, 1998) nagyszerű összefoglalást nyújt erről a kutatási területről. A *MindKit* kényszert Regin (Regin, 1994) tanulmányozta. Az alsó és felső határokból álló kényszereket Van Hentenryck és társai vezették be a kényszerlogikai programozásba (Van Hentenryck és társai, 1998).

Az alapvető visszaugró módszer John Gaschnigtől (Gaschnig, 1977; 1979) származik. Kondrak és van Beek (Kondrak és Van Beek, 1997) megmutatták, hogy ezt az algoritmust lényegében magában foglalja az előrenéző ellenőrzés. A konfliktusvezérelt visszaugrást Prosser (Prosser, 1993) alakította ki. Az intelligens visszalépés legáltalánosabb és legerősebb formáját tulajdonképpen már nagyon korán kifejlesztette Stallman és Sussman

(Stallman és Sussman, 1977). Technikájuk, a függőségevezérelt visszalépés (**dependency-directed backtracking**) az igazság-karbantartó rendszerek (**truth maintenance systems**) kifejlesztéséhez vezetett (Doyle, 1979), amelyekkel a 10.8. alfejezetben foglalkozunk. A két terület közti kapcsolatot De Kleer (De Kleer, 1989) vizsgálta.

Stallman és Sussman munkája a kényszerfeljegyzés (**constraint recording**) elkezelését is bevezette: a keresés által elérő részleges eredményeket elmentjük és a keresés során később felhasználjuk. Ezt az elkezpelést a visszalépéses keresésbe formális módon Dechter (Dechter, 1990a) vezette be. A visszajegyzés (**backmarking**) (Gaschnig, 1979) egy különlegesen egyszerű módszer, amelyben a konzisztens és inkonzisztens páronkénti hozzárendeléseket elmentjük, hogy elkerüljük a kényszerek későbbi újraellenőrzését. A visszajegyzés ötvözhető a konfliktusvezérelt visszaugrással; Kondrak és Van Beek (Kondrak és Van Beek, 1997) bemutatnak egy hibrid algoritmust, amely bizonyíthatóan tartalmazza minden külön módszert. A dinamikus visszalépés (**dynamic backtracking**) (Ginsberg, 1993) megőrzi a változók későbbi részhalmazaiból származó sikeres parciális hozzárendeléseket, amikor egy olyan korábbi választási pontra ugrik vissza, amely a későbbi sikert nem teszi érvénytelenné.

A kényszerkielégítési problémák lokális keresését Kirkpatrick és társai (Kirkpatrick és társai, 1983) szimulált lehűtésről (**simulated annealing**) (lásd 4. fejezet) szóló munkája tette népszerűvé, és ezt széles körben alkalmazták az ütemezési problémáknál. A *min-konfliktusok* heurisztikát először Gu (Gu, 1989) javasolta, és tőle függetlenül Minton és társai (Minton és társai, 1992) is kifejlesztették. Sosic és Gu (Sosic és Gu, 1994) megmutatta, hogyan lehet ennek a heurisztikának az alkalmazásával a 3 000 000-királynő problémát kevesebb, mint egy perc alatt megoldani. A bámulatos siker, amit a min-konfliktusokat használó lokális keresés ért el az *n*-királynő problémában, a „könnű” és a „nehéz” problémák természetének és elterjedtségének újraértékeléséhez vezetett. Peter Cheesman és társai (Cheesman és társai, 1991) feltérképeztek a véletlenszerűen generált kényszerkielégítési problémák nehézségét, és azt találták, hogy majdnem minden ilyen probléma vagy triviálisan könnyű, vagy megoldhatatlan. Csak akkor találunk „nehéz” probléma példányokat, ha a problémagenerátor paramétereit egy bizonyos szűk tartományba állítjuk be, melyen belül a problémák közelítőleg fele megoldható. Ezzel a jelenséggel a 7. fejezetben foglalkozunk részletesebben.

A kényszerkielégítési problémák struktúrájának és nehézségének kapcsolatával foglalkozó kutatást Freuder (Freuder, 1985) indította el, aki megmutatta, hogy az élkonzisztens fák esetében a keresés visszalépések nélkül fut le. Egy hasonló eredmény az aciklikus hipergráfokra való kiterjesztésével együttermékként létre az adatbázisokkal foglalkozó kutatói közösségen (Beeri és társai, 1983). Ezen cikkek publikálása óta komoly haladás történt a kényszergráf struktúrája és a kényszerkielégítési probléma megoldási komplexitásának kapcsolatát illetően. A faszélesség fogalmát a gráfelmélettel foglalkozó Robertson és Seymour (Robertson és Seymour, 1986) vezették be. Freuder munkáságára építve Dechter és Pearl (Dechter és Pearl, 1987, 1989) ugyanezt a fogalmat – amit ők **indukált szélességnak (induced width)** hívtak – alkalmazták a kényszerkielégítési problémákra és kifejlesztették az 5.4. alfejezetben felvázolt fadecompozíciót. Az adatbázis-elméletre és erre az eredményre alapozva Gottlob és társai (Gottlob és társai, 1999a, 1999b) kialakították a kényszerkielégítési probléma hipergráfként történő felfogásán alapuló **hiperfaszélesség (hypertree width)** fogalmat. Annak megmutatásán túl, hogy minden  $w$  szélességű hiperfa-CSP megoldható  $O(n^{w+1} \log n)$  időben, azt is bebizonyították, hogy minden  $w$  szélességű hiperfa-CSP megoldható  $O(n^{w+1} \log n)$  időben, ahol  $n$  a keresendő variáns száma,  $w$  pedig a hipergráf szélessége.

nyították, hogy a hiperfaszélesség az összes korábbi „szélesség”-mértéket magában foglalja (abban az értelemben, hogy bizonyos esetekben a hiperfaszélesség korlátozott, bizonyos esetekben pedig nem).

Sok jó áttekintés létezik a CSP-technikákhoz, például Kumar (Kumar, 1992), Dechter és Frost (Dechter és Frost, 1999) és Bartak (Bartak, 2001); továbbá a Dechter-től (Dechter, 1992) és Mackworthtől (Mackworth, 1992) származó enciklopédiaikkek. Pearson és Jeavons (Pearson és Jeavons, 1997) a könnyen kezelhető CSP-osztályokat veszik számba, beleérve minden a strukturális dekompozíciós módszereket, minden a maguknak a tartományoknak vagy a kényszereknek a tulajdonságaira támaszkodó módszereket. Kondrak és Van Beek (Kondrak és Van Beek, 1997) a visszalépéses keresési algoritmusok analitikus áttekintését adják, Bacchus és Van Run (Bacchus és Van Run, 1998) pedig empirikusabb képet festenek. Tsang (Tsang, 1993), valamint Marriott és Stuckey (Marriott és Stuckey, 1998) szövegei jóval mélyebbre hatolnak a témaban, mint azt a fejezet korlátai számunkra lehetővé tették. Sok érdekes alkalmazást mutat be a Freuder és Mackworth szerkesztésében megjelent gyűjtemény (Freuder és Mackworth, 1994). Kényszerkielégítéssel foglalkozó cikkek rendszeresen jelennek meg az *Artificial Intelligence*-ben és egy specialistáknak szóló újságban, a *Constraints*ben. A legfőbb konferenciakiadvány az International Conference on Principles and Practice of Constraint Programming, amit gyakran CP-nek hívnak.

## Feladatok

- 5.1.** Fogalmazza meg a saját szavaival a kényszerkielégítési problémák, a kényszerek, a visszalépéses keresés, az élkonzisztencia, a visszaugrás és a min-konfliktusok definícióját.
- 5.2.** Hány megoldása van az 5.1. ábra térképszínezési problémájának?
- 5.3.** Magyarázza el, miért jó heurisztika a *leginkább* korlátozott változót és a *legkevésbé* korlátozott értéket választani a kényszerkielégítési probléma megoldásának keresése közben.
- 5.4.** Tekintsük a keresztrejtvények készítésének (nem megoldásának) problémáját:<sup>5</sup> szavakat kell illeszteni egy négyzetes rácoba. A rácst, amely része a probléma specifikációjának, megadja, hogy mely négyzetek legyenek üresek és melyek sötétek. Tegyük fel, hogy adott a szavak egy listája (például egy szótár), és az a feladat, hogy az üres négyzeteket a lista tetszőleges részhalmazát használva kitöltsük. Fogalmazza meg pontosan ezt a problémát kétféle módon:
  - (a) Mint általános keresési problémát. Válasszon egy megfelelő keresési algoritmust, és specifikálja a heurisztikus függvényt (amennyiben elképzelése szerint szüksége van rá). A fehér kockákba egyszerre egy betűt vagy egész szavakat érdemes-e beírni?

<sup>5</sup> Ginsberg és társai (Ginsberg és társai, 1990) több módszert tárgyalnak keresztrejtvények készítésére. Littman és társai pedig a nehezebb problémát, megoldásukat veszik célba (Littman és társai, 1999).

- (b) Mint kényszerkielégítési problémát. A változók betűk vagy szavak legyenek? Melyik megfogalmazást tartja jobbnak? Miért?
- 5.5.** Adjon precíz megfogalmazást az alábbiakra mint kényszerkielégítési problémákra:
- Négyszögletes kirakó: találjon nemátfedő helyeket egy nagy négyzetben kisebb négyzögek számára.
  - Órarend-ütemezés: adott számú professzor és terem van, valamint rögzített az órarendi órák listája is a lehetséges időablakkal együtt. Mindegyik professzorhoz adott az általa tartott órák halmaza.
- 5.6.** Oldja meg az 5.2. ábra betűrejtvényét kézzel, visszalépéses kereséssel, előrenéző ellenőrzéssel, valamint az MRV-, illetve a legkevésbé korlátozó érték heurisztikával.
- 5.7.** Az 5.5. ábra a különböző algoritmusokat az  $n$ -királynő problémán teszeli. Próbálja meg ugyanezt egy véletlenszerűen generált térképszínezési problémával is: osszon el az egységsíkon véletlenszerűen  $n$  pontot, válasszon ki véletlenszerűen egy  $X$  pontot, kösse  $X$ -et a legközelebbi olyan  $Y$  ponthoz, amelyikkel  $X$  még nincs összekötve, és a vonal semelyik más vonalat nem metsz; ismételje a fenti lépést mindaddig, amíg újabb összeköttetés már nem lehetséges. Számítsa ki a teljesítménytáblázatot a legnagyobb  $n$ -re, amire csak tudja (mind  $d = 3$ -at mind,  $d = 4$ -et használva). Fűzzön magyarázatokat a kapott eredményhez.
- 5.8.** Az AC-3 algoritmus felhasználásával mutassa meg, hogy az élkonisztenzia alkalmas arra, hogy kimutassa az 5.1. ábra  $\{NyA = \text{vörös}, V = \text{kék}\}$  parciális hozzárendelésének inkoniszenciáját.
- 5.9.** Mi a fastruktúrájú kényszerkielégítési problémán futtatott AC-3 legrosszabb esetbeli komplexitása?
- 5.10.** Az AC-3 visszarak a sorba minden  $(X_k, X_i)$  élet, amikor  $X_i$  tartományából bármely értéket töröltek, akkor is, ha  $X_k$  minden értéke konzisztens  $X_i$  több fennmaradó értékével. Tegyük fel, hogy minden egyes  $(X_k, X_i)$  élhez nyilvántartjuk a fennmaradó  $X_i$  értékek számát, amelyek az  $X_k$  minden egyes értékével konzisztensek. Magyarázza el, hogyan lehet hatékonyan frissíteni ezeket az értékeket, és hogyan lehet ennek segítségével az élkonisztenciát  $O(n^2 d^2)$  lépésben elérni.
- 5.11.** Mutassa meg, hogy egy ternáris kényszer, mint például az „ $A + B = C$ ” egy segédváltozó bevezetésével három bináris kényszerré alakítható. Feltételezheti, hogy a tartományok végesek. (Segítség: gondoljon egy olyan új változóra, amelynek értékei más értékekből álló párok, és gondoljon olyan kényszerekre, mint „ $X$  az első eleme az  $Y$  párnak”.) Ezután mutassa meg, hogyan lehet hasonlóan kezelni a háromnál több változót tartalmazó kényszereket. Végül mutassa meg, miként lehet kiküszöbölni az unáris kényszereket a változók tartományának megváltoztatásával. Ez teljessé teszi annak bizonyítását, hogy bármely CSP átalakítható olyan problémákká, melyek csak bináris kényszereket tartalmazhatnak.

- 5.12.** Tegyük fel, hogy ismerjük egy gráfról, hogy van egy legfeljebb  $k$  csomópontot tartalmazó ciklikusság-vágóhalmaza. Írjon le  $n$  változós CSP-k esetén egy egyszerű algoritmust a minimális ciklikusság-vágóhalmaz megkeresésére, ahol a futási idő maximuma  $O(n^k)$ . Végezzen irodalomkutatást olyan vágóhalmaz-keresési eljárások után, amelyek a vágóhalmaz méretében közelítőleg polinomiális időben találnak közelítőleg minimális ciklikusság-vágóhalmazt. Praktikussá teszi az ilyen módszerek létezése a ciklikusság-vágóhalmaz módszereket?
- 5.13.** Tekintsük a következő logikai rejtvényt: Öt különböző színű házban öt különböző nemzetiségi személy él, és mindenkiük más márkjáú cigarettát, más italt és más háziállatot szeret. Az alábbi tények alapján a megválaszolandó kérdés a következő: „Hol lakik a zebra, és melyik házban isznak vizet?”
- Az angol a vörös házban lakik.
  - A spanyolnak kutyája van.
  - A norvég balról az első házban lakik.
  - A Kools cigarettát a sárga házban szívják.
  - A Chesterfieldset szívó ember a rókás ház mellett lakik.
  - A norvég a kék ház mellett lakik.
  - A Winstont szívó ember kígyókat tart.
  - A Lucky Strike-ot szívó narancslevet iszik.
  - A ukrán teát iszik.
  - A japán Parliamentset szív.
  - A Koolsot abban a házban szívják, amely mellett lovat tartanak.
  - Kávéét a zöld házban isznak.
  - A zöld ház közvetlenül jobbra (Ön felől nézve) van az elefántcsontszínű háztól.
  - Tejet a középső házban isznak.
- Vizsgálja meg a probléma különböző CSP-reprezentációit. Milyen okokból részesíténel előnyben az egyiket a másikkal szemben?

# 6. KERESÉS ELLENSÉGES KÖRNYEZETBEN

*Ebben a fejezetben azokat a problémákat vizsgáljuk meg, amelyek akkor merülnek fel, amikor egy ágens előre tervez ellenséges ágenseket tartalmazó világban.*

## 6.1. KÉTSZEMÉLYES JÁTÉKOK<sup>1</sup>

A 2. fejezetben többágenses környezeteket (**multiagent environments**) vezettünk be, ahol minden ágensnek számolnia kell más ágensek cselekvéseivel és azzal is, hogy azok hogyan befolyásolják a jólétét. Más ágensek nem megijosolható viselkedése számos lehetséget **eshetőséget** (**contingencies**) visz be az ágens problémamegoldásába, ahogy ezzel a 3. fejezetben foglalkoztunk. A 2. fejezetben bevezettük a több ágensből álló **kooperatív** (**cooperative**) és **verseny-** (**competitive**) környezeteket is. A verseny-környezetek, ahol az ágensek céljai konfliktusban vannak, elvezetnek az **elleniségek melletti kereséshez** (**adversarial search**) – amit sokszor **kétszemélyes** játékoknak (**games**) nevezünk.

A matematikai játékelmélet (**game theory**) a gazdaságtan egyik ága, mely a többágenses környezeteket játéknak tekinti, feltéve, hogy egy-egy ágens hatása másokra „szignifikáns”, függetlenül attól, hogy az ágensek kooperatívak vagy versengők.<sup>2</sup> Az MI-ben a „játékok” általában igen specializáltak – amit a játékelméleti szakemberek determinisztikus, váltott lépésű, kétszemélyes, zérusösszegű teljes információjú játékoknak (**zero-sum games of perfect information**) neveznek. A mi nyelvezetünkben ez azt jelenti, hogy két ágens helyezkedik el egy determinisztikus és teljesen megfigyelhető környezetben, a cselekvések váljtják egymást, és a játék végén a hasznosságértékeik minden azonosak és ellentétes előjelűk. A sakkból például, ha az egyik játékos győz (+1), akkor a másik szükségszerűen veszít (-1). Éppen a hasznosságértékekben tapasztalt ellentéttől lesz a helyzet ellenséges. Ebben a fejezetben röviden foglalkozunk a többjátékos játékokkal, a nem zérusösszegű játékokkal és a sztochasztikus játékokkal, de a tényleges játékelmélettel csak a 17. fejezetben fogunk foglalkozni.

A játékok a civilizáció kezdete óta – néha már ijesztő mértékben – foglalkoztatják az emberek intellektuális képességeit. A játékok absztrakt természetük miatt vonzó területet jelentenek az MI-kutatók számára. Egy játék állását könnyű reprezentálni, és az ágensek képességei általában kisszámú, jól definiált eredményre vezető cselekvésre kor-

<sup>1</sup> Annak ellenére, hogy az angol „game” szó magyar szó szerinti fordítása „játék”, úgy éreztük, hogy az ilyen fordítás magyarul összemossa a „game” és a „toy” közötti különbséget. A „kétszemélyes játékok” szóhasználat alkalmas kompromisszum, annak ellenére, hogy az ebben a fejezetben tárgyalott módszerek olyan „kétszemélyes játékokra” is kiterjeszhetők, ahol több játékos van. (A ford.)

<sup>2</sup> A nagyon sok ágensből álló környezeteket jobb **gazdaságoknak** (**economies**) és nem játékoknak nézni.

látózódnak. Az olyan fizikai játékoknak, mint a krikett vagy a jégkorong, sokkal bonyolultabb a leírása. Sokkal több lehetséges cselekvéssel rendelkeznek, és a cselekvések legalis voltát éppenséggel nem túl precíz szabályok határozzák meg. A robotfutballt kivéve e fizikai játékok az MI-közösségen sok érdeklődést nem keltettek.

A kétszemélyes játékok az egyik legrégebbi, az MI által vizsgált területet jelentik. 1950-ben, alighogy a számítógépek programozhatóvá váltak, Konrad Zuse (az első programozható számítógép és az első programozási nyelv megalkotója), Claude Shannon (az információelmélet atya), Norbert Wiener (a korszerű szabályozáselmélet megteremtője) és Alan Turing elkezdtek sakkprogramokkal foglalkozni. Azóta a játékok színvonala sokat fejlődött, addig a szintig, hogy a gépek túlszármányolták az embert dámajátékban és Othelloban, megverték (bár nem minden alkalommal) ostáblában és sakkból az emberi bajnokokat, és versenyképesek más játékokban is. Kivétel a gó, ahol a számítógép csak amatőr szinten játszik.

A 3. fejezetben tanulmányozott játékproblémákkal ellentétben a kétszemélyes játékok azért érdekesek, mert nagyon nehéz őket megoldani. A sakknál például az átlagos elágazási tényező 35, és egy játék során gyakran előfordul, hogy minden fél 50–50 lépést is megtesz, vagyis a keresési fának  $35^{100}$ , illetve  $10^{154}$  csomópontja van (bár a keresési gráfnak ebből „csak”  $10^{40}$  különböző csomópontja lesz). A játékok, éppúgy, mint a valós világ, azt a képességet igénylik, hogy *valamilyen* döntést hozzunk, akkor is, ha az *optimális* döntés kiszámlítása kivitelezhetetlen. A játékok nagyon komolyan büntetik a rossz hatékonyságot. Míg az A\* algoritmus azon implementációja, ami csak fele olyan hatékony, egyszerűen csak kétszer annyi ideig fut, hogy megkapja az eredményt, addig egy olyan sakkprogramot, amely feleolyan hatékonyan gazdálkodik az idejével, feltéve, hogy minden más szempontból egy másik implementációval azonos, valószínűleg a földbe döngörnének. A járékelmeleti kutatás ezért számos érdekes ötlethez vezetett, hogy a rendelkezésre álló időt hogyan használjuk a lehető legjobb módon.

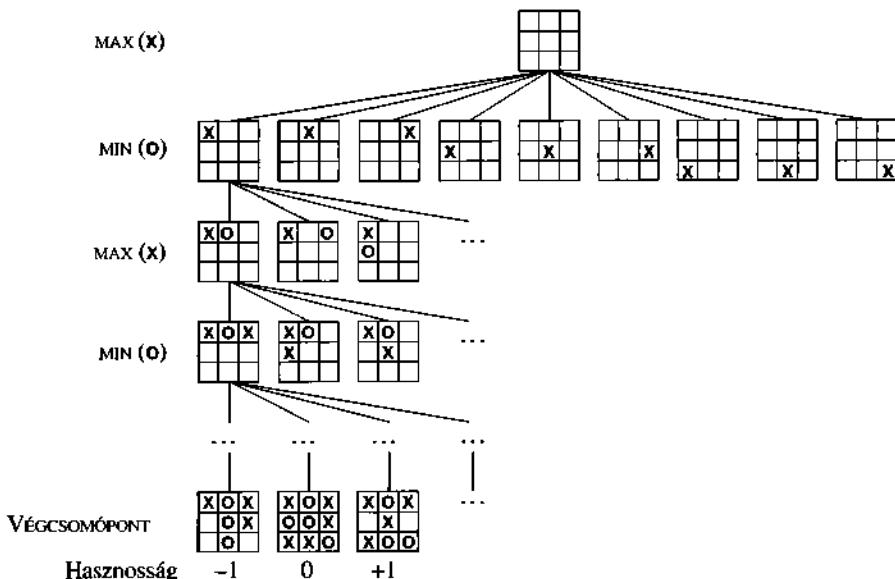
A probléma tárgyalását az elméletileg lehetséges legjobb lépés definíciójával és ennek egy keresőalgoritmusával kezdjük. Ezek után olyan technikákat nézünk meg, amelyek korlátozott idő alatt is alkalmasak egy jó lépés megválasztására. A *nyesés* vagy *metszés* (*pruning*) lehetővé teszi számunkra, hogy figyelmen kívül hagyjuk a keresési fa azon részeit, amelyek nincsenek befolyással a végső választásra. A heurisztikus *kiértékelő függvények* (*evaluation functions*) lehetővé teszik, hogy kimerítő keresés nélkül meg tudjuk becsülni egy adott állapot valódi hasznosságát. A 6.5. alfejezet olyan kétszemélyes játékokat tárgyal, amelyekben a véletlen is megjelenik, mint például az ostábla.<sup>3</sup> Foglalkozunk a briddzel is, amely tartalmazza a **hiányos információ** (*imperfect information*) elemeit, hiszen egy-egy játékos számára az összes kártya nem ismert. Megnézzük végül, hogy a jelenlegi legfejlettebb játékprogramok hogyan győzik le az erős emberi ellenfeleket, és milyenek a jövőbeli trendek.

<sup>3</sup> Az ostábla (backgammon) Magyarországon nem annyira elterjedt, bár számos vidéken ismert. A lényege, hogy egy speciális táblán, felváltva lépve, a bábkat az ellenfél térfelére juttassuk. Fő érdekkessége, ami a nehezségek forrása is egyben, hogy a legális lépések mindenkor választékába véletlenszerűen – egy kockadobásnak megfelelően – alakul. (A ford.)

## 6.2. OPTIMÁLIS DÖNTÉSEK KÉTSZEMÉLYES JÁTÉKOKBAN

A kétszemélyes játékokkal fogunk foglalkozni, ahol a két játékos – a hamarosan nyilvánvalóvá váltó okból – MAX-nak és MIN-nek fogjuk hívni. MAX lép először, majd a játékosok felváltva lépnek, amíg a játék véget nem ér. A játék végén a győztes játékos pontot kap (vagy néha a vesztes kap büntetőpontokat). A játékot formálisan egyfajta keresési problémaként lehet definiálni az alábbi komponensekkel:

- A **kiinduló állapot** (*initial state*), ami magában foglalja a táblaállást, valamint azt, hogy ki fog lépni.
- Egy **állapotátmenet-függvény** (*successor function*), amely (*lépés, állapot*) párok listájával tér vissza, megadva a legális lépéseket és az azokból következő állapotokat.
- Egy **végeszt** (*terminal test*), ami meghatározza, hogy a játéknak mikor van vége. Azok az állapotok, ahol a játék befejeződött, a **végállapotok** (*terminal states*).
- Egy **hasznosságfüggvény** (*utility function*, amit **nyereségfüggvénynek** – *payoff function* – is neveznek) a játék végeredményéhez egy számértéket rendel. A sakkból a végeredmény győzelem, vereség vagy döntetlen lehet, amit a +1, -1 és 0 értékekkel ábrázolhatunk. Néhány játék ennél több végeredményre vezethet. Például az osztáblában a nyereség +192 és -192 között változhat. Ebben a fejezetben főleg a zérusösszegű játékokkal foglalkozunk, bár a nem zérusösszegű játékokat is megemlítjük.



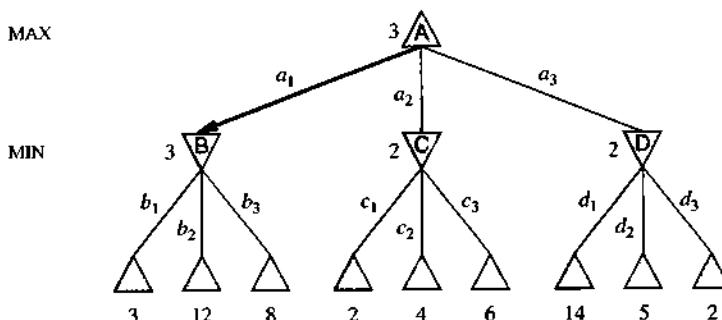
**6.1. ábra.** A  $3 \times 3$ -as amőbajáték (részleges) keresési faja. A legfelső csomópont a kiinduló állapot. MAX lép először, egy X-et téve valamelyik üres négyzetbe. A keresési fa egy részét mutatjuk, MIN (O) és MAX váltakozó, egymást követő lépései megadva, amíg el nem érjük a végállapotokat, melyekhez a játék szabályai szerint lehet hasznossági értékeket hozzárendelni.

A kezdeti állapot és minden fél legális lépései a játék játékfáját (**game tree**) definiálják. A 6.1. ábra a  $3 \times 3$ -as amőbajáték keresési fájának egy részét mutatja. Kezdeti állapoiban MAX-nak kilenc lehetséges lépése van. A játék során MAX és MIN felváltva tesznek x-et illetve o-t, míg egy végállapotnak megfelelő levélcsomópontba el nem jutnak, ahol az egyik játékosnak egy sorban, egy oszlopban vagy az egy átló mentén három o-ja vagy x-e lesz, vagy minden négyzet ki lesz töltve. Az egyes levélcsomópontok alatt található számok a végcsomópontnak a MAX szempontjából mért hasznosságát jelölik. A nagy értelekről feltételezzük, hogy jók MAX számára és rosszak MIN számára (ami-ból a játékosok neve is ered). MAX feladata, hogy a játékfát (és főleg a végállapotok hasznosságát) a legjobb lépés meghatározására használja fel.

## Optimális stratégiák

Egy normális keresési problémánál, az optimális megoldás nem lenne más, mint a célállapothoz vezető lépések szekvenciája – azaz egy olyan végállapothoz vezető lépés-szekvencia, amely a győzelmet jelenti. Egy játékban azonban MIN-nek is van beleszólása a dologba. Ezért MAX-nak egy olyan stratégiát (**strategy**) kell találnia, amely meghatározza MAX lépéset a kezdeti állapotban, majd a MIN lehetséges válaszaiból keletkező állapotokban, majd ismét MAX lépéseit a MIN erre vonatkozó lehetséges válaszaiból keletkező állapotokban és így tovább. Nagyából azt lehet mondani, hogy egy optimális stratégia olyan kimenetelekhez vezet, amelyek legalább olyan jók, mintha bármilyen más stratégiával egy tévedhetetlen opponens ellen játszanánk. Először megmutatjuk, hogyan kell megkeresni az optimális stratégiát, bár ennek kiszámítására MAXnak általában nem lesz elegendő ideje az amőbánál bonyolultabb játékokban.

Még egy olyan egyszerű játék, mint az amőba is túl bonyolult ahhoz, hogy megmutassuk a teljes keresési fát, ezért áttérünk a 6.2. ábrán látható abszolút triviális játékra. MAX lehetséges lépései  $a_1$ -gyel,  $a_2$ -vel és  $a_3$ -mal címkéztük meg. Az  $a_1$ -re MIN lehetséges válaszait a  $b_1$ ,  $b_2$ ,  $b_3$  stb. jelölik. Ez a konkrét játék MAX és MIN egy-egy lépése



**6.2. ábra.** Egy lépésváltásos játékfa. A  $\Delta$  csomópontok MAX lépései, míg a  $\nabla$  csomópontok MIN lépései jelölik. A végcsomópontok a hasznossági függvényrel számított hasznossági értéket MAX szemszögből mutatják, míg a többi csomópontnál a minimax értékét jelöltük be. MAX legjobb lépése a gyökérben  $a_1$ , mert ez vezet a legmagasabb minimax értékű követőhöz. MIN legjobb válasza  $b_1$ , mert ez vezet a minimális minimax értékű követőhöz.

után véget ér. (A játékok nyelvén azt mondjuk, hogy ez a fa egy lépés mély és két fél lépésből vagy lépésváltásból (ply) áll.) A végcsomópontok hasznossága ebben a játékból 2 és 14 közé esik.

Adott játékfa mellett az optimális stratégia meghatározásához az egyes csomópontok **minimax értékét** kell megvizsgálni, amit MINIMAX-ÉRTÉK( $n$ )-ként írunk le. Egy csomópont minimax értéke a csomópont hasznossága MAX szemszögéből, feltéve, hogy innen kezdve egészen a játék befejezéséig minden követő játékos optimálisan lép. Egy végállapot minimax értéke természetesen a saját hasznossága. Továbbá, adott minimax értékek mellett, MAX szeretne a maximális értékű, MIN pedig a minimális értékű állapotba jutni. Rendelkezünk tehát az alábbi függvényteljesítésekkel:

**MINIMAX-ÉRTÉK( $n$ ) =**

HASZNOSÍTÁS( $n$ )

ha  $n$  egy végállapot,

$\max_{s \in K_{\text{követők}}(n)} \text{MINIMAX-ÉRTÉK}(s)$

ha  $n$  egy MAX csomópont,

$\min_{s \in K_{\text{követők}}(n)} \text{MINIMAX-ÉRTÉK}(s)$

ha  $n$  egy MIN csomópont.

Alkalmazzuk ezeket a definíciókat a 6.2. ábrán látható játékfára. Az ábra alján lévő cél-állapotokat már felcímkeztük hasznossági értékükkel. Az első,  $B$  címkéjű, MIN csomópontnak három követője van 3, 12 és 8 értékkel, a  $B$  csomópont minimax értéke tehát 3. Hasonlóan a két másik MIN csomópontnak 2 a minimax értéke. A gyökér egy MAX csomópont; a követőinek minimax értékei 3, 2, és 2, így a gyökér minimax értéke 3. Azonosíthatjuk a gyökér **minimax döntését (minimax decision)** is. MAX számára az  $\sigma_1$  cselekvés az optimális választás, mert maximális értékű követőhöz vezet.

Ezen optimális játékdefiníció MAX számára feltételezi, hogy MIN is optimálisan játszik – hiszen a dolgok kimenetelét MAX számára a legrosszabb esetre kivételve maximálizálja. Mi van azonban, ha MIN nem játszik optimálisan? Ekkor könnyű belátni (6.2. feladat), hogy MAX még jobban jár. Szuboptimális ellenféllel szemben sok stratégia el-képzelhető, amely az optimálisnál jobb, azonban ezek a stratégiák rosszabbnak fognak bizonyult optimális ellenfelekkel szemben.

## A minimax algoritmus

A **minimax algoritmus (minimax algorithm)** (6.3. ábra) az optimális döntést az aktuális állapotból számítja ki, felhasználva az egyes követő állapotok minimax értékeit a kiszámítására a definíáló egyenletekből közvetlenül származtatott, egyszerű rekurzív formulát. A rekurzió egészen a falevelekig folytatódik, majd a minimax értékeket a fa mentén visszafelé terjesztjük (back-up), ahogy a rekurzió visszalép. A 6.2. ábrán például az algoritmus először rekurzív módon leereszkedik a három bal alsó csomóponthoz, a HASZNOSÍTÁS függvényteljesítésekkel kiszámítva, hogy az értékek rendre 3, 12, és 8. Majd az algoritmus előveszi ezen értékek minimumát, azaz 3-t, és ezt adjja vissza, ahogy a  $B$  csomóponthoz visszatér. Hasonló procedúra eredményezi a további visszaadott értékeket: 2-t a  $C$  és 2-t a  $D$  csomópont számára. Végül vesszük a 3, 2 és 2 értékek maximumát, hogy a gyökér által visszaadott 3-as értéket megkaphassuk.

A minimax algoritmus a játékfa teljes mélységi feltárást végezzi. Ha a fa maximális mélysége  $m$ , és minden csomópontban  $b$  legális lépés létezik, akkor a minimax algorit-

```
function MINIMAX-DÖNTÉS(állapot) returns egy cselekvés
```

**Inputs:** *állapot*, a játék aktuális állapota

$v \leftarrow \text{MAX-ÉRTÉK}(\text{állapot})$

**return** *v* értékű cselekvés KÖVETŐÁLLAPOTOK(*állapot*)-ban

```
function MAX-ÉRTÉK(állapot) returns egy hasznossági érték
```

**if** VÉG-TESZT(*állapot*) **then return** HASZNOSÍG(*állapot*)

$v \leftarrow -\infty$

**for minden**  $s \in \text{KÖVETŐÁLLAPOTOK}(\text{állapot})$  **do**

$v \leftarrow \text{MAX}(v, \text{MIN-ÉRTÉK}(s))$

**return** *v*

```
function MIN-ÉRTÉK(állapot) returns egy hasznossági érték
```

**if** VÉG-TESZT(*állapot*) **then return** HASZNOSÍG(*állapot*)

$v \leftarrow \infty$

**for minden**  $s \in \text{KÖVETŐÁLLAPOTOK}(\text{állapot})$  **do**

$v \leftarrow \text{MIN}(v, \text{MAX-ÉRTÉK}(s))$

**return** *v*

**6.3. ábra.** Egy algoritmus a minimax döntések kiszámítására. Az algoritmus a lehető legjobb lépéshoz tartozó operátort adja vissza, vagyis ahhoz a lépéshoz tartozó operátort, amelyik a legnagyobb hasznossági értékkel rendelkező eredményre vezet, feltételezve, hogy az ellenfél úgy játszik, hogy minimalizálja a hasznossági értéket. A MAX-ÉRTÉK és MIN-ÉRTÉK függvények végigmennek a teljes játéksán, le egészben a levélcsomópontokig, hogy meghatározzák a csomópont felfelé terjesztett értékét.

mus időkomplexitása  $O(b^m)$ . A tárkomplexitása  $O(bm)$  egy olyan algoritmus számára, amely az összes követőt egyszerre számítja ki, és  $O(m)$  egy olyan algoritmus esetében, amely a követőket egyenként generálja (lásd 115–116. oldal). Valós játékok esetén ez az időkomplexitás az algoritmust teljesen haszontalanítja, az algoritmus azonban jó alap a játékok matematikai elemzéséhez és a gyakorlati szempontból alkalmasabb algoritmusokhoz.

## Optimális döntések többszemélyes játékokban

Számos elterjedt játékban több játékos is részt vehet, nem csupán kettő. Vizsgáljuk meg, hogy a minimax ötlelet hogyan terjeszthetjük ki többszemélyes játékok esetére. Technikai szempontból a dolog egyszerű, azonban felmerül néhány érdekes koncepcionális kérdés.

Először is egy csomópontokhoz rendelt egyetlen értéket egy értékvektorral kell felváltani. Például egy háromszemélyes játékban, ahol három játékos,  $A$ ,  $B$  és  $C$  vesz részt, minden csomóponttal egy  $(v_A, v_B, v_C)$  vektort társítunk. Végállapotok esetén ez a vektor megadja az állapot hasznosságát minden játékos szemszögéből (kétszemélyes zérus-összegű játékokban a kételemű vektort egy értékre le lehet egyszerűíteni, mert az értékek mindenkorral ellentétesek). Ezt a kibővítést legjobb úgy implementálni, hogy a HASZNOSÍG függvény adja vissza a hasznosságok vektorát.

Most a nem terminális állapotokkal fogalkozunk. Nézzük meg a 6.4. ábrán látható játékfában az  $X$  jelzésű csomópontot. Ebben az állapotban a  $C$  játékos dönti el, hogy mit

Aki lép

*A*

*B*

*C*

*A*

(1, 2, 6)

(1, 5, 2)

(1, 2, 6)

(6, 1, 2)

(1, 5, 2)

(5, 4, 5)

(1, 2, 6)

(6, 1, 2)

(5, 1, 1)

(7, 7, 1)

(4, 2, 3)

(7, 4, 1)

(1, 5, 2)

(5, 4, 5)

(6, 1, 2)

(7, 4, 1)

(5, 1, 1)

(7, 7, 1)

(7, 4, 1)

(5, 1, 1)

(1, 5, 2)

(5, 4, 5)

(5, 4, 5)

(7, 7, 1)

(5, 4, 5)

(5, 4, 5)

6.4. ábra. Három játékos (*A*, *B*, *C*) játékfája a három első lépés esetén. minden csomópontot az összes játékos szemszegéből számított értékkel címkéztük meg. A legjobb lépést a gyökérnél jelöltük be.

csináljon. Egyik választása a  $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ , míg a másik a  $\langle v_A = 4, v_B = 2, v_C = 3 \rangle$  vektorokkal rendelkező végállapothoz vezet. Mivel 6 több, mint 3, *C*-nek az első lépést kellene választania. Ez azt jelenti, hogy ha a játék az *X* csomópontot eléri, a következő lépés a  $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$  hasznosságú végállapothoz fog vezetni. *X* visszaadott értéke így ez a vektor. Általánosságban egy *n* csomópont visszaadott értéke annak a követőnek a hasznosságvektora, amely követőnek az *n* csomópontról választó játékos szempontjából legnagyobb az értéke.

Mindenki, aki olyan többszemélyes játékokat játszik, mint például a Diplomacy™, gyorsan meggyőződhet, hogy a kétszemélyes játéknál sokkal többről van itt szó. Többszemélyes játékban a játékosok között általában lehetségesek formális vagy informális szövetségek (aliances). A játék előrehaladtával szövetségek kötötnek és bontatnak fel.

Hogyan is kellene értelmezni egy ilyen viselkedést? Természetes következménye-e a szövetség az egyes játékosok optimális stratégiáinak egy többjátékos játékban? Úgy tűnik, hogy ez igaz lehet. Tegyük fel például, hogy *A* és *B* gyengén, míg *C* erősebben áll. Akkor néha optimális mind *A*, mind *B* számára, ha nem egymást, hanem *C*-t támadják meg, hogy az egyenként ne végezzen velük. Ily módon az együttműködés tisztán egoista viselkedésből is kialakulhat. Persze ahogy az együttes támadásnak kitett *C* gyengül, a szövetség értéke csökken, és vagy *A*, vagy *B* a megegyezést megszegheti. Egyes esetekben az explicit szövetségek az úgyis bekövetkezendő eseményeket rögzítik konkrét módon. Más esetekben szociális megbílyezés jár a szövetség megszegéséért, így a játékosnak mérlegelnie kell a szövetség megszegésének rövid idejű előnyét és a szavahihetetlenként való megbílyezés hosszú távú hátrányát. (Az ilyen bonyodalmakról többet a 17.6. alfejezetben.)

Ha a játék nem zérusszegű, akkor együttműködés két játékos esetén is létrejöhét. Tegyük fel például, hogy létezik olyan végállapot, amelynek hasznossága  $\langle v_A = 1000, v_B = 1000 \rangle$ , és 1000 minden játékos számára az elérőd legnagyobb hasznosság. A két játékos optimális stratégiája akkor az, hogy ennek az állapotnak az elérése érdekében minden megtesznek – azaz a két játékos automatikusan kooperálni fog, hogy a kölcsönösen előnyös célt elérjék.

### 6.3. ALFA-BÉTA NYESÉS

A minimax keresés problémája, hogy a játékban a megvizsgálandó állapotok száma exponenciális a lépések számában. A kitevőtől sajnos megszabadulni nem tudunk, ám lényegében megfelezhetjük. A trükk az, hogy lehetséges a korrekt minimax döntés ki-számítása anélkül, hogy a játékfában minden csomópontra rá kelljen nézni. Ehhez kölcsönözhetjük a 4. fejezetben megismert nyesést (**prunning**) gondolatát, és a játékfa nagyobb részét a megfontolásokból kihagyhatjuk. A konkrét vizsgált technika az **alfa-béta nyesés (alpha-beta pruning)**. Ha ezt egy standard minimax fára alkalmazzuk, ugyanazt az eredményt adja vissza, mint a minimax, a döntésre hatással nem lévő ágakat azonban lenyeli.

Tekintsük ismét a 6.2. ábrán látható egylépés-váltásos játékfát. Kövessük még egyszer az optimális döntés kiszámításának menetét, most azonban kísérjük figyelemmel, hogy mit is tudunk valójában a folyamat minden pontjában. A lépésekhez a 6.5. ábra ad magyarázatot. Az eredmény az, hogy meg tudjuk határozni a minimax döntést anélkül, hogy a két levélcsomópontot bármikor is megnéznénk.

A módszert értelmezhetjük másiképpen is, mint a MINIMAX-ÉRTÉK függvény egy egyszerűsítését. Legyen a  $C$  csomópontot a 6.5. ábrán látható, ki nem értékelt követő csomópontok értéke  $x$  és  $y$ , és legyen  $z$  a kettő minimuma. A gyökércsomópont értéke ekkor:

$$\begin{aligned}\text{MINIMAX-ÉRTÉK(gyökér)} &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{ahol } z \leq 2 \\ &= 3\end{aligned}$$

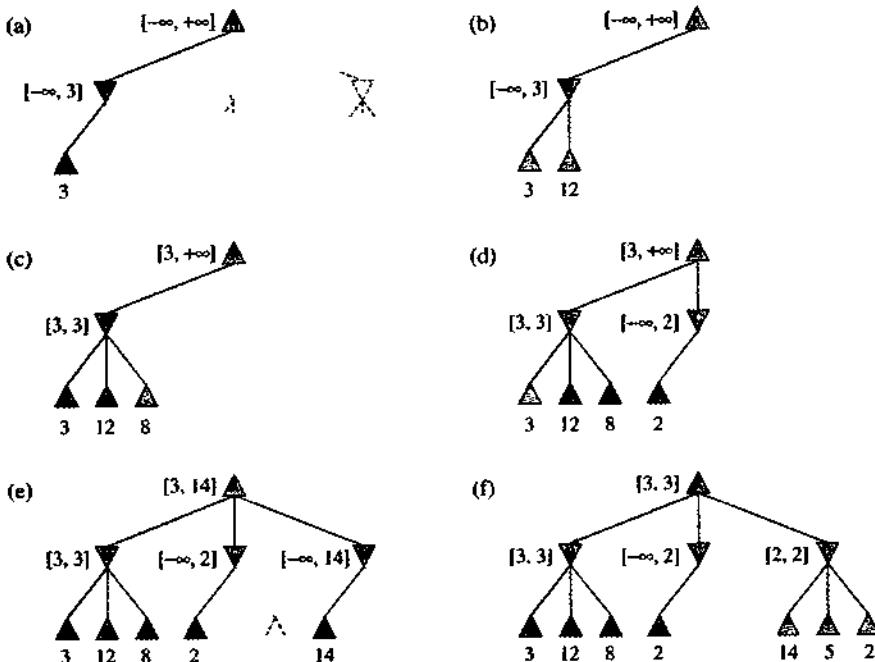
Más szóval a gyökér értéke és így a minimax döntés *független* a lenyestett levelek  $x$  és  $y$  értékeitől.

Az alfa-béta nyesést tetszőleges mélységű fákra lehet alkalmazni, és sokszor lehetséges, hogy levelek helyett teljes részfákat nyessünk le. Az általános elv az alábbi: tekintünk egy olyan  $n$  csomópontot valahol a fa belsejében (lásd 6.6. ábra), hogy a Játékosnak lehetősége legyen e csomópontba lépni. Ha a Játékosnak van az  $n$  szülőcsomópontjánál vagy ettől feljebb bármelyik döntési pontban egy jobb választása,  $m$ , akkor az  $n$ -t az aktuális játékban soha nem érjük el. Ha az  $n$ -ről tehát (néhány követőjének megvizsgálásával) már eleget tudunk, hogy ehhez a konklúzióhoz jussunk, a csomópontot nyugodtan lenyeshetjük.

Emlékezzünk, hogy a minimax keresés mélyiségi keresés, egy adott időben tehát elegendő a fában egy egyedi út menti csomópontokkal foglalkozni. Az alfa-béta nyesés megnevezése két paramétertől származik, mely paraméterek az út mentén megjelenő visszaléptetett értékek korlátjaira vonatkoznak:

$\alpha$  = az út mentén tetszőleges döntési pontban a MAX számára eddig megtalált legjobb (azaz a legmagasabb értékű) választás értéke.

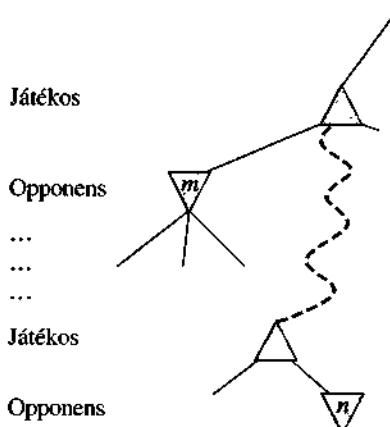
$\beta$  = az út mentén tetszőleges döntési pontban a MIN számára eddig megtalált legjobb (azaz a legkisebb értékű) választás értéke.



**6.5. ábra.** Az optimális döntés kiszámításának lépései a 6.2. ábrán látható játékfa esetén. minden ponton a lehetséges értékek terjedelmét mutatjuk meg minden egyes csomópont számára. (a) A B csomópont alatt az első levélnek 3 az értéke. B tehát, ami egy MIN csomópont, legfeljebb 3 értékű. (b) A B alatti második csomópontnak 12 az értéke. MIN ezt a lépést elkerüli, így B értéke még mindig legfeljebb 3. (c) A B alatti harmadik levélnek 8 az értéke. B összes követőit láttuk már, B értéke tehát pontosan 3. Most azt következtethetjük ki, hogy a gyökér értéke legalább 3, mert MAX-nak a gyökérben 3-as értékű választása van. (d) A C alatti első levélnek 2 az értéke. C tehát, ami egy MIN csomópont, legfeljebb 2 értékű. De mi tudjuk már, hogy B-nek 3 az értéke, MAX tehát C-t soha nem fogja választani. Azért nem is érdekes C további követőit megvizsgálni. Ez az alfa-béta nyesés egy példája. (e) A D alatti első levélnek 14 az értéke, így D értéke legfeljebb 14. Ez több mint MAX legjobb alternatívája (azaz 3), meg kell vizsgálni ezért D követőit. Jegyezzük meg, hogy most a gyökér minden követője esetén rendelkezünk értékkorláttal, a gyökér értéke legfeljebb 14. (f) D második követőjének értéke 5, így tovább kell folytatnunk a vizsgálatot. A harmadik követő értéke 2, D értéke tehát pontosan 2. MAX döntése a gyökérnél, hogy B felé kell lépni, 3-as értékkel.

Az alfa-béta keresés az  $\alpha$  és a  $\beta$  értékeit munka közben frissíti, és a csomópontnál a megmaradó ágakat lenyeli (a rekurzív hívást terminálja), amint csak biztosával válik, hogy az aktuális csomópont értéke rosszabb lesz, mint az aktuális  $\alpha$  és  $\beta$  érték. MAX-ra, illetve MIN-re. A teljes algoritmust a 6.7. ábra mutatja. Bátorítjuk az olvasót, hogy a 6.5. ábrán látható fára alkalmazva, kövesse végig az algoritmus működését.

Az alfa-béta nyesés hatékonyisége erősen függ a követő csomópontok vizsgálatának sorrendjétől. Például a 6.5. (e) és (f) ábrán D egyetlen követőjét sem lehetne lenyelni, mert először a legrosszabb követőt (MIN szemszögből) generáltuk. Ha a harmadik követőt elsőnek generálnánk, a maradó kettőt le lehetne nyesni. Ez azt sugallja, hogy érdemes lenne először a legjobbnak tűnő csomópontokat megvizsgálni.



**6.6. ábra.** Alfa-béta nyesés: általános eset. Ha  $m$  a Játékos számára jobb, mint  $n$ , akkor a játék során sosem fogunk elérni  $n$ -be.

Ha feltételezzük, hogy ez megtehető,<sup>4</sup> akkor az alfa-béta nyesésnek elég az  $O(b^{m/2})$  csomópontot megvizsgálni, a legjobb lépés kiszámításához, a minimax  $O(b^m)$  értékével szemben. Ez azt jelenti, hogy az effektív elágazási tényező  $b$  helyet  $\sqrt{b}$  lesz – például a sakk esetében 35 helyett 6. Más szóval, az alfa-béta nyesés ugyanannyi idő alatt kétszer olyan messzire néz, mint a minimax. Ha a követő csomópontokat véletlen sorrendben vizsgálnánk a legjobbat-először helyett, a vizsgált csomópontok összszáma  $O(b^{3m/4})$  lenne, mérsékelt értékű  $b$ -kre. A sakk esetében egy igen egyszerű rendezőfüggvény (mint például az, hogy először a leütésekkel, majd a támadásokkal, ezt követően az előrelépésekkel és végül a hátralépésekkel próbálkozunk) a legjobb esetre vonatkozó  $O(b^{m/2})$  eredményt egy 2-es tényező erejéig közelíti meg. Dinamika hozzáadása a lépésrendező sémkéhoz, mint például azokkal a lépésekkel próbálkozni először, amelyek legutóbb a legjobbnak bizonyultak, az elvi korláthoz egészen közel visz.

A 3. fejezetben megjegyeztük, hogy a keresési fában az ismétlődő állapotok a keresési költségekben exponenciális növekedéshez is vezethetnek. Játékokban gyakran előfordulnak ismétlődő állapotok a **transpozíciók** (*transpositions*) – a lépési szekvenciának az ugyanahoz az álláshoz vezető különböző permutációi – miatt. Ha például Fehér lépése  $a_1$ , amire Fekete  $b_1$  lépéssel válaszolhat, és a tábla más részén van egy vele nem kapcsolatos  $a_2$  lépése, amire Fekete  $b_2$ -vel válaszolhat, akkor az  $[a_1, b_1, a_2, b_2]$  és az  $[a_1, b_2, a_2, b_1]$  szekvenciák mind ugyanabban az állásban végződnek (ahogy ebben az állásban végződik az előbbi lépések minden  $a_2$ -vel kezdődő permutációja). Érdemes ennek az állásnak a kiértékelését az első előforduláskor egy hash-táblában eltárolni, mert így a későbbi előfordulásoknál a kiszámítás megismétlését megspóroljuk.

Az előbb látott pozíciók hash-tábláját hagyományosan **transpozíciós táblának** (*transposition table*) hívják. Ez lényegében azonos a GRÁF-KERESÉS zárt listájával (lásd 122. oldal). A transpozíciós tábla használata drámai hatású lehet és a sakkban

<sup>4</sup> Természetesen nem tökéletes módon, killőben a sorbarendező-függvény segítségével egy hibátlan játékot tudnánk játszani.

**function ALFA-BÉTA-KERESÉS(*állapot*) returns egy cselekvés**

**inputs:** *állapot*, a játék jelenlegi állása

$v \leftarrow \text{MAX-ÉRTÉK}(\text{állapot}, -\infty, +\infty)$

**return** egy cselekvés a KÖVETŐÁLLAPOTOK(*állapot*)-ban  $v$  értékkel

**function MAX-ÉRTÉK(*állapot*,  $\alpha$ ,  $\beta$ ) returns egy hasznossági érték**

**inputs:** *állapot*, a játék aktuális állása

$\alpha$ , az *állapot*-ig vezető út mentén a legjobb érték MAX számára

$\beta$ , az *állapot*-ig vezető út mentén a legjobb érték MIN számára

**if** VÉG-TEST(*állapot*) **then return** HASZNOSÍTÁS(*állapot*)

$v \leftarrow -\infty$

**for minden**  $s \in$  KÖVETŐÁLLAPOTOK(*állapot*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-ÉRTÉK}(s, \alpha, \beta))$

**if**  $v \geq \beta$  **then return**  $v$

$\alpha \leftarrow \text{MAX}(\alpha, v)$

**return**  $v$

**function MIN-ÉRTÉK(*állapot*,  $\alpha$ ,  $\beta$ ) returns egy hasznossági érték**

**inputs:** *állapot*, a játék aktuális állása

$\alpha$ , az *állapot*-ig vezető út mentén a legjobb érték MAX számára

$\beta$ , az *állapot*-ig vezető út mentén a legjobb érték MIN számára

**if** VÉG-TEST(*állapot*) **then return** HASZNOSÍTÁS(*állapot*)

$v \leftarrow +\infty$

**for minden**  $s \in$  KÖVETŐÁLLAPOTOK(*állapot*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-ÉRTÉK}(s, \alpha, \beta))$

**if**  $v \leq \alpha$  **then return**  $v$

$\beta \leftarrow \text{MIN}(\beta, v)$

**return**  $v$

**6.7. ábra.** Az alfa-béta keresési algoritmus. Ugyanazt a számítást hajtja végre, mint egy normál minimax algoritmus a 6.3. ábrán, kivéve két sort a MAX-ÉRTÉK és a MIN-ÉRTÉK hívásokban, ahol az  $\alpha$  és a  $\beta$  értékeket kezeli (és adminisztrálja e paramétereik átadását).

néha az elérhető mélység megduplázásához vezet. Másrészt ha másodpercenként a csomópontok millióit értékeljük ki, nem praktikus ezeket *mind* a transzpozíciós táblában tartani. A legértékesebb csomópontok megválasztására különböző stratégiákat dolgoztak ki.

## 6.4. NEM TÖKÉLETES, VALÓS IDEJŰ DÖNTÉSEK

A minimax algoritmus a játék egész keresési terét állítja elő, az alfa-béta nyesés viszont lehetőséget ad annak nagy részét lenyenesni. Az alfa-béta nyesésnek mégis a végállapotokig kell keresnie legalább a keresési tér egy részében. Ez a mélység általában nem praktikus, mert a lépéseket elfogadható idő alatt – tipikusan legfeljebb percek alatt – kell megtenni. Shannon az 1950-es *Programming a computer for playing chess* c. cikkében azt javasolta, hogy a programnak a keresést korábban kell abbahagynia, és a csomópontrakra egy heurisztikus kiértékelő függvényt (evaluation function) kell alkalmaznia, a

nem végállapotokat lényegében végállapotoknak tekintve. Más szóval a minimaxot vagy az alfa-bétát kétféle módon kellene módosítani: a hasznossági függvényt egy heurisztikus **KIERTÉKEL** kiértékelő függvény helyettesíti, amely az állás hasznosságának egy becslését adja meg, valamint a célállapottesztet egy vágási teszt (cutoff test) váltja fel, amely eldönti, hogy **KIERTÉKEL**-t kell-e alkalmazni.

## Kiértékelő függvények

Egy kiértékelő függvény egy adott állásból kiindulva a játék várható hasznosságának becslését adja vissza, éppúgy, ahogy a 4. fejezet heurisztikus függvénye a céltól való távolság becslését adja. A becslés ötlete, amikor Shannon felvetette, nem számított újnak. A sakkozók (és természetesen más játékok szerelmesei) az évszázadok során kifejlesztettek olyan módszereket, amelyekkel mérlegelni tudták egy állás értékét, hiszen a keresés mennyiségenek tekintetében az emberek még inkább korlátozottak, mint a számítógépes programok. Nyilvánvaló, hogy egy játékprogram teljesítménye függ az alkalmazott kiértékelő függvény minőségétől. Egy pontatlan függvény a programot olyan állásokhoz vezeti, amelyek valójában vesztes pozíciók. Milyen pontosan tudunk jó kiértékelő függvényeket tervezni?

Először is a kiértékelő függvénynek a **végállapotokat** ugyanúgy kellene sorba rendeznie, mint az igazi hasznosságfüggvénynek, máskülönben az azt használó ágens még akkor is szuboptimális lépéseket választhatna, ha történetesen a játék végéig előre látta minden. Másodsorban a kiértékelő függvény kiértékelése nem tarthat túl sokáig! (A kiértékelő függvény szubruttinként meg tudná hívni a MINIMAX-DÖNTÉS-t, és ki tudná számítani a pozíció pontos értékét, ez azonban az egész gyakorlat értelmét – az időmegtakarítást – megkérdőjelezné.) Harmadsorban a nem végállapotok tekintetében a kiértékelő függvénynek pontosan kell tükröznie a nyerés valódi esélyét.

El lehet merengeni a „nyerés esélye” kifejezésen. Végül is a sakk nem szerencsejáték, az aktuális állapotot biztosan tudjuk, és nincs a dologban kockavetés. Ha azonban egy nem végállapotban levágtuk a keresést, akkor az algoritmus szükségszerűen bizonytalan lesz annak valódi kimenetelében. Ezt a bizonytalanságot nem információs, hanem inkább számítási korlátok okozzák. Ha a kiértékelő függvény egy adott állapotban csak egy bizonyos mennyiségi számítást végezhet el, a legjobb, amit tehet, hogy kitalálja a végkimenetelt.

Próbáljuk meg ezt az ötletet pontosan megfogalmazni. A kiértékelő függvények többsége az állapot különböző **tulajdonságait** (**features**) kiszámítva dolgozik – a sakkban például ilyen a minden fénél lévő gyalogok száma. A tulajdonságok, egybevéve, az állapotok különböző **kategóriáit**, avagy **ekvivalenciaosztályait** definiálják: az egyes kategóriákhoz tartozó állapotok az összes tulajdonság szempontjából ugyanazt az értéket képviselik. minden egyes kategória általánosságban tartalmazni fog olyan állapotokat, melyek győzelemhez vezetnek, olyanokat, amelyek döntetlen eredményeznek, és olyanokat, amelyek vereséggel végződnek. A kiértékelő függvény nem tudhatja, hogy melyik állapot melyik, azonban visszatérhet egyetlen olyan értékkel, amely tükrözi az egyes kimenetelekhez tartozó állapotok *urányát*. Tegyük fel például, hogy a tapasztalatunk azt sugallja, hogy a kategóriához tartozó állapotok 72%-a vezet győzelemhez (+1 hasznosság), 20%-a veszteshez (-1 hasznosság) és 8%-a döntetlenhez (0 hasznosság). Akkor a kategóriához tartozó állapotok értelmes értékelése a súlyozott átlag vagy

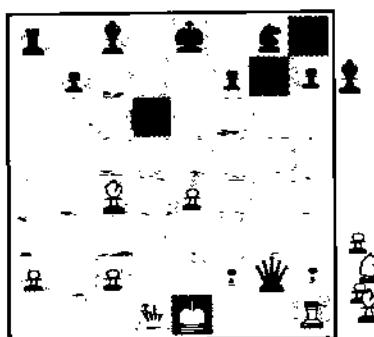
a várható érték (**expected value**):  $(+1 \times 0,72) + (-1 \times 0,20) + (0 \times 0,08) = 0,52$ . Elvben a várható értéket minden kategóriára meg lehetne határozni, ami egy tetszőleges állapot esetén is működő kiértékelő függvényhez vezetne. A végállapotokhoz hasonlóan, a kiértékelő függvénynek nem kell tényeges várható értékeket visszaadnia, amíg az állapotok *sorba rendezése* ugyanaz.

A gyakorlatban az ilyen elemzés túl sok kategóriát és így túl sok tapasztalatot kíván ahhoz, hogy a győzelem valószínűségét megállapítsuk. A kiértékelő függvények többsége ehelyett az egyes tulajdonságokból adódó numerikus értékeket külön számítja, majd a végleges értéket ezek *kombinációjaként* határozza meg. A kezdőknek szánt sakk-könyvek például egy közelítő **pontértéket** (**material value**) rendelnek minden egyes bábuhoz: minden gyalog 1 pontot, egy huszár vagy egy futó 3 pontot, egy bánya 5 pontot, míg egy vezér 9 pontot ér. Más jellemzők, mint például egy „jó gyalogstruktúra” és a „király biztonsága” érhetnek mondjuk egy fél gyalogot. Az állás kiértékeléséhez ezeket az értékeket egyszerűen összeadjuk. Egy gyaloggal ekvivalens biztonságos előny a győzelem lényeges valószínűségét adja, három gyaloggal ekvivalens biztonságos előny majdnem biztos győzelemhez vezet, ahogy ezt a 6.8. (a) ábra mutatja. Az ilyen típusú kiértékelő függvényt a matematikában **súlyozott lineáris függvénynek** (**weighted linear function**) nevezik, mert kifejezhető az alábbi alakban:

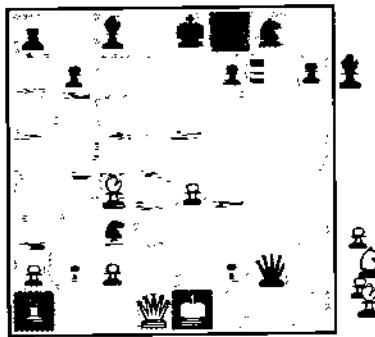
$$\text{KIÉRTÉKEL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

ahol a  $w$ -k a súlyok és az  $f$ -ek az adott táblaállás jellemzői. Sakk esetében az  $f_i$ -k a táblán levő egyes bábufajták számát jelölhetik és a  $w_i$ -k a bábukhoz rendelt pontértékek (gyalog 1, futó 3 stb.) lehetnek.

A tulajdonságértékek összeadása értelmes dolognak tűnik, azonban ezzel valójában egy igen erős feltételezéssel élünk: t. i. hogy az egyes tulajdonságok hozzájárulásai más tulajdonságok értékeitől *függetlenek*. A 3-as érték hozzárendelése a futóhoz figyelmen kívül hagyja például azt, hogy a végijátékban, amikor sok tér áll rendelkezésre a manőverezéshez, a futó erősebb. Ezért a jelenlegi sakk- és más játékokat játszó programok a tulajdonságok *nemlineáris* kombinációt is használják. Egy futópár egy kicsit értékesebb lehet például, mint egy futó értékének kétszerese, és egy futó a végijátékban értékesebb, mint a nyitásnál.



(a) Fehér lép



(b) Fehér lép

6.8. ábra. Két, kissé eltérő sakktáblaállás. (a) Feketének egy huszár és két gyalog előnye van, és a játékot megnyeri. (b) Fekete veszít, miután Fehér a vezérét leüti.

Az éles elméjű olvasó észreveszi, hogy a tulajdonságok és a súlyok a sakk szabályainak *nem* részei! Ezek az emberi sakkjátzsás évszázados tapasztalataiból származnak. Lineáris kiértékelő képletet alkalmazva a tulajdonságok és a súlyok az állapotoknak az értékeik szerinti igazi sorrendezésük legjobb közelítését adják meg. A tapasztalat különösképpen azt sugallja, hogy az egy pontnál nagyobb biztonságos anyagi előny valószínűleg győzelemhez vezet, ha minden más tényező azonos. Hárompontos előny elegendő a majdnem biztos győzelemhez. Olyan játékokban, ahol ilyen tapasztalatra szert tenni nem lehet, a kiértékelő függvény súlyértékeit a 18. fejezetben leírt gépi tanulási technikákkal lehet becsülni. Biztató, hogy e technikák sakkra történő alkalmazása tényleg azt támasztja alá, hogy egy futó kb. három gyalogot ér.

## A keresés levágása

Következő lépés az ALFA-BÉTA-KERÉS olyan módosítása, hogy az algoritmus meghívja a heurisztikus Kiértékel függvényt, amikor a keresést le kell vágni. Az implementációt tekintve a 6.7. ábrának a VÉG-TESZT-et tartalmazó két sorát az alábbi sorra cseréljük:

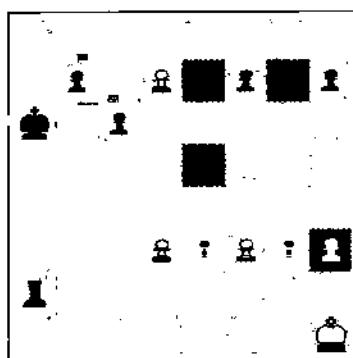
```
if LEVÁGÁS-TESZT(állapot, mélység) then return KIÉRTÉKEL(állapot)
```

Meg kell oldani annak az adminisztrálását is, hogy az aktuális *mélység* inkrementálisan változzék minden rekurzív hívás alkalmából. A legkézenfekvőbb megközelítés a keresés vezérlésére, ha a kereséshez rögzített *mélységekorlátot* rendelünk, hogy a LEVÁGÁS-TESZT(*állapot*, *mélység*) *igazzal* térjen vissza egy *d*-nél nagyobb *mélységre* (*igazzal* kell visszatérnie minden végállapot esetében is, ahogy azt a VÉG-TESZT tette). A *d* mélységet úgy választják meg, hogy a felhasznált idő ne haladja meg a játék szabályai által megengedett időt.

Egy kicsit robusztusabb megközelítést kapunk, ha a 3. fejezetben definiált iteratívan mélyülő algoritmust alkalmazzuk. Ha a program kifut az időből, akkor a legmélyebb, befejezett keresés által kiválasztott lépést adja vissza. Az ilyen megközelítések azonban hibákhoz vezethetnek a kiértékelő függvény közelítő jellege miatt. Tekintsük ismét a sakkban a pontelőnyön alapuló kiértékelő függvényt. Tegyük fel, hogy a program a mélységekorlátig keres, és a 6.8. (b) ábra táblaállásába jut, ahol Feketének huszár és két gyalog előnye van. A program a heurisztikus érték alapján ezt az állapotot Fekete számára valószínű győzelemnek fogja minősíteni. Fehér következő lépései között azonban leüti a fekete vezért, kárpótlás nélkül. Az állás így valójában Fehér számára jelent győzelmet, viszont hogy ezt belássuk, egy további lépésváltásig előre kellene nézni.

Nyilvánvalóan egy kifinomultabb levágási tesztre van szükség. A kiértékelő függvényt csak az **egyensúlyi (quiescent)** táblaállásokra szabad alkalmazni, vagyis olyan állásokra, amelyek értéke a közeljövőben nem változik meg radikálisan. A sakkban például az olyan táblaállások, amelyeknél tisztet lehet leütni, egy csak a pontértéket figyelembe vevő kiértékelő függvény esetén nem tekinthetők egyensúlyi állásnak. A nem egyensúlyi táblaállásokat tovább ki lehet fejteni, amíg egyensúlyi állásokat nem érünk el. Ezt a többletkeresést **egyensúlyi keresésnek (quiescence search)** nevezzük. Ezt a keresést néha csak bizonyos típusú lépésekre korlátozzák, mint például a táblaállás bizonytalanságait gyorsan feloldó bábuleütésekre.

A horizontproblémát (**horizont problem**) nehezebb megszüntetni. A probléma akkor jelentkezik, amikor a program az ellenfél egy olyan következő lépései kerül szembe, ami komoly károkat okoz, és egyben elkerülhetetlen. Tekintsük a 6.9. ábra sakkjátszmáját. Fekete egy kicsit erősebb pontértékű, ha azonban Fehér a hetedik sorról a nyolcadikba tudja juttatni a gyalogját, akkor vezérré változik, és Fehér könnyen megnyeri a játszmát. Fekete ezt 14 lépésváltással elodázhatja, ha bástyával sakkot ad, de a fehér gyalog elkerülhetetlenül vezérré fog változni. A rögzített mélységű keresések problémája, hogy ezek azt gondolják, hogy ilyen elodázó lépésekkel el lehet kerülni a vezérré változtató lépést. Azt mondjuk, hogy az elodázó lépések a vezér váltását „a horizonton túlra kitolják”, egy olyan helyre, ahol ezt nem lehet észrevenni.



Fekete lép

**6.9. ábra.** A horizonthatás. A fekete bástyával adott sorozatos sakk a „horizonton túlra” kényszeríti az elkerülhetetlen vezérré változtató lépést, és ezt a táblaállást enyhén előnyös állásként tünteti fel Fekete számára, holott Fehér számára ez egy biztos nyerő állás.

Mivel a hardverek fejlődése mélyebb kereséshez vezet, azt várjuk, hogy a horizontprobléma kevésbé lesz gyakori, a nagyon hosszú késleltető lépésszekvenciák igen ritkák. A **szinguláris kiterjesztés** (**singular extension**) használata szintén hatásos a horizontprobléma elkerülésében. Ilyenkor a keresési költségtöbblet nélkül. A szinguláris kiterjesztés egy olyan lépés, amely „lényegesen jobb”, mint az állás minden más lépése. A szinguláris kiterjesztés keresése lényeges többletköltség nélkül a normális mélységekorláton túl is mehet, mert elágazási tényezője egységes. (Az egyensúlyi keresést a szinguláris kiterjesztés egy változatának lehet tekinteni.) A 6.9. ábrán a szinguláris kiterjesztés a lehetséges vezérlépester meg fogja találni, feltéve, hogy Fekete sakkot adó és Fehér védekező lépései „nyilvánvalóan jobb”-nak azonosítja a többi alternatívához képest.

Eddig a keresés egy adott szinten történő levágásáról és az eredményt igazoltan nem befolyásoló alfa-béta nyelésről beszélünk. De **előrenyelést** (**forward pruning**) is alkalmazhatunk, ahol egy adott csomópontban egyes lépéseket minden további elemzés nélkül azonnal lenyelünk. A sakkozók többsége egy adott állásnál nyilván csak néhány lépést vesz figyelembe (legalább tudatosan). Sajnos ez a megközelítés veszélyes lehet, mert nincs garancia arra, hogy nem fogjuk a legjobb lépést lenyenesni. Ez katasztrófális lehet, ha a gyökér közelében történik, mert a program sokszor néhány „nyilvánvaló” lépést el fog nézni. Az előrenyelést speciális helyzetekben biztonságosan lehet alkال-

mazni – például amikor a két lépés szimmetrikus, vagy más szempontból ekvivalens, elegendő csak az egyiket figyelembe venni –, vagy olyan csomópontoknál, amelyek a keresési fában mélyen találhatók.

Az itt leírt technikák kombinált alkalmazásával egy elfogadható szinten sakközö (vagy más játékot játszó) programot kaphatunk. Tegyük fel, hogy a sakk esetére implementáltunk egy kiértékelő függvényt, egy értelmes kereséslevágást egyensúlyi keresés-sel és egy nagy transzpozíciós táblát. Tegyük fel azt is, hogy hónapokig tartó munkás bitfaragással, a legújabb PC-n tudunk generálni és kiértékelni másodpercenként kb. egy millió csomópontot, ami kb. 200 millió csomópont megvizsgálását teszi lehetővé lépé-senként a standard időkorlátok mellett (3 perc lépésenként). A sakk elágazási tényezője átlagosan kb. 35. Mivel  $35^5$  kb. 50 millióval egyenlő, a minimax keresést alkalmazva így csak öt lépésváltásig tudnánk előrenézni. Bár egy ilyen program nem hasznávehetetlen, egy átlagos emberi játékos, aki esetenként hat vagy nyolc lépésváltásig tud előrenézni, könnyűszerrel bolondot tudna belőle csinálni. Az alfa-béta nyeséssel kb. 10 lépésváltást kapunk, ami a játék mesteri színvonalának felel meg. A 6.7. alfejezet további nyesési technikákat ír le, amikkel az effektív keresési mélység durván 14 lépésváltásig kiter-jeszthető. A nagymesteri minősítéshez egy lényegesen finomított kiértékelő függvény kellene továbbá az optimális megnyitások és a végjátékok nagy adatbázisa. És az sem ártana, ha lenne egy szuperszámítógépünk, amin a programot futtnánk.

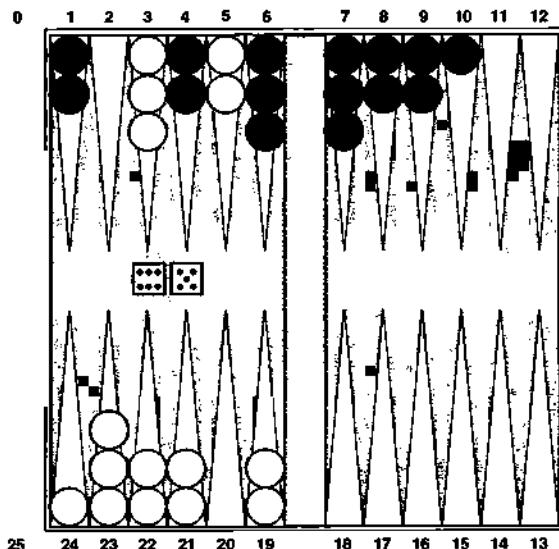
## 6.5. VÉLETLEN ELEMET IS TARTALMAZÓ JÁTÉKOK

A valós életben számos olyan megjósolhatatlan külső eseménnyel van dolgunk, ami miatt előre nem látható helyzetbe kerülünk. Sok játék ezt a megjósolhatatlanságot egy véletlen elem, mint például a kockadobás, bevezetésével veszi figyelembe. Ily módon egy lépéssel közelebb visznek bennünket a valósághoz, és megéri megvizsgálni, hogy ez milyen hatással van a döntési folyamatra.

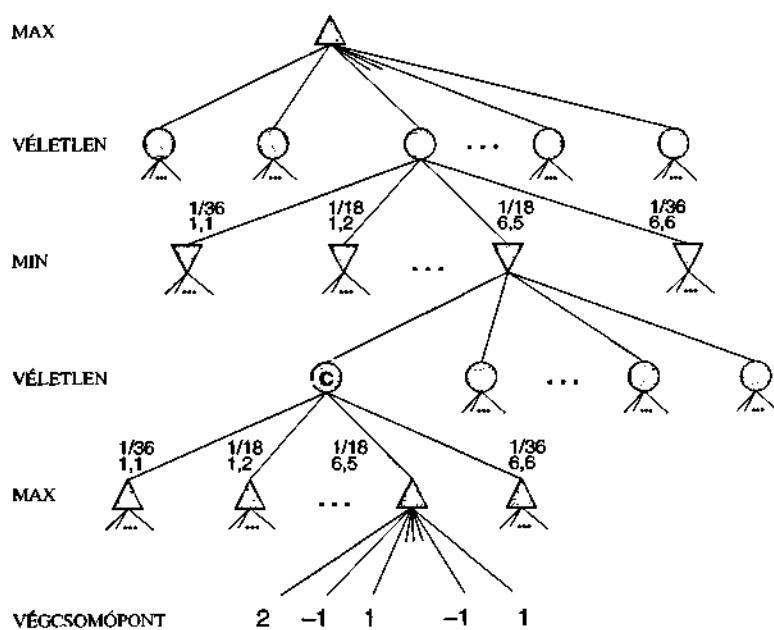
Az ostábla tipikusan olyan játék, amely ötvözi a szerencsét és a tudást. A játékosok a lépések előtt kockát dobnak, amivel meghatározzák az adott játékos számára megengedett lépéseket. A 6.10. ábra ostáblaállásánál Fehér egy 6-ost és egy 5-öst dobott, és négy lehetséges lépés közül választhat.

Habár Fehér tudja, hogy számára melyek a megengedett lépések, azt nem tudja, hogy Fekete milyen számokat fog dojni, így nem tudhatja, hogy melyek lesznek Fekete megengedett lépései. Ez azt jelenti, hogy Fehér nem tud egy  $3 \times 3$ -as amőbánál látott teljes játékfához hasonlót elkészíteni. Az ostábla esetén a játékfának a MAX és MIN csomópontok mellett tartalmaznia kell véletlen csomópontokat (chance nodes) is. A 6.11. ábrán a véletlen csomópontokat körök jelölik. A véletlen csomópontokból kiinduló ágak a lehetséges kockadobásokat jelölik. minden ág a dobott értékkel és előfordulásának valószínűségével van felcímkezve. Két kockával történő dobásnál harminchatsfél eredményt kaphatunk. Ezek a dobások mind azonos valószínűségek, mivel azonban a 6–5 megegyezik az 5–6-tal, így csak 21 különböző dobás létezik. A hat dupla (1–1-től egészen 6–6-ig) 1/36-od valószínűséggel, míg a többi 15 különböző dobás 1/18-ad valószínűséggel fordul elő.

A következő lépés annak megértése, hogyan kell helyes döntéseket hozni. Nyilvánvalóan továbbra is azt a lépést szeretnénk kiválasztani, ami a legjobb álláshoz vezet. A lehetséges állások azonban most nem rendelkeznek egy jól meghatározott minimax



**6.10. ábra.** Egy tipikus ostáblaállás. A játék lényege, hogy az egyik fél összes bábuját eltávolítsuk a tábláról. Fehér az óramutató járásával egyező irányban halad a 25-ös felé, míg Fekete az óramutató járásával ellentétes irányban, a 0 felé halad. Egy bábuval bármelyik helyre lehet lépni, kivéve azokat a helyeket, ahol az ellenfélnek már legalább két bábuja található. Ha egy olyan helyre lépünk, ahol az ellenfélnek egy bábuja található, akkor azt a bábut elfognak, és a lépkedést az elejéről kell újrakezdenie. Ennél a táblaállásnál Fehér épp most dobott 6–5-öt, és négy megengedett lépés közül választhat: (5–10,5–11), (5–11,19–24), (5–10,10–16) és (5–11,11–16).



**6.11. ábra.** Egy ostáblaállás sematikus játékfája

értékkel. Ehelyett csak egy átlagos vagy várható értéket (*expected value*) tudunk kiszámítani, ahol az átlagolást az összes lehetséges kockadobásra végezzük el. Ezzel a determinisztikus játékok **minimax értékét** (*minimax value*) a véletlen csomópontokat tartalmazó játékok **várhatóminimax értékére** (*expectiminimax value*) általánosítottuk. A végállapotok és a MAX és MIN állapotok (ahol a kockadobás eredménye ismert) ugyanúgy viselkednek, mint eddig. A véletlen csomópontok kiértékeléséhez az összes lehetséges kockadobás figyelembevételével számított súlyozott átlagot kell venni, vagyis:

$$\text{VÁRHATÓMINIMAX}(n) = \begin{cases} \text{HASZNOSÍG}(n) & \text{ha } n \text{ egy végállapot} \\ \max_{s \in \text{Követők}(n)} \text{VÁRHATÓMINIMAX}(s) & \text{ha } n \text{ egy MAX csomópont} \\ \min_{s \in \text{Követők}(n)} \text{VÁRHATÓMINIMAX}(s) & \text{ha } n \text{ egy MIN csomópont} \\ \sum_{s \in \text{Követők}(n)} P(s) \cdot \text{VÁRHATÓMINIMAX}(s) & \text{ha } n \text{ egy véletlen csomópont} \end{cases}$$

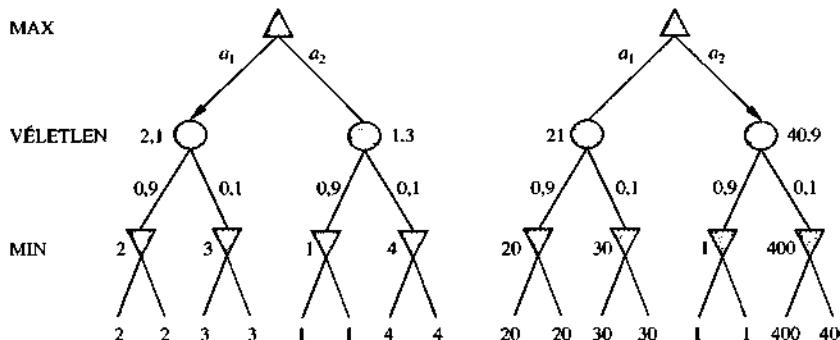
ahol a véletlen  $n$  csomópont állapotátmenet-függvénye az  $n$  állapotot egyszerűen módosítja az összes kockadobás eredményezte  $s$  követő állapot értékével és a dobás  $P(s)$  valószínűségével. Ezt az eljárást rekurzívan alkalmazhatjuk a gyökérig, ugyanúgy, mint a minimax esetén. Az algoritmus részleteit feladatnak hagyjuk meg az olvasónak.

## Az állás kiértékelése véletlen csomópontokat tartalmazó játékok esetén

Hasonlóan a minimaxhoz, a várhatóminimax esetén is alkalmazhatjuk a nyilvánvaló közelítést, vagyis valamelyen ponton levághatjuk a keresést, és a levelekre egy kiértékelő függvényt alkalmazhatunk. Úgy gondolhatnánk, hogy egy olyan játék esetén, mint az ostábla, a kiértékelő függvény elvben semmiben sem tér el a sakknál alkalmazott kiértékelő függvényektől, melyeknek most is magasabb pontot kell adniuk a jobb állásokra. Valójában azonban a véletlen csomópontok jelenléte azt jelenti, hogy sokkal körültekintőbbnek kell lennünk abban a tekintetben, hogy a kiértékelő függvény értékei mit jelentenek. A 6.12. ábra mutatja a problémát: ha a kiértékelő függvény a levelekhez az  $[1, 2, 3, 4]$  értékeket rendeli hozzá, az  $A_1$  lépés lesz a legjobb, az  $[1, 20, 30, 400]$  értékekkel az  $A_2$  lesz a legjobb. Ebből adódóan egy program teljesen másként viselkedik, ha átskálazzuk a kiértékelő függvény értékeit! Kiderül, hogy ha el akarjuk kerülni ezt az érzékenységet, akkor a kiértékelő függvény az egy adott állásból való győzelem valószínűségének (vagy általánosabban az állás várható hasznosságának) csak egy pozitív *lineáris transzformációja* lehet. Ez a bizonytalanságot is tartalmazó helyzetek fontos és általános tulajdonsága. A 16. fejezetben tovább foglalkozunk ezzel a kérdéssel.

## A várhatóminimax komplexitása

Ha a program előre tudná a játék folyamán a későbbiekben előforduló összes kockadobás eredményét, akkor egy kockadobást is tartalmazó játék megoldása ugyanolyan lenne, mint egy kockadobást nem tartalmazó játéké, amit a minimax  $O(b^m)$  idő alatt tesz meg. Mivel a várhatóminimax figyelembe veszi az összes lehetséges kockadobás-



6.12. ábra. A levélértékeken végrehajtott sorrendőrző transzformáció megváltoztatja a legjobb lépést

sorozat eredményét is, annak időigénye  $O(b^m n^m)$  lesz, ahol  $n$  a különböző dobások számát jelöli.

Még ha a fa mélységekorlátja egy kis értékű  $d$  is, a minimaxhoz képesti többletköltség miatt irreális túl messzire előrenézni a véletlen játékok többségénél. Ostáblánál  $n$  értéke 21,  $b$  pedig általában 20 körüli értéket vesz fel, de bizonyos helyzetekben akár 4000 is lehet, a dupla kockadobásoknál. Valószínűleg csak három lépésváltást tudunk kezelní.

A problémáról másfelékeképp is gondolkodhatunk: az alfa-béta algoritmus előnye, hogy figyelmen kívül hagyja azokat a későbbi fejleményeket, amelyek a legjobb játék során egyszerűen nem következnek be. Így a valószínű előfordulásokra koncentrál. A kockát használó játékok esetén *nem léteznek* valószínű lépéssorozatok, mivel adott lépések megtételéhez először a kockának kell a megfelelő oldalára esnie, hogy ezek a lépések egyáltalán megengedhetők legyenek. Ez egy általános probléma minden olyan esetben, amikor a bizonyalanság is képbe kerül: a lehetőségek óriási mértékben megsokszorozódnak, és értelmetlennek válik részletes cselekvési tervezet kidolgozni, mert a világ valószínűleg nem ezek szerint fog viselkedni.

Kétségtelen, hogy az olvasónak már eszébe jutott, hogy valami alfa-béta nyeséshez hasonló dolgot a véletlen csomópontokat tartalmazó játékfákra is alkalmazni lehetne. Kiderül, hogy ez valóban lehetséges. A MAX és a MIN csomópontok elemzése változatlan, de egy kis leleményességgel a véletlen csomópontokat is nyeshetjük. Tekintsük a 6.11. ábra C véletlen csomópontját, és nézzük meg, mi történik az értékével, amikor a gyermeket vizsgáljuk és kiértékeljük. Kérdés, hogy lehetséges-e C értékére egy felső korlátot találni, még mielőtt az összes gyermeket megvizsgálnánk? (Idézzük vissza, hogy az alfa-béta nyesések pontosan erre van szüksége ahhoz, hogy egy csomópontot és az abból kiinduló részfát lenyeshesse.) Ez első pillantásra lehetetlennek tűnhet, mivel a C értéke a gyermekei értékének az átlaga, és amíg nem láttuk az összes kockadobás eredményét, addig ez az átlag akármi is lehet, mivel a meg nem vizsgált gyermekek akármilyen értéket felvehetnek. Ha azonban a hasznosságfüggvény lehetséges értékeit korlátok közé szorítjuk, akkor korlátokat kaphatunk az átlagra is. Például ha azt mondjuk, hogy a hasznosságértékek +3 és -3 között lehetnek, akkor a levélcsomópontok értékei már korlátosak, és a véletlen csomópont értékére az összes gyermekének megvizsgálása nélkül is *adhatur* felső korlátot.

## Kártyajátékok

A kártyajátékok sok szempontból érdekesek, nem csupán a szerencsejáték-jellegük miatt. A játékok óriási választékból azokra fogunk összpontosítani, ahol a kártyákat a játsék elején véletlen módon osztják szét, és ahol minden játékos által kapott leosztás más játékosok számára nem megfigyelhető. Ilyen játék a bridzs, a hearts<sup>5</sup> és a póker néhány formája.

Az első pillantásra úgy tűnhet, hogy a kártyajátékok pont olyanak, mint a kockadobásos játékok. A kártyákat véletlenül osztjuk szét, és ez meghatározza a játékosok lehetséges lépéseiit. Itt azonban az összes kocka a legelején gördül le! Ezt a hasonlatot próbáljuk továbbinni. Ki fog derülni, hogy ez a gyakorlatban igen hasznos lesz. Mellesleg hibás is, igen érdekes okok miatt.

Képzeljük el, hogy két játékos, MAX és MIN, négykártyás kétkezes bridzset gyakorol, ahol minden kézleosztás látható. A leosztások az alábbiak, és MAX hív elsőnek:

MAX: ♠ 6 ♦ 6 ♣ 9 8

MIN: ♡ 4 ♠ 2 ♣ 10 5

Tegyük fel, hogy MAX ♣ 9-et játszik. MIN-nek szint kell játszania, tehát ♠ 10-et vagy ♣ 5-öt. MIN ♠ 10-et játszik és elviszi az ütést. Most MIN következik és ♠ 2-vel indul. MAX-nak nincs pikkje (és így az ütést elvinni nem tudja), következésképpen valamelyik kártyáját le kell játszania. Nyilvánvaló választás a ♦ 6, mert a két másik győztes kártya. Most függetlenül attól, hogy MIN mivel indul a következő ütében, MAX az utolsó két ütést elviszi, és a játék két-két ütés szinten döntetlen lesz. Könnyű megmutatni a minimax megfelelő változatával (lásd 6.12. feladat), hogy MAX ♣ 9-es indulása valójában egy optimális választás.

Módosítsuk most MIN leosztását, ♡ 4-et ♦ 4-re cserélve:

MAX: ♠ 6 ♦ 6 ♣ 9 8

MIN: ♦ 4 ♠ 2 ♣ 10 5

A két eset tökéletesen szimmetrikus: a játék lefolyása azonos lesz, kivéve, hogy a második ütésnél MAX a ♡ 6-ot fogja megjátszani. A játék ugyanúgy döntetlen lesz két-két ütéssel, és a ♣ 9 optimális döntés lesz most is.

Egyelőre minden jónak tűnik. Most rejtük el MIN kártyáinak egyikét: MAX tudja, hogy MIN-nek vagy az egyik (♦ 4-gyel), vagy a másik (♣ 4-gyel) leosztása van, de fogalma sincs melyik. MAX a következőképpen érvel:

A ♣ 9-es indulás optimális választás MIN első leosztásával szemben és hasonlóan a második leosztással szemben is, így most is optimálisnak kell lennie, mert tudomásom szerint MIN-nél a két leosztás valamelyike van.

Amit MAX használ, azt általánosságban a „jövőbe látás szerinti átlagolásnak” nevezhetnék. Az ötlet a cselekvés értékelése nem látott kártyák mellett oly módon, hogy először kiszámítunk minimax értékeket minden lehetséges leosztás esetére, majd várható értéket számítunk a leosztásokra nézve, azok valószínűségét felhasználva.

Ha azt gondolja, hogy ez egy értelmes stratégia (vagy ha nincs véleménye, mert nem érti a bridzset), tekintse a következő történetet:

<sup>5</sup> Az Angol-magyar Nagyszótár sem segít – ezeknek a játékoknak nincs a mai magyar nyelvben használatos nevük. (A ford.)

1. nap: Az A út egy halom aranyhoz vezet. A B út egy elágazáshoz vezet. Ha az elágazásnál balra fordul, egy ékszerhegyet talál, ha az elágazásnál jobbra fordul, elüti egy busz.
2. nap: Az A út egy halom aranyhoz vezet. A B út egy elágazáshoz vezet. Ha az elágazásnál jobbra fordul, egy ékszerhegyet talál, ha az elágazásnál balra fordul, elüti egy busz.
3. nap: Az A út egy halom aranyhoz vezet. A B út egy elágazáshoz vezet. Ha jól választ, egy ékszerhalmot talál, ha rosszul, elüti egy busz.

Nyilvánvaló, nem buta döntés, ha az első két napon a B utat választjuk. Nincs olyan épelméjű személy azonban, aki a harmadik napon is maradna B-nél. Mégis pontosan ez az, amit a jövőbe látás szerinti átlagolás sugall. A B út az 1. nap és a 2. nap helyzetekben optimális, következetképpen optimális a 3. nap szituációban is, hiszen az első két eset egyike fog előfordulni. Térjünk vissza a kártyajátékunkhoz: miután MAX ♣ 9-et hív, MIN ♣ 10-zel győz. MIN ♠ 2-vel indul, mint korábban, és most MAX ott van az elágazásnál bármiféle eligazítás nélkül. Ha MAX a ♥ 6-ot játszotta meg, és MIN-nek van még ♥ 4-e, ♥ 4 lesz a győztes, és MAX a játéket elveszíti. Hasonlóan, ha MAX a ♦ 6-ot játszotta meg, és MIN-nek van még ♦ 4-e, MAX szintén veszteni fog. A ♣ 9 első ízben való hívása tehát olyan helyzethez vezet, ahol MAX-nak 50%-os esélye van a vesztésre (sokkal jobb lenne, ha a ♥ 6-ot vagy a ♦ 6-ot hívna elsőnek, döntetlen biztosítva).

A leszűrendő lecke az, hogy amikor hiányos az információ, meg kell fontolni, hogy *mi-lyen információval fogunk rendelkezni a játék minden pillanatában*. MAX algoritmusával az a probléma, hogy feltételezi, hogy minden lehetséges leosztásnál a játék úgy folytatódik, mintha *minden kártya látható lenne*. Ahogy a példánk mutatja, ez olyan cselekvésre kész-tesi MAX-ot, mintha minden *jövőbeli* bizonytalanság feloldódna, ha eljön az ideje. MAX algoritmusára sem fog soha *információgyűjtéshez* folyamodni (vagy a partner informálásához), mert egy-egy leosztáson belül erre nincs szüksége. Az olyan játékoknál, mint a brídzs, gyakran értelmes dolg olyan kártyával indulni, amely segíti kideríteni az ellenség leosztását, vagy a partnerünket a saját leosztásunkról informálja. Ilyen viselkedést automa-tikusan generálhatunk a nem tökéletes információjú játékokra kifejlesztett optimális algoritmussal. Az ilyen algoritmus nem a világállapotok terében (kártyaleosztások), hanem a *hiedelmi állapotok* terében (*belief states*) (hiedelmek, hogy kinek milyen kártyája van, milyen valószínűséggel) keres. Az algoritmust a 17. fejezetben tudjuk majd megfelelően elmagyarázni, miután felépítettük a szükséges valószínűségi apparátust. Abban a fejezetben azzal a nagyon fontos szemponttal is fogunk foglalkozni, hogy a nem tökéletes infor-mációjú játékokban az a legjobb, ha minél kevesebb információt adunk ki az ellenségnak, és ennek legjobb módszere, ha *nem megjósolható* módon cselekszünk. Ez az oka annak, hogy az értelemellenőrök véletlen módon választják meg az ellenőrzések időpontját.

## 6.6. A JELENLEG LEGFEJLETTÉBB JÁTÉKPROGRAMOK

Azt lehetne mondani, hogy a játékok valami olyat jelentenek az MI számára, mint amit a Forma-1 versenyek a gépkocsi-páros számára. A legfejlettebb játékprogramok villámgyors, hihetetlenül finoman hangolt rendszerek, amelyek nagyon fejlett módszereket alkalmaznak, de amelyeknek nem sok hasznát vesszük a vásárláskor. Bár vannak kutatók, akik azt gondolják, hogy a játékok az MI fő vonalát tekintve többé-kevésbé irrele-vánsak, mégis élénk érdeklődést váltanak ki, és folyamatosan fenntartják az innovációt, amit a szélesebb közösséggel fel is karol.

## Sakk

Herbert Simon 1957-ben megjósolta, hogy 10 éven belül a számítógépek legyőzik az emberi világbajnokot. Negyven év múltával hatátszmás kirakatmérkőzésen a Deep Blue legyőzte Garri Kaszparovot. Simon tévedett, de csak egy 4-es szorzóval. Kaszparov azt írta:

A mérkőzés döntő játszmája a 2. játszma volt, amely emlékeimben sebként maradt meg ... láttunk valamit, ami a legvadabb elképzeléseinket is felülmúlta, hogy egy számítógép hogyan lesz képes döntéseinek hosszú távú pozíciós konzekvenciáit előre látni. A gép – emberi veszélyérzetet producálva – megtagadta, hogy egy döntően rövid távú előnyt jelentő állásba kerüljön. (Kaszparov, 1997)

A Deep Blue-t Murray Campbell, Feng-Hsiung Hsu és Joseph Hoane fejlesztették ki az IBM-nél (Campbell és társai, 2002), a Campbell és Hsu által korábban a Carnegie Mellon Egyetemen kifejlesztett Deep Thought tervére alapozva. A győzedelmes gép egy párhuzamos számítógép volt 30 IBM RS/6000 processzorral, amelyek a „szoftverkeresést” futtatták, és 480 VLSI sakk-célprocesszorral, amelyek a lépésgenerálást (beleírtve a lépések sorba rendezését), a fa utolsó néhány szintjén a „hardverkeresést” és a levélcsomópontok kiértékelését valósították meg. A Deep Blue átlagosan másodpercenként 126 millió, csússebességnél 330 millió csomópontot vizsgált meg. Lépésenként 30 milliárd állást generált, rutinszerűen 14-es mélységet elérve. A gép szíve a standard alfa-béta keresés volt, transzpoziciós táblával, a sikeres kulcsa azonban úgy tűnik, az a képessége, hogy kiterjesztéket tudott generálni az érdekes kényszerlépés-sorozatok mélységi korlátjain túl. Egyes esetekben a keresés a 40-es mélységet is elérte. A kiértékelő függvény 8000 tulajdonsággal dolgozott, ezekből egyesek a bábuk igen specifikus konfigurációit is leírták. A rendszer felhasználta a 4000 állást tartalmazó „megnyitások könyvét” és a 700 000 nagymesteri játszmaileírást tartalmazó adatbázist, ahonnan konszenzusos javaslatokat lehetett generálni. A rendszer a megoldott végjáték-pozíciók nagy adatbázisával is rendelkezett, amely minden ötbábus és számos hatfigurás állást tartalmazott. Ez az adatbázis a keresési mélység lényeges kiterjesztését jelentette, lehetségesével Deep Blue számára egyes esetekben a tökéletes játékot annak ellenére, hogy sok lépésre volt még a matthelyzettől.

A Deep Blue sikere megerősítette azt a széles körű hiedelmet, hogy számítógépes játékokban az előrehaladás főleg az egyre hatékonyabb hardver eredménye – ezt a véleményt az IBM is bátorította. Másfelől a Deep Blue megalkotói azt nyilatkozták, hogy a keresés kiterjesztése és a kiértékelő függvény szintén kritikusak voltak (Campbell és társai, 2002). Azonban tudjuk azt is, hogy néhány közelmúltbeli algoritmikus javítás lehetővé tette, hogy standard PC-n futó programok 1992 óta minden Számítógépes Sakkvilágbajnokságot megnyerjenek, sokszor 1000-szer több csomópont keresésére is képes masszíván parallel ellenségeket is legyőzve. A nyelészeti heurisztikák egész választékának használatával az effektív elágazási tényezőt kevesebb mint 3-ra lehet csökkenteni (a tényleges 35-ös elágazási tényezőhöz viszonyítva). Ezek közül legfontosabb a **nulla lépés (null move)** heurisztika, amely sekély keresést alkalmazva – ahol az ellenség az indulásnál kétszer is léphet – az állás értékének jó alsó becslését adja. Ez az alsó korlát sokszor lehetővé teszi az alfa-béta nyelést a teljes mélységű keresés költsége nélkül. Fontos még a **hatástarlanság nyelése (futility pruning)**, amely segíti eldönteni, hogy mely lépések vezetnek báta nyeléshez a követő csomópontok szintjén.

A Deep Blue csapata a Kaszparovval való visszavágó alól kibújt. Helyette 2002-ben a fő mérkőzés a FRITZ program és Vlagyimir Kramnyik világbajnok összecsapása volt, amely nyolc játszmában döntetlennel végződött. A mérkőzés feltételei az emberi játékosnak sokkal jobban kedveztek, és a hardver egy közönséges PC volt, nem egy szuper-számítógép. Kramnyik véleménye szerint mégis „most már világos, hogy a csúcsprogramok és a világbajnok közel azonos szinten vannak”.

## Dámajáték

Az IBM-nél dolgozó Arthur Samuel szabad idejében 1952-től kezdve kifejlesztett egy dámaprogramot, mely a kiértékelő függvényét több ezreszer saját ellen játszva maga tanulta meg. Ezt az ötletet a 21. fejezetben tárgyaljuk részletesebben. Samuel programja egy kezdő játékos szintjén kezdtet játszani, de miután néhány napig önmagával játszott, már Samuel játékszintjét is felülmúlt (bár Samuel nem számított erős játékosnak). 1962-ben a program Robert Nealyt – a „vak dámajáték” bajnokát is megverte, egy hibás lépése miatt. Sokan azt hitték, hogy ezzel bizonyítást nyert, hogy dámajátékban a gép erősebb, mint az ember, azonban ez így nem igaz. Ha viszont figyelembe vesszük, hogy Samuel számítógépe (egy IBM 704-es) 10 000 szavas memóriával, háttértárként mágnesszalagos egységgel és majdnem egy milliszekundumos ciklusidővel rendelkezett, ez a győzelem az MI nagy eredményeinek egyike marad.

Kevesen próbálták meg túlszárnyalni ezt a teljesítményt, amíg Jonathan Schaeffer és kollégái ki nem fejlesztették a Chinookot, amely közönséges PC-n futott és alfa béta keresést alkalmazott. A Chinook egy olyan előre létrehozott adatbázissal dolgozott, amely az összes nyolc vagy kevesebb bábút tartalmazó 444 milliárd állásból épült fel, hogy a végjátéka hibátlan legyen. 1990-ben az U.S. Openen a Chinook másodikként futott be és megnyerte a jogot, hogy a világbajnoki címért mérkőzzön. Ekkor a program Marion Tinsley képében egy problémába ütközött. Dr. Tinsley már 40 éve világbajnok volt és ez alatt a 40 év alatt összesen csak három játszmát vesztett. A Chinookkal vívott első mérkőzésén elszenvedte a negyedik, majd az ötödik vereségét is, de a bajnokságot 20,5:18,5 pontra megnyerte. 1994 augusztusában a Tinsley és a Chinook közötti világbajnoki mérkőzés idő előtt félbeszakadt, amikor Tinsleynek egészségügyi problémák miatt vissza kellett vonulnia. A Chinook lett a hivatalos világbajnok.

Schaeffer úgy gondolja, hogy megfelelő számítási teljesítménnyel a végjátékok adatbázisát annyira ki lehetne bővíteni, hogy a kezdeti pozíciótól induló előre keresés valamelyik már megoldott pozíciót mindig el tudná érni, vagyis a dámajáték teljesen megoldható lenne (volt, hogy a Chinook már az 5-ik lépésnél győzött). Az ilyen kimerítő elemzést kézi módszerekkel a  $3 \times 3$ -as amőbára meg lehet tenni és számítógépen a  $4 \times 4 \times 4$ -es amőbára (Qubic), a Go-Mokura (öt egy sorban) és a Nine-Men's Morrisra is elvégezték (Gasser, 1998). Ken Thompson és Lewis Stiller figyelemre méltó munkája (Thompson és Stiller, 1992) az összes öt- és hatfigurás sakkvégjátszmát is megoldotta, az eredményeket az interneten hozzáférhetővé téve. Stiller egy olyan kényszermatt esetét is felfedezte, amely 262 lépést igényelt. Ez egy kis felfordulást okozott, mert a sakk szabályai megkövetelik, hogy 50 lépés alatt valamilyen „előrehaladásnak” kell történnie.

## Othello

Az Othello vagy más néven Reversi számítógépes játékként talán népszerűbb, mint táblajátékként. Keresési tere kisebb, mint a sakké, általában 5-15 megengedett lépés van, de a kiértékelő szaktudást a semmiből indulva kellett kifejleszteni. 1997-ben a Logistello program (Buro, 2002) az emberi világbanokot, Takeshi Murakamit, képes volt hat nullára megverni. Általánosságban elfogadott, hogy az Othellóban az emberek nem elmenfelei a számítógépeknek.

## Ostábla

A 6.5. alfejezetben elmagyaráztuk, hogy a kockadobásból eredő bizonytalanság a mély keresést drága luxussá teszi. Az ostáblára vonatkozó kutatások zöme a kiértékelő függvény javítására összpontosult. Gerry Tesauro (Tesauro, 1992) neurális hálós technikákkal (lásd 20. fejezet) ötvözte Samuel megerősítéses tanulási módszerét, hogy egy figyelemre méltó kiértékelő függvényt hozzon létre, amit 2-es, illetve 3-as mélységű kereséssel együtt használt. A saját magával játszott több mint egy millió tanító játszma után Tesauro programja, a TD-GAMMON a világranglistán stabilan az első három játékos között foglal helyet. A program eredményei alapján a játék nyitó lépései tekintve egyes esetekben az eddigi gyakorlat radikálisan megváltozott.

## Gó

Ázsiában a gó a legnepszerűbb táblajáték, ami a mesterektől legalább olyan felkészültséget igényel, mint a sakk. A  $19 \times 19$ -es tábla miatt az elágazási tényező 361-gel indul, így a hagyományos keresési algoritmusok teljesen használhatatlanok. Egészen 1997-ig kompetens programoknak híre-hamva sem volt, most azonban már programok is képesek elfogadható lépésekkel tenni. A legjobb programok többsége alakzatfelismerési technikákat (amikor adott alakzat előfordul, akkor bizonyos lépés megtételét kell megfontolni) korláatos kereséssel (annak eldöntésére, hogy bizonyos köveket el lehet-e fogni, egy lokális területen maradva) kombinál. A könyv írásának idején a legerősebb programok valószínűleg Chen Zhixing Goemate és Michael Reiss Go4++ programja voltak, mindegyiket kb. 10 kyu-ra (gyenge amatőrszint) értékeltek. A gó olyan területeknek tűnik, ami a kifinomult következetési módszereket alkalmazó intenzív kutatásokból valószínűleg nyerhet. Sikert hozhat például az, hogy megtaláljuk annak a módját, hogy sok, lazán csatolt „részjátszmára” (amire a gó dekomponálható) vonatkozó lokális következtetés néhány szálát hogyan kellene integrálni. Az ilyen technikák az intelligens rendszerek szempontjából hihetetlen értéket képviselnek.

## Bridzs

A bridzs egy nem tökéletes információjú játék – az egyik játékos kártyái a többi játékos elől el vannak rejtve. A bridzs többszemélyes játék is egyben, négy játékossal kettő

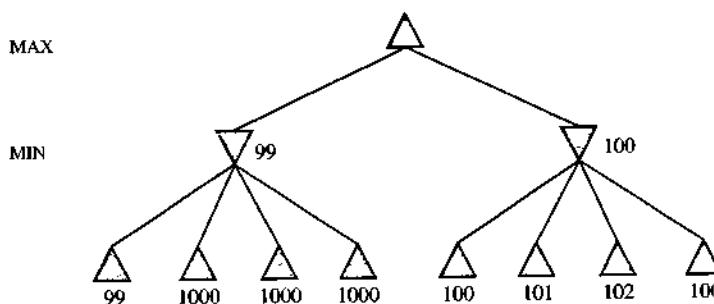
helyett, bár a játékosok két csapatot alkotnak, párban. Ahogyan ezt a 6.5. alfejezetben láttuk, egy bridzsparti optimális lejátszása az információgyűjtés, a kommunikáció, a blöffölés és a valószínűségek gondos mérlegelésének az elemeit is tartalmazhatja. E technikából számos elemet használ a Bridge Baron™ program (Smith és társai, 1998), amely megnyerte az 1997-es Számítógépes Bridzsbajnokságot. Bár nem játszik optimálisan, a Bridge Baron azon ritka sikeres, játékot játszó programok egyike, amelyek hierarchikus tervkészítési technikákat alkalmaznak (Jásd 12. fejezet), olyan magas szintű, bridzsjátékosok számára ismerős ötleteket felhasználva, mint az **impasszolás** (*finessing*) és a **dobáskényszer** (*squeezing*).

A GIB program (Ginsberg, 1999) meggyőző félénnnyel nyerte meg a 2000. évi bajnokságot. A GIB a „jövőbe látás szerinti átlagolás” módszerét használja, két lényegi módosítással. Először, ahelyett hogy megvizsgálná, hogy egy adott választás mennyire bizonyul jónak a rejtejt kártyák minden lehetséges kombinációja mellett (amiből 10 millió is lehet), a program egy véletlenszerűen választott 100 kombinációból álló mintát vizsgál. Másodszor, a GIB **magyarázatalapú általánosítást** (*explanation-based generalization*) használ arra, hogy különféle standard helyzetekre az optimális játékvezetés általános szabályait kiszámla és eltárolja. Ez lehetővé teszi minden leosztás *egzakt* megoldását. A GIB taktikai pontossága ellenállózza azt, hogy az információra következtetni nem tud. Az 1998-as emberi világbajnokságon az egyenlő rangú mérkőzésen (egy leosztás lejátszásában) 35-ből 12-ikként végzett, az emberi szakértők elvárasait messze túlszárnyalva.

## 6.7. ÉRTÉKELÉS

Mivel a játékokban az optimális döntés kiszámítása a legtöbb esetben kezelhetetlen, ezért az összes algoritmusnak valamilyen feltételezéssel és közelítéssel kell elnie. A minimax algoritmuson, kiértékelő függvényeken és az alfa-béta algoritmuson alapuló standard megközelítés csak egy lehetséges megoldás. Valószínűleg a standard megközelítés azért emelkedik ki a versenyjátékokra alkalmazott többi módszer közül, mert korán vetették fel és intenzíven fejlesztették. A terület néhány szakértője úgy gondolja, hogy a játékok épp ezért váltak el az MI fő irányvonalaítól, mert a standard megközelítés már nem hagy elegendő teret ahhoz, hogy új betekintést nyerjünk a döntéshozatal általános kérdéseibe. Ebben a részben az alternatív megközelítésekre fogunk rápillantani.

Először tekintsük a minimax algoritmust. A minimax optimális módszert ad egy lépés kiválasztásához egy adott keresési fából, *feltéve, hogy a levélcsomópontok kiértékelései tökéletesen pontosak*. A valóságban azonban a kiértékelések az állás értékének durva becslői, és úgy vehetjük, hogy nagy hibával rendelkeznek. A 6.13. ábra egy olyan egylépésváltásos játékfát mutat, amire a minimax alkalmatlannak tűnik. A minimax a jobb oldali ágat javasolja, holott nagyon valószínű, hogy a bal oldali ág valódi értékei nagyobbak. A minimax azon a feltételezésen alapul, hogy a 100-as, a 101-es, a 102-es és a 100-as címkét viselő csomópontok *mindegyike valóban* jobb, mint a 99-es címkét viselő csomópont. Az a tény azonban, hogy a 99-es címkéjű csomópontnak 1000-es címkéjű testvérei vannak, azt sugallja, hogy a csomópont valódi értéke valójában magasabb. Ezen probléma kezelésének egyik lehetséges módja, ha a kiértékelés egy, a lehetséges értékek feletti *valószínűség-elosztást* ad vissza. Ekkor standard statisztikai



6.13. ábra. Egy olyan egylépés-váltásos játékfa, amire a minimax alkalmatlannak tűnik

módszerekkel ki lehet számítani a szülő értékének valószínűség-eloszlását. Sajnos a testvércsomópontok értékei általában erősen korreláltak, ezért ez költséges számítás lehet, mert olyan részletes ismeretre van szükség, amit általában nehéz beszerezni.

A következőkben tekintsük a fát generáló kereső algoritmust. Az algoritmustervező célja, hogy olyan algoritmust adjon meg, ami megfelelő idő alatt lefut, és jó lépést választ. Az alfa-béta algoritmus legszembetűnöbb problémája, hogy nem arra terveztek, hogy egyszerűen kiválasszon egy jó lépést, hanem arra, hogy értékkorlátokat számítson ki az összes megengedett lépés értékére. Hogy lássuk, ez a többletinformáció miért szükségtelen, tekintsünk egy állást, ahol egyetlen megengedett lépés létezik. Ettől függetlenül az alfa-béta keresés egy hatalmas és teljesen haszontalan keresési fát generál és értékel ki. Természetesen beiktathatunk egy tesztet az algoritmusba, de ezzel csak elrejtjük a problémát – az alfa-béta algoritmus által elvégzett számítások nagy része egyáltalán nem releváns. Az az eset, amikor csak egyetlen megengedett lépés létezik, nem sokban tér el attól az esettől, amikor néhány megengedett lépés létezik, ami közül egy jó, a többi pedig nyilvánvalóan katasztrofális. Egy ilyen „egyértelműen favorit helyzetben” szerencsesebb lenne egy kis keresés után gyors döntésre jutni, mintsem az időt fecsérálni, amit később egy problémásabb helyzetben esetleg hasznosabban fel lehetne használni. Ez elvezet a csomópontkifejtés hasznosságának gondolatához. Egy jó keresési algoritmusnak nagy hasznosságú csomópontot kell kiválasztania kifejtésre, vagyis olyat, ami valószínűleg egy lényegesen jobb lépés felfedezéséhez vezet. Ha nincs olyan csomópontkifejtés, aminek a hasznossága nagyobb, mint a kifejtésének (időre vonatkoztatott) költsége, akkor az algoritmusnak be kell fejeznie a keresést, és lépnie kell. Vegyük észre, hogy ez nemcsak az egyértelmű favorithelyzetekre működik, hanem a szimmetrikus lépésekre is, ahol bármennyi keresést is végeznénk, egyik lépés sem mutatkozna jobbnak.

Azt a fajta következtetést, amikor megmondjuk, hogy milyen számításokat kell végrehajtani, metakövetkeztetésnek (*metareasoning*) nevezik (következtetés a következetetőről). Ez nemcsak a játékokra vonatkozik, hanem mindenfajta következtetésre is. minden számítást azért végzünk el, hogy jobb döntéseket hozhassunk. Mindegyiknek van valamilyen költsége, és mindegyik valamilyen valószínűsséggel bizonyos javulást eredményezhet a döntés minőségében. Az alfa-béta algoritmus a lehető legegyszerűbb metakövetkeztetést foglalja magában: olyan tételek, miszerint a fa bizonyos ágait veszteség nélkül figyelmen kívül lehet hagyni. Ennél sokkal jobbat is lehet tenni. A 16. fejezetben megnézzük, hogy ezeket a gondolatokat hogyan lehet pontossá és megvalósíthatóvá tenni.

Végezetül ismét vizsgáljuk meg magának a keresésnek a természetét. A heurisztikus keresés és a játékok algoritmusai úgy működnek, hogy a kiinduló állapotból induló konkrét állapotsorozatokat generálnak, majd alkalmaznak egy kiértékelő függvényt. Az emberek nem így játszanak. A sakkban gyakran van egy konkrét célunk – például le akarjuk ütni az ellenfél vezérét –, és ezt a célt használhatjuk arra, hogy a cél elérése érdekében **szelektíven** megfelelő terveket generálunk. Ez a fajta **célvezérelt következtetés** (**goal-directed reasoning**) vagy **tervkészítés** (**planning**) néha teljesen eltünteti a kombinatorikus keresést (lásd IV. rész). David Wilkins PARADISE programja (Wilkins, 1980) az egyetlen olyan program, amely sikeresen alkalmazta a célvezérelt következtetést a sakorra: néhány 18 lépésből álló kombinációt igénylő sakkfeladványt is képes volt megoldani. Ez idáig azonban még nem látjuk tisztán, hogyan kellene a két algoritmustípust egyetlen robosztus és hatékony rendszerbe ötvözni, bár a Bridge Baron™ jelenthet egy, a helyes irányban tett lépést. Egy ilyen teljesen integrált rendszer jelentős eredménynek számítana nemcsak a játtekkutatás területén, hanem általánosságban véve az MI-kutatás területén is, mert egy általános intelligens ágens számára megfelelő alapot is tudna nyújtani.

## 6.8. ÖSSZEFOLGLALÁS

A játékok sokaságát néztük meg, hogy megértsük, mit is jelent optimálisan és a gyakorlatban is jól játszani. A legfontosabb gondolatokat az alábbiakban foglalhatjuk össze:

- Egy játékot a **kiinduló állappittal** (**initial state**) (a táblaállással), minden egyik állapotban a **legális cselekvésekkel** (**actions**), egy végeszettel (**terminal test**) (ami megmondja, hogy mikor ért véget a játék) és egy **hasznossági függvénnyel** (**utility function**) (ami megmondja, hogy ki és mennyivel nyert) lehet megadni.
- **Tökéletes információval** (**perfect information**) rendelkező kétszemélyes zérusösszegű játékoknál a **minimax** algoritmus a teljes játékfa mélységi felsorolásával meg tudja határozni a játékos legjobb lépését.
- Az **alfa-béta** algoritmus ugyanazt a számítást végzi el, mint a minimax algoritmus, azonban jóval hatékonyabb, mivel lenyeli a keresési fa azon ágait, amiről be tudja bizonyítani, hogy a végső eredmény szempontjából irrelevánsak.
- A teljes játékfa általában nem kezelhető (még az alfa-béta algoritmussal sem), ezért a keresést valahol abba kell hagynunk, és egy **kiértékelő függvényt** (**evaluation function**) kell alkalmaznunk, ami az adott állapot hasznosságának becslójét adja.
- A véletlen elemet is tartalmazó játékokat a minimax algoritmus olyan kiterjesztéssel lehet kezelni, amely a **véletlen csomópontokat** (**chance node**) úgy értékeli ki, hogy az egyes hasznosságokat a gyermekcsomópontok valószínűségével súlyozva veszi az összes gyermekcsomópontjának az átlagos hasznosságát.
- A **nem tökéletes információjú játékok** (**imperfect information**), mint például a bridzs, optimális lejátszása minden egyik játékestől az aktuális és a jövőbeli **hiedelmi állapotaira** (**belief states**) vonatkozó következtetést igényli. Egy egyszerű közelítés kapható, ha átlagoljuk egy cselekvés értékét a hiányos információ minden lehetséges konfigurációjára.
- A programok egyenlő ellenfelek, vagy akár meg is verik a legjobb emberi játékosokat dámajátékban, Othellóban és ostáblajátékban, és igen közel állnak ehhez a bridzsben. Egy program legyőzte egy kirakatmérkőzésen az emberi sakkvilágbanokot. A góban a programok még amatőr szinten játszanak.

## Irodalmi és történeti megjegyzések

A mechanikus játékok korai történetére számos csalás rányomta a békéget. Ezek közül a legnevezetesebb Kempelen Farkas báró 1769-ben kiállított „Törökje”, egy feltételezett sakkautomata, amely Napóleont is megverte, mielőtt kiderült, hogy a szekrénye valójában egy törpe növésű emberi sakkmestert rejttet (Levitt, 2000). A Török 1769-től 1854-ig játszott. Úgy tűnik, Charles Babbage (aki a Török hatása alatt állt) volt az első, aki 1846-ban számítógépes sakkkészítést megvalósíthatóságának első komoly elemzését adta (Morrison és Morrison, 1961). Tervezett egy  $3 \times 3$ -as amóbát játszó célgépet is, amit soha sem épített meg. Az első igazi játékgépet 1890 táján egy spanyol mérnök, Leonardo Torres y Quevedo tervezte és építette meg. A „KRK” a sakkvégjátékról specializált gép volt (király és bánya a király ellen) és képes volt bármilyen kiinduló állásból mattot adni.

A minimax algoritmust sokszor Ernst Zermelo, a modern halmaelmélet atya 1912-es cikkéhez vezetik vissza. A cikk sajnálatos módon tartalmaz hibákat, és a minimaxot nem írja le helyesen. A játékelmélet komoly alapjait a nagy hatású *Theory of Games and Economic Behavior* c. munkában Neumann és Morgenstern (Neumann és Morgenstern, 1944) fektették le, kimutatva azt is, hogy egyes játékokban szükség van randomizált (avagy nem megjósolható) stratégiákra. (További információért lásd 17. fejezet.)

A korai számítógépes korszak számos prominens személyét kíváncsivá tette a számítógépes sakkképessége. Konrad Zuse – aki elsőként tervezett programozható számítógépet – igen részletes ötleteket dolgozott ki arra, hogy ezt hogyan lehetne megvalósítani (Zuse, 1945). Norbert Wiener nagy befolyású *Cybernetics* c. könyve (Wiener, 1948) tartalmazta egy számítógépes sakkkódásnak egy lehetséges vázlatát, a minimax keresést, a mélységi levágást és a kiértékelő függvényt is beleértve. Claude Shannon a modern számítógépes játékok elvi alapjait sokkal részletesebben fejtette ki, mint Wiener (Shannon, 1950). Shannon bevezette az egyensúlyi állás fogalmát, és néhány ötletet vázolt fel a szelektív (nem kimerítő) játékfakeresésre vonatkozólag. Slater és az ugyanabban a kötetben a cikkére reflektáló szerzők szintén megvizsgálták a számítógépes sakkozás lehetőségeit (Slater, 1950). I. J. Good Shannontól függetlenül kidolgozta az egyensúlyi állás fogalmát (Good, 1950).

1951-ben Alan Turing írta meg az első valódi számítógépes programot, ami képes volt egy teljes sakkjátszmát lejátszani (Turing és társai, 1953). Valójában azonban Turing programja sohasem futott számítógépen, kézi szimulációval tesztelték egy nagyon gyenge emberi sakkjátékos ellen, aki legyőzte a programot. Időközben D. G. Prinz megírt és valóban futtatott is egy programot (Prinz, 1952), ami sakkkeladványokat oldott meg, bár nem játszott teljes játszmát. Alex Bernstein írta az első olyan sakkkódásnak, ami egy teljes standard sakkjátszmát játszott (Bernstein és Roberts, 1958; Bernstein és társai, 1958).<sup>6</sup>

Az alfa-béta keresés alapötletét John McCarthy dolgozta ki 1956-ban, habár nem publikálta. Az NSS-sakkkódásnak az alfa-béta algoritmus egy leegyszerűsített változata használta, ez volt az első sakkkódás, ami ezt alkalmazta (Newell és társai, 1958). Nilsson szerint Arthur Samuel díjakódásnak (Samuel, 1959; 1967) szintén alfa-béta

<sup>6</sup> Newell (Newell és társai, 1958) egy orosz BESM programot említi, amely lehet, hogy Bernstein programját me előzte.

algoritmust használt, habár maga Samuel ezt nem említi a rendszerről publikált beszámolóiban (Nilsson, 1971). Az 1960-as évek elején jelentek meg az alfa-béta algoritmust ismertető cikkek (Hart és Edwards, 1961; Brudno, 1963; Slagle, 1963b). Az alfa-béta algoritmus egy teljes implementációját Slagle írta le egy cikkben (Slagle és Dixon, 1969), ami a kalah<sup>7</sup> játékok játszó játékprogram működését ismertette. A John McCarthy egyik diáka által írt „Kotok–McCarthy” sakkprogram (Kotok, 1962) is az alfa-béta algoritmust használta. Knuth ismerteti az alfa-béta algoritmus történetét (Knuth és Moore, 1975), megadja az algoritmus teljességének a bizonyítását és elvégzi az időigény elemzését. Knuth és Moore elemzése az alfa-béta keresésnek a követők véletlen sorba rendezésével  $O((b/\log b)^d)$  aszimptotikus komplexitást mutatott ki, ami lehangoló eredmény, mert a  $b/\log b$  effektív elágazási tényező magánál a  $b$ -nél nem sokkal jobb. Később jöttetek rá, hogy az aszimptotikus képlet csak a  $b > 1000$  igaz, és az aktuális játékokra található elágazási tényezőkre a gyakran idézett  $O(b^{3d/4})$  érvényes. Pearl megmutatta (Pearl, 1982b), hogy az alfa-béta algoritmus aszimptotikusan optimális az összes rögzített mélységű játékfa-keresési algoritmus között.

Az első két sakkprogram, ami egymás ellen játszott, a Kotok–McCarthy-program és a Moszkvai Elméleti és Kísérleti Fizika Intézet által megírt „ITEP” program (Adelson-Velsky és társai, 1970) volt az 1960-as évek közepén. Ezt az interkontinentális mérkőzést távirón játszották le. A küzdelem 1967-ben az ITEP-program 3:1 arányú győzelmével ért véget. A MacHack 6 volt az első olyan sakkprogram, ami sikeresen játszott emberek ellen (Greenblatt és társai, 1967). 1400-as Élő-pontszáma jóval több volt, mint a kezdők 1000-es Élő-pontja, azonban így is igen messze volt a 2800 vagy több Élő-ponttól, ami szükséges lett volna, hogy Herb Simon 1957-es jóslata teljesüljön, miszerint 10 éven belül a számítógépes sakkprogramok lesznek a sakkvilágbaajnokok (Simon és Newell, 1958).

Az 1970-es első ACM Észak-Amerikai Számítógépes Sakkbaajnoksággal a sakkprogramok versengése komolytá vált. A korai 1970-es évek programjai igen bonyolultak voltak, számos trükköt vetettek be, hogy a keresés bizonyos ágait levágják, elfogadható lépéseket generáljanak stb. Az első Számítógépes Sakkvilágbaajnokságot 1974-ben Stockholm-ban rendezték. Ezt az első világbaajnokságot a Kaissa (Adelson-Velsky és társai, 1975), egy másik ITEP-program nyerte meg. Kaissa egy egyszerűbb megközelítésen alapult és kimerítő alfa-béta keresést használt egyensúlyi kereséssel vegyítve. A megközelítés felőffbrendűségét igazolta a CHESS 4.6 győzelme az 1977-es Számítógépes Sakkvilágbaajnokságon. A CHESS 4.6 lépéseként 400 000 állást elemzett és 1900 Élő-pontot ért el.

Greenblatt MacHack 6-osának egy későbbi változata volt az első olyan sakkprogram, amely már kifejezetten a sakkrá tervezett célhardveren futott (Moussouris és társai, 1979), de a Belle (Condon és Thompson, 1982) volt az első olyan program, ami a célhardvernek köszönhetően jelentős sikereket ért el. A Belle lépésgeneráló és állásértékelő hardvere lehetővé tette, hogy lépéseként néhány millió állást is elemezzen. A Belle 2250 Élő-pontot ért el, és az első mesteri fokozat szintű program lett. A Hitech rendszer egy speciális rendeltetésű számítógép volt, amit Hans Berliner, a korábbi levelező sakkvilágbaajnok és egy CMU-beli diáka, Carl Ebeling tervezett, hogy a kiértékelő függvények gyors számítását tegyék lehetővé (Ebeling, 1987; Berliner és Ebeling, 1989).

<sup>7</sup> A kalah egy afrikai eredetű kétszemélyes játék, amelyben kavicsokat (gyémántokat) kell átrakosgnati csészék között. Több számítógépes változata is létezik, l. például: <http://ceeman.ecn.purdue.edu/~ee373/kalah.html>. (A ford.)

A HITECH észak-amerikai sakkbajnok lett 1985-ben, és 1987-ben az első olyan program volt, amely egy emberi nagymestert is legyőzött. A CMU-n szintén kifejlesztett Deep Thought (Hsu és társai, 1990) a tiszta keresés sebességét tovább fokozta. 2551 Elő-pontot ért el és a Deep Blue előfutára lett. Az 1980-ban alapított Fredkin-díj 5000 dollárt ajánlott fel annak a sakkprogramnak, amelyik elsőként éri el a mesterfokozatot, és 10 000 dollárt ajánlott fel annak a sakkprogramnak, amely elsőként éri el a USCF (Amerikai Egyesült Államok Sakkszövetsége) 2500-as Elő-pontot (ez a nagymesteri szinthez közeli érték), és 100 000 dollárt ajánlott fel azon sakkprogramnak, amelyik elsőként legyőz egy emberi sakkvilágbajnokot. Az 5000 dolláros díjat 1993-ban a Belle, a 10 000 dolláros díjat 1989-ban a Deep Thought, majd a 100000 dolláros díjat 1997-ben a Deep Blue nyerte el a Kaszparov felett aratott győzelemért. Fontos emlékezni, hogy a Deep Blue sikerét mind az algoritmikus javítások, mind a hardver biztosította (Hsu, 1999; Campbell és társai, 2002). Az olyan technikák, mint a nulla lépés heurisztika (Beal, 1990) a keresésben igen szelektív programokhoz vezettek. Az utolsó három Számítógépes Sakkvilágbajnokságot 1992-ben, 1995-ben és 1999-ben a standard PC-n futó programok nyerték meg. Egy korszerű sakkprogramnak talán a legrészletesebb leírását Ernst Heinz adja meg (Heinz, 2000), amelynek DARKTHOUGHT programja volt a legmagasabb pontszámú nem kereskedelmi program az 1999-es világbajnokságon.

Kísérletek történtek arra, hogy a „standard megközelítés” 6.7. alfejezetben leírt problémáit leküzdjék. Az első szelektív kereső algoritmus, elméleti igénnyel, valószínűleg a B\* algoritmus (Berliner, 1979) volt, ami megkíséri, hogy a játékfa csomópontjainak az értékeire egyetlen becsült érték helyett intervallumkorlátokat adjon. A levélcsomópontok kifejtése annak érdekében történik, hogy a legfelső szintű korlátokat finomítsuk, amíg egy „nyilván legjobb” lépést nem találunk. Palay az alfa-béta algoritmus becsült értékei, illetve a B\* algoritmus intervallumai helyett valószínűség-eloszlásokat használ (Palay, 1985). David McAllester konspirációs szám keresési algoritmusára azokat a levélcsomópontokat fejti ki, amelyek, ha megváltoznának az értékeik, előidéznék, hogy az algoritmus a gyökérnél új lépést válasszon (David McAllester, 1988). Az MGSS\* (Russell és Wefald, 1989) a 16. fejezet fejlett döntéselméleti technikáit használja minden levélcsomópont kifejtésének értékbecslésére a gyökérszintű döntés minőségében tapasztalt várható javulás függvényében. Ez a program az Othelloban képes volt jobb eredményeket elérni, mint az alfa-béta algoritmus, annak ellenére, hogy egy nagysághorddel kevesebb csomópontot vizsgált meg. Az MGSS\* megközelítés elvben alkalmas a következetés bármilyen formájának vezérlésére.

Az alfa-béta keresés több szempontból a mélységi áglenyessé megfelelője két játékos esetén, amit az egyedi ágens esetében az A\* dominál. Az SSS\* algoritmus (Stockman, 1979) egy kétszemélyes A\*-nak tekinthető, és azonos döntés eléréséhez soha nem fejt ki több csomópontot, mint az alfa-béta algoritmus. Eredeti formájában az SSS\* memóriaigénye és a sorba állítás számítási overheadje miatt nem praktikus, azonban az RLEK algoritmusból kifejlesztettek egy lineáris tárkomplexitású változatot (Korf és Chickering, 1996). Plat az SSS\*-ja egy új megközelítést dolgozott ki (Plat és társai, 1996) az alfa-béta keresés és a transzpozíciós táblák együtteseként, és megmutatta, hogy az eredeti algoritmus problémáit hogyan kell elkerülni, végül MTD(f) néven egy új változatot fejlesztett ki, amit sok kiemelkedő teljesítményű programba építettek be.

D. F. Beal és Dana Nau a minimax közelítő kiértékelésekre történő alkalmazásoknál jelentkező gyengeségeit tanulmányozták (Beal, 1980; Dana Nau, 1980; 1983). Kimutatták,

hogy a levélértékeknek a fában való eloszlására bizonyos függetlenségi feltételezésekkel elve, a minimax algoritmus kevésbé megbízható becsléseket ad a gyökérre, mintha a ki-értékelő függvényeket közvetlenül, mindenféle keresés nélkül alkalmaznánk. Pearl *Heuristics* c. könyve (Pearl, 1984) ezt a látszólagos paradoxont részben megmagyarázza és számos játékalgoritmust elemez. Baum és Smith (Baum és Smith, 1997) a minimax egy valószínűség-alapú helyettesítését javasolja, és azt mutatja ki, hogy ez bizonyos játékokban jobb döntésekhez vezet. A keresés különféle szinteken való levágásából és a kiértékelő függvény alkalmazásából adódó hatások elmélete még mindig szegényes.

A várhatóminimax algoritmust Donald Michie vetette fel (Michie, 1966), habár az közvetlenül következik Neumann és Morgenstern játékfa-kiértékelési elméletéből. Bruce Ballard az alfa-béta nyelést kiterjesztette a véletlen csomópontokat is tartalmazó fákra (Bruce Ballard, 1983). Az első sikeres ostáblaprogram a BKG volt (Berliner, 1977; 1980b). Csak 1 mélysegig keresett és bonyolult, manuálisan összeállított kiértékelő függvényt használt. Ez volt az első számítógépes program, ami képes volt legyőzni egy emberi világbajnokot az ismert klasszikus táblajátékok egyikében (Berliner, 1980a), habár Berliner elsőként ismerte el, hogy ez csak egy rövid, bemutató mérkőzés volt (nem egy világbajnoki mérkőzés), és hogy a BKG nagyon szerencsés „kézzel” dobott a kockával. Gerry Tesauro kutatásai, először a NEUROGAMMON (Tesauro, 1989), majd a TD-GAMMON (Tesauro, 1995) programokkal, azt mutatták, hogy sokkal jobb eredmény érhető el a megerősítéses tanulással (amely területtel a 21. fejezetben foglalkozunk).

A dáma, és nem a sakk volt az első olyan klasszikus játék, amire egy valóban számítógépen futó program képes volt végigjátszani egy teljes játszmát. Christopher Strachey írta az első működőképes dámajátékprogramot (Strachey, 1952). Schaeffer (Schaeffer, 1997) nagyon olvasmányosan leírja a Chinook – a világbajnok dámajátékprogram – fejlesztéstörténetét az összes műhelytitkával együtt.

Az első góprogramokat valamivel a dáma- és a sakkprogramok után fejlesztették ki (Lefkovitz, 1960. Remus, 1962), és lassabban fejlődtek a dáma- és a sakkprogramoknál. Ryder tiszta keresésalapú megközelítést használt (Ryder, 1971) a selektív nyelési módszerek egész palettájával, hogy le tudja győzni a hatalmas elágazási tényezőből adódó problémát. Zobrist feltétel-cselekvés szabályokat használt elfogadható lépések generálására, ha a játékban ismert alakzatok jelentek meg (Zobrist, 1970). Reitman és Wilcox jó eredménnyel kombinálta a szabályokat és a keresést (Reitman és Wilcox, 1979), a legtöbb modern program követte ezt a hibrid megközelítést. A számítógépes gó jelenlegi állását Müller (Müller, 2002) foglalja össze, és tömördekk hivatkozást is közöl. Anshelevich (Anshelevich, 2000) hasonló módszereket a Hex játéakra alkalmaz. A friss fejleményekről a Számítógépes Go Szövetség által kiadott *Computer Go Newsletter* számol be.

A számítógépes játékokról több fórumon jelennek meg cikkek. A felettesebb félrevezető nevű konferenciakiadvány, a *Heuristic Programming in Artificial Intelligence* számol be a Számítógépes Sakkolimpiákról, amely rendezvények a játékok egész sorára terjednek ki. A játékprogramok kutatásáról fontos cikkeket publikáltak számos szerkesztett cikkgyűjteményben (Levy, 1988a; Levy, 1988a; Marsland és Schaeffer, 1990). Az 1977-ben alapított Nemzetközi Számítógépes Sakkszövetség (International Computer Chess Association, ICCA) negyedévenként adja ki az *JCGA Journal*t (korábban *ICCA Journal*). (Clarke, 1977) óta fontos cikkek jelentek meg az *Advances in Computer*

*Chess* antológiasorozatban. Az *Artificial Intelligence* folyóirat 134. kötete (*Artificial Intelligence*, 2002) leírásokat tartalmaz a legfejlettebb sakk, Othello, Hex, shogi, gő, ostábla, póker, Scrabble™ és más játékprogramokról.

## Feladatok

- 6.1.** Ez a feladat a játékok alapvető elveit a  $3 \times 3$ -as amőbán (körök és ikszek) keretből gyakorolatja be.  $X_n$  azon sorok, oszlopok vagy átlók számát jelöli, ahol pontosan  $n$  db  $X$ , míg egyetlen  $O$  sem található. Hasonlóan,  $O_n$  azon sorok, oszlopok vagy átlók számát jelöli, ahol pontosan  $n$  db  $O$ , míg egyetlen  $X$  sem található. A hasznosságfüggvény  $+1$ -et rendel minden olyan álláshoz, ahol  $X_3 = 1$ , és  $-1$ -et rendel minden olyan álláshoz, ahol  $O_3 = 1$ . Az összes többi végállás hasznossága 0. A nem végállapotok esetén az alábbi módon definiált lineáris kiértékelő függvényt fogjuk használni:

$$\text{Kiértékel}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

- (a) Körülbelül hány különböző  $3 \times 3$ -as amőbajátszmát lehet lejátszani?
- (b) Üres táblából kiindulva, a szimmetriát is figyelembe véve, 2-es mélységgel (vagyis egy  $X$  és egy  $O$  a táblán) mutassa meg a teljes játékfát.
- (c) A fában 2-es mélységen tüntesse fel az összes álláskiértékelést.
- (d) A fában az 1-es és a 0-s mélységen levő állásokhoz a minimax algoritmust felhasználva tüntesse fel a felfelé terjesztett értékeket, és használja fel azokat a legjobb kezdő lépés meghatározásához.
- (e) Karikázza be a 2-es szinten levő azon csomópontokat, amelyeket az alfa-béta nyesés alkalmazása esetén az algoritmus *nem* értékelne ki, feltételezve, hogy a csomópontokat az algoritmus *az alfa-béta nyeséshez optimális sorrendben generálja*.

- 6.2.** Bizonyítsa be az alábbi állítást: minden játékfa esetén a MAX által kapott hasznosság, amikor MAX a minimax döntést használva játszik egy szuboptimális MIN-nel szemben, sohasem lesz alacsonyabb, mint az a hasznosság, ami akkor ér el, ha egy optimális MIN-nel szemben játszana. Ki tud találni egy olyan játékfát, ahol MAX egy szuboptimális stratégiával egy szuboptimális MIN-nel szemben még jobban teljesítene?

- 6.3.** Tekintse a 6.14. ábrán látott kétszemélyes játékot.

- (a) Rajzolja fel a teljes játékfát az alábbi konvenciót alkalmazva:
  - minden állapotot ( $S_A$ ,  $S_B$ ) formában írja fel, ahol  $S_A$  és  $S_B$  a zsetonok helyzete.
  - minden végállapotot egy négyzetbe, a játékértékét pedig egy körbe írja be.
  - a *hurokállapotokat* (azon állapotok, amelyek már megjelentek a gyökkérig vezető út mentén) dupla négyzetekbe írja be. Mivel nem világos, hogy az ilyen állapotok értéke mennyi, jelölje be mindegyiket egy körbe írt „?”-lel.

- (b) Most mindegyik csomóponthoz írja be a visszaterjesztett minimax értékét (szintén egy körbe írva). Magyarázza meg, hogy a „?” értékeket hogyan kezelte, és miért?
- (c) Magyarázza meg, hogy a standard minimax ebben a játékfában miért fulladna kudarcba, és röviden vázolja fel, hogy a (b) kérdésre adott válaszra támashodva hogyan lenne képes az algoritmust megjavítani. Képes-e a módosított algoritmus megadni az optimális döntéseket az összes hurkokat tartalmazó játékban?
- (d) Ez a 4-négyzetes játék  $n$  négyzet esetére általánosítható, minden  $n > 2$ -re. Bizonyítsa be, hogy A győz, ha  $n$  páros, és veszít, ha  $n$  páratlan.



**6.14. ábra.** Egy egyszerű játék kiinduló állása. Az A játékos indul elsőnek. Mindkét játékos felváltva lép és a zsetonját a szomszédos szabad helyre helyezheti, minden irányban. Ha a szomszédos mező foglalt, akkor a játékos az ellenfél felett átugorhat a következő szabad helyre, ha van ilyen. (Például ha A a 3-n és B a 2-n van, akkor A visszaléphet 1-re.) A játéknak vége, ha az egyik játékos eléri a tábla ellentétes végét. Ha az A játékos elsőnek éri el a 4-et, a játék értéke +1, ha B játékos éri el elsőnek az 1-et, a játék értéke az A számára -1.

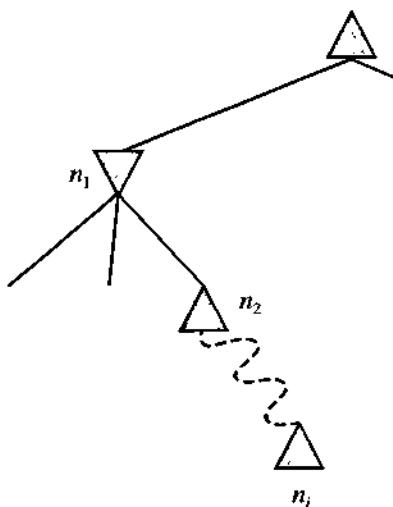
- 6.4.** Implementáljon lépésgenerátort és kiértékelő függvényt az alábbi játékok közül néhányat: kalah, Othello, dáma, sakk. Az implementációt használva tervezzen egy általános alfa-béta játékágenst. Hasonlítsa össze a keresési mélység növelésének, a lépéssorrendezés tökéletesítésének és a kiértékelő függvény tökéletesítésének a hatását. Milyen közel esik az Ön effektív elágazási tényezője a tökéletes lépéssorrendezés ideális esetéhez?

- 6.5.** Állítsa elő az alfa-béta nyesés helyességének formális bizonyítását. Ehhez tekintse a 6.15. ábrán bemutatott helyzetet. A kérdés az, hogy az algoritmus lenyesse-e az  $n_j$  csomópontot, ami egy max-csomópont és az  $n_1$  leszármazottja. Az alapötlet, hogy akkor és csak akkor nyessük le, ha  $n_1$  minimax értéke igazolhatóan független  $n_j$  értékétől.

- (a)  $n_1$  értékét az alábbi képlet adja meg:

$$n_1 = \min(n_2, n_{21}, \dots, n_{2b2})$$

- Adjon egy hasonló kifejezést  $n_2$ -re, és  $n_1$ -re,  $n_j$ -t felhasználva.
- (b) Legyen  $l_i$  az  $n_i$  csomóponttól balra levő, ismert minimax értékű,  $i$  mélységen található csomópontok minimális (maximális) csomópontértéke. Hasonlóan, legyen  $r_i$  az  $n_i$  csomóponttól jobbra levő, még ki nem fejtett,  $i$  mélységen található csomópontok minimális (maximális) csomópontértéke. Az  $n_1$ -re előbb meghatározott kifejezést írja át  $l_i$  és  $r_i$  értékekkel kifejezve.
- (c) Most fogalmazza át a kifejezést, hogy megmutassa, ahhoz, hogy  $n_j$  befolyásolja  $n_1$ -et,  $n_j$ -nek meg kell haladnia bizonyos, az  $l_i$  értékekből meghatározott korlátot.



**6.15. ábra.** Az a helyzet, amikor az algoritmus eldönti, hogy lenyesheti-e az  $n_j$  csomópontot

(d) Az eljárást ismételje meg arra az esetre, amikor az  $n_j$  egy min-csomópont.

- 6.6.** Implementálja a véletlen csomópontokat tartalmazó játékfák lenyelésére alkalmas várhatóminimax és  $^*$ -alfa-béta algoritmust, amit Ballard (Ballard, 1983) ír le. Próbálja ki azokat az algoritmusokat olyan játékokon, mint az ostábla, és mérje meg a  $^*$ -alfa-béta nyerési hatékonyását.
- 6.7.** Bizonyítsa be, hogy a levélcsomóponti értékek pozitív lineáris transzformációja (azaz egy transzformáció  $x$ -től  $ax + b$ -ig, ahol  $a > 0$ ), nincs befolyással a lépések választékára a játékfában, akkor sem, ha léteznek benne véletlen csomópontok.
- 6.8.** Tekintse az alábbi eljárást a lépések megválasztására a véletlen csomópontot is tartalmazó játékokban:
- Generáljon egy megfelelő számú (mondjuk 50) kockadobásból álló sorozatot egy megfelelő (mondjuk 8) mélységgig.
  - Ismert kockadobások esetén a játékfa determinisztikussá válik. minden egyes kockadobás-sorozatra oldja meg az eredményül kapott determinisztikus játékfát az alfa-béta algoritmussal.
  - Az eredményeket használja fel az egyes lépések értékének megbecsülésére, és válassza ki a legjobb lépést.
- Helyesen fog működni ez az eljárás? Miért (nem)?
- 6.9.** Írjon és implementáljon egy valós *idejű, többszemélyes* játszókörnyezetet, ahol az idő a környezeti állapot része, és a játékosok rögzített időszeleteket kapnak.
- 6.10.** Adja meg és/vagy implementálja az alábbi játékok egyikére vagy akár többre is az állapotleírást, lépésgenerálást és a kiértékelő függvényt: Monopoly, Játék a

betűkkel, bridzs (egy konkrét licitet feltételezve) és póker (válassza meg a kedvenc változatát).

- 6.11.** Gondosan tekintse át a 6.10. feladat minden egyes játékában a véletlen események és a részleges információk összefüggéseit.
- Melyikre lesz jó a standard várhatóminimax modell? Implementálja az algoritmust és futtassa le a játékágensben a játékkörnyezet szükséges módsításaival.
  - Melyekre lenne jó a 6.8. feladatban leírt séma?
  - Vitassa meg, hogyan kellene kezelni azt a tényt, hogy egyes játékokban a játékosok nem rendelkeznek ugyanazzal az információval az aktuális állapotról.
- 6.12.** A minimax algoritmus feltételezi, hogy a játékosak felváltva lépnek, az olyan kártyajátékokban azonban, mint a whist vagy a bridzs, minden a leütés győztese indul a következőnek.
- Módosítsa algoritmusát, hogy e játékokra is megfelelően működjön. Feltételezheti, hogy rendelkezésére áll a Győztes(*leütés*) függvény, amely azt adja vissza, hogy az adott leütést mely kártya nyerte meg.
  - Rajzolja fel a 228. oldalon látható első leosztásra a játékfát.
- 6.13.** A Chinook dátaprogram kiterjedten használja a végjáték adatbázisokat, amelyek pontos értéket adnak a játék utolsó hat lépésében előálló összes álláshoz. Hogyan lehet egy ilyen adatbázist hatékonyan generálni?
- 6.14.** Vitassa meg, hogyan alkalmazható a játékok standard megközelítése az olyan játékokra, mint például a tenisz, a póló és a krikett, amelyeket folytonos, fizikai állapottérben játszanak.
- 6.15.** Írja le, hogy a minimax és az alfa-béta hogyan változik kétszemélyes, nem zérusösszegű játékok esetén, ahol minden játékosnak külön kiértékelő függvénye van. Feltételezheti, hogy mindegyik játékos mások kiértékelő függvényeit ismeri. Ha a két véghasznosság értékére nincs korlát, lehetséges-e valamelyik csomópont számára, hogy az alfa-béta lenyési?
- 6.16.** Tegyük fel, hogy egy olyan sakkprogrammal rendelkezik, amely képes egymilliós csomópontot kiértékelni másodpercenként. Válassza a játékállás egy tömör reprezentációját a transzpozíciós tábla számára. Kb. mennyi állást képes eltárolni memóriában, az 500 Mb által nagyságú táblában? Elég lesz-e ez a lépésekkel allokált 3 perces kereséshez? Hány táblakiolvasást képes megvalósítani egy állandókiértékelés ideje alatt? Most tételezze fel, hogy a transzpozíciós tábla nagyobb, és nem fér el a memóriában. Hány kiértékelést lehetne megvalósítani egy diszkhozzáférés ideje alatt, standard diszkhardver mellett?

# **III. RÉSZ**

# **TUDÁS ÉS**

# **KÖVETKEZTETÉS**

---

## 7. LOGIKAI ÁGENSEK

*Ebben a fejezetben olyan ágenseket tervezünk, amelyek képesek reprezentációkat kialakítani a vilagról, következtetési folyamatot alkalmaznak a világ új reprezentációjának a származtatására, és felhasználják az új reprezentációt a teendők kikövetkeztetésére.*

Ez a fejezet tudásbázisú ágenseket mutat be. A fogalmak, amelyeket itt tárgyalunk – a tudás *reprezentációja* és a tudás alkalmazását lehetővé tevő következtetési folyamatok – központi témaí a mesterséges intelligencia minden területének.

Az emberek, úgy tűnik, ismernek a világról számos dolgot, és következtetéseket végeznek. A tudás és a következtetés fontos a mesterséges ágenseknek is, mert sikeres viselkedést tesz lehetővé, amelyet nagyon nehéz volna másképpen elérni. Láthattuk, hogy a problémamegoldó ágens számára cselekvéseinek ismerete lehetővé teszi, hogy komplex környezetekben jól teljesítsen. A reflexív ágens csak a vakszerencse segítségével tudta megtalálni az utat Aradról Bukarestbe. A problémamegoldó ágens tudása azonban igen specifikus és rugalmatlan. Egy sakkprogram ki tudja számítani egy király helyes lépéseiit, de nincsen semmilyen értelmezhető ismerete arról, hogy egy figura egyidejűleg nem lehet két különböző mezőn. A tudásbázisú ágens képes kihasználni a nagyon általános formában leírt tudást, újra és újra összegyűjtve ennek elemeit úgy, hogy az számos célra megfelelő legyen. Gyakran ez a folyamat nagyon messzire kerül a pillanatnyi igénytől, például amikor egy matematikus egy tételt bizonyít, vagy amikor egy csillagász a Föld várható élettartamát számítja.

A tudás és a következtetés alapvető szerepet játszanak a részben megfigyelhető környezetek kezelésénél is. A tudásbázisú ágens képes összekombinálni az általános tudást a pillanatnyi érzetekkel, hogy kikövetkeztesse a pillanatnyi állapotot rejtett aspektusait, mielőtt cselekvést választ. Ilyen például, amikor egy orvos diagnosztizál egy beteget, azaz kikövetkeztet egy közvetlenül nem megfigyelhető betegséggállapotot, mielőtt meghatározná a kezelés módját. A tudás egy része, amelyet az orvos használ, könyvekből vagy tanároktól megtanult szabályok formájában áll rendelkezésre, más része pedig asszociációs minták formájában van, amelyeket lehet, hogy az orvos nem is tud tudatosan leírni. Ha ezek is az orvos fejében vannak, akkor tudásnak számítanak.

A természetes nyelv megértése szintén igényli, hogy rejtett állapotokra következtesünk, nevezetesen, hogy a beszélő szándékát megismерjük. Amikor azt halljuk, hogy „János egy gyémántot látott az ablakon keresztül, és szeretné (azt) megkapni”, akkor tudjuk, hogy az „azt” a gyémántra vonatkozik, és nem az ablakra. Ilyenkor, talán nem is tudatosan, a relatív értékekről meglévő ismereteink segítségével következtetünk. Hasonlóan, amikor azt halljuk, hogy „János kidobott egy követ az ablakon keresztül, és betörte (azt)”, akkor tudjuk, hogy az „azt” az ablakra vonatkozik. A következtetés lehetővé teszi számunkra, hogy megbirkózzunk a kijelentések virtuálisan végtelen változatosságával, hétköznapi tudásunk egy véges halmozat felhasználva.

Az utolsó érvünk a tudásbázisú ágensek tanulmányozása mellett e rendszerek rugalmassága. Képesek explicit célok formájában megadott új feladatokat elfogadni, egy új környezetről kapott vagy megtanult új ismeretek révén kompetensé válni, és frissítve a tudásuk releváns részét képesek alkalmazkodni a környezet változásához.

A 7.1. alfejezetben az ágens általános tervezésével kezdünk. A 7.2. alfejezet egy egyszerű új környezetet, a wumpus világot mutatja be, amelyben illusztrálni fogjuk a tudásbázisú ágens működését a technikai részletek ismertetése nélkül. Ezután a 7.3. alfejezetben elmagyarázzuk a logika (logic) általános elveit. A könyv III. részében végig a logika lesz az elsőleges eszköz a tudás reprezentálására. A logikai ágensek tudása minden határozott, minden kijelentés vagy igaz, vagy hamis a világban, habár az ágens lehet agnosztikus néhány kijelentéssel kapcsolatban.

A logika jelentős pedagógiai előnye, hogy egy tudásbázisú ágens egyszerű reprezentációs formáját jelenti, viszont van néhány komoly korlátja is. A helyzet az, hogy az emberek vagy más ágensek által részlegesen megfigyelhető környezetekben végzett következetések jelentős része függ *bizonytalan* tudás felhasználásától. A logika nem tudja ezt a bizonytalanságot jól reprezentálni, ezért az V. részben a valószínűséget tárgyaljuk, amely már képes erre. A VI. és VII. részben számos reprezentációt tárgyalunk, köztük néhány folytonos matematikán alapulót, mint a Gauss-görbék keverését, neurális hálózatokat és más reprezentációkat.

A 7.4. alfejezet bemutat egy egyszerű logikát, az ítéletkalkulust (*propositional logic*). Miközben ez lényegesen kevésbé kifejező, mint az elsőrendű logika (*first-order logic*) (8. fejezet), az ítéletkalkulus lehetőséget ad arra, hogy illusztráljuk a logika minden alapvető fogalmát. Létezik egy jól megtervezett technológia az ítéletkalkuluson történő következetésre, amelyet a 7.5. és 7.6. alfejezetben írunk le. Végül a 7.7. alfejezet összekombinálja a logikai ágensek fogalmát az ítéletkalkulus technológiájával, hogy egyszerűbb ágenseket építsünk a wumpus világ számára. Azonosítjuk majd az ítéletkalkulus bizonyos hiányosságait, amelyek még hatékonyabb logikák fejlesztését fogják motiválni a következő fejezetekben.

## 7.1. A TUDÁSBÁZISÚ ÁGENS

Egy tudásbázisú ágens központi eleme a **tudásbázisa (knowledge base)**. Egyszerűen fogalmazva, a tudásbázis **mondatok (sentences)** halmaza. (Itt a „mondatot” technikai fogalomként használjuk. A fogalom kapcsolatban van az angol, a magyar vagy más természetes nyelvek mondataival, de nem azonos velük.) A mondatokat egy nyelv segítségével fejezzük ki, amelyet **tudásreprezentációs nyelvnek (knowledge representation language)** nevezünk, és a világról szóló állításokat fogalmazunk meg vele.

Új mondatoknak a tudásbázishoz való hozzáadására, illetve a tudás lekérdezésére valamelyen eljárásra van szükségünk. Ezeknek a feladatoknak a tipikus elnevezése a **KUELENT**, illetve a **KÉRDEZ**. Mindkét feladat tartalmazhat **következtetést (inference)**, azaz új mondatok levezetését régiekből. A logikai ágenseknél (*logical agents*), amelyekkel ebben a fejezetben foglalkozunk, a következtetéssel szemben az alapvető követelmény, hogy amikor valaki KÉRDEZ egy kérdést a tudásbázisról, akkor a válasznak következnie kell azokból a mondatokból, amit korábban a tudásbázishoz hozzáadtunk (Pontosabban **KIJELENT-ettünk**). Később a fejezetben még pontosabban leírjuk az igen

fontos „következik” fogalom jelentését. Most egyelőre ez jelentsen csak annyit, hogy a következetési folyamat során nem csak úgy egyszerűen keletkeznek dolgok.

A 7.1. ábra egy tudásbázisú ágens programjának vázát mutatja. Mint ahogy az összes ágensünk, ez is észlel valamit bemenetként, és egy cselekvést ad vissza válaszként. Az ágens fenntart egy tudásbázist, *TB*-t, amely kezdetben bizonyos **háttértudást** (**background knowledge**) tartalmazhat. Ha az ágensprogramot meghívják, az három dolgot tesz. Először KJELENT-i a tudásbázisnak, hogy mit észlelt. Másodszor KÉRDEZ-i a tudásbázist, hogy milyen cselekvést kell végrehajtania. A lekérdezés megválaszolásának folyamatában egyre bővülő következetést lehet végezni a világ pillanatnyi állapotáról, a lehetséges cselekvéssorozatok eredményéről és így tovább. Harmadszor az ágens rögzíti a kiválasztott cselekvést a KJELENT felhasználásával, és végrehajtja a cselekvést. A második KJELENT azért szükséges, hogy a tudásbázissal tudassuk, hogy a feltételezett cselekvés végrehajtása megtörtént.

```
function TB-ÁGENS(észlelés) returns egy cselekvés
static: TB, egy tudásbázis
t, egy számláló, kezdetben 0, mutatja az időt

KJELENT(TB, ÉSZLELÉS-MONDAT-KÉSZÍTÉS(észlelés, t))
cselekvés  $\leftarrow$  KÉRDEZ(TB, CSELEKVÉS-KÉRDEZÉS-KÉSZÍTÉS(t))
KJELENT(TB, CSELEKVÉS-MONDAT-KÉSZÍTÉS)(cselekvés, t)
t  $\leftarrow$  t + 1
return cselekvés
```

7.1. ábra. Egy általános tudásbázisú ágens

A reprezentációs nyelv részletei három függvényben vannak elrejtve, amelyek az érzékelők és beavatkozók, az alapreprezentáció, valamint a következető rendszer közötti kapcsolatot valósítják meg. Az ÉSZLELÉS-MONDAT-KÉSZÍTÉS eljárás egy olyan mondatot konstruál, amelyik megállapítja, hogy az ágens egy adott pillanatban észlelte az érzékelőt dolgot. A CSELEKVÉS-KÉRDEZÉS-KÉSZÍTÉS az időt felhasználva bemeneti adatként, visszatér egy mondattal, amely alkalmas arra, hogy megkérdezzük, milyen cselekvés szükséges ebben a pillanatban. Végül a CSELEKVÉS-MONDAT-KÉSZÍTÉS egy olyan mondatot hoz létre, amely megállapítja, hogy a kiválasztott cselekvés végrehajtása megtörtént. A következetési mechanizmus részletei a KJELENT és a KÉRDEZ eljárások belséjében vannak elrejtve. A későbbi fejezetekben ezeket a részleteket is bemutatjuk.

A 7.1. ábra ágense egészen hasonlónak tűnik, mint a 2. fejezetben bemutatott belső állapottal rendelkező ágens. Azonban a KJELENT és a KÉRDEZ eljárások definíciója miatt a tudásbázisú ágens nem egy tetszőleges, cselekvéseket meghatározó program. Meghatározható egy leírással a **tudásszinten** (**knowledge level**), ahol csak azt kell megfogalmaznunk, hogy mit tudjon és milyen céljai legyenek az ágensnek, hogy rögzíthessük a viselkedését. Például egy automata taxi számára a cél lehet egy utas elszállítása Marin megyébe, és a taxi ismerheti, hogy ez San Franciscónál van, és a Golden Gate híd az egyetlen, amelyen keresztül oda el lehet jutni. Ezután feltételezhetjük, hogy átmegegy a Golden Gate hídon, mivel tudja, hogy el fogja érni a célját. Vegyük észre,

hogy ez az elemzés független attól, hogy a taxi hogyan is működik az **implementációs szinten (implementation level)**. Nem számít, hogy a földrajzi elhelyezkedéssel kapcsolatos tudása láncolt listákkal vagy pixeltérképekkel van megvalósítva, vagy hogy vajon a következtetést regiszterékben tárolt szimbólumok listáinak manipulálásával vagy neuronhálókban zajos jelek továbbküldésével végzi el.



Említettük a fejezet bevezetőjében, hogy *egy tudásbázisú ágens megalkotható úgy, hogy KIELENT-jük számára, hogy mit szükséges tudnia*. Az ágens inicializáló programja a tervező környezetről lévő tudását leíró mondatok egyenkénti átadásával építhető fel, még mielőtt az ágens elkezdené érzékelni a környezetét. Rendkívül leegyszerűsíti az ágens létrehozásának problémáját, ha olyan reprezentációs nyelvet tervezünk, amellyel egyszerű lesz ennek a tudásnak mondatokban való megfogalmazása. A rendszerépítésnek ezt a megközelítési módját **deklaratív (declarative)** módszernek nevezzük. Ezzel szemben a **procedurális (procedural)** megközelítés közvetlenül a program kódjában rögzíti a kívánt viselkedést, amivel minimalizálja az explicit reprezentáció és a következtetés szerepét, és így lényegesen hatékonyabb rendszer jöhet létre. Mind a kétféle ágensre látunk majd példát a 7.7. alfejezetben. Az 1970-es és 1980-as években a két megközelítés támogatói parázs vitákat folytattak. Mi mára felismertük, hogy egy sikeres ágens tervezésekor kombinálni kell a deklaratív és a procedurális elemeket.

Amellett hogy KIELENT-hetjük az ágensnek, hogy mit kell tudnia, adhatunk is számára egy olyan mechanizmust, amely képessé teszi őt, hogy saját maga megtanulja ezt. Ez a mechanizmus, amit a 18. fejezetben tárgyalunk, általános, a környezetre vonatkozó tudást hoz létre érzetek sorozatának felhasználásával. Ezt a tudást bele lehet illeszteni az ágens tudásbázisába, és felhasználható a döntések meghozatalakor. Ily módon az ágens teljesen autonómmá váthat.

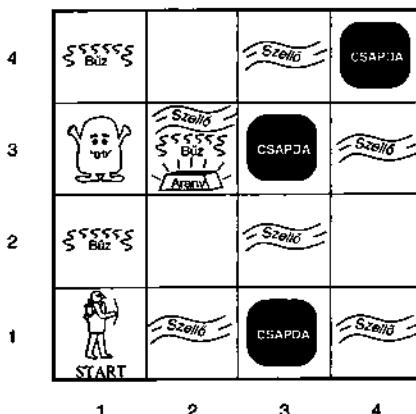
Ezek a képességek – a reprezentáció, a következtetés és a tanulás – mind a logika elméletének és technológiájának több évszázados fejlődésén alapulnak. Mielőtt azonban az elméletet és a technológiát elmagyaráznánk, létrehozunk egy egyszerű világot, amelyet majd felhasználunk ezek illusztrálásához.

## 7.2. A WUMPUS VILÁG

A **wumpus világ (wumpus world)** egy barlang, amely szobákból és az ezeket összekötő átjárókból áll. A wumpus egy szörnyeteg, aki mindenkit megesz, ha a szobájába lép, a barlangban lapul valahol. Az ágens le tudja lőni a wumpust, de csak egyetlen nyíla van ehhez. Néhány szoba fenecketlen csapdát tartalmaz, amely mindenkit csapdába ejt, aki belép a szobába (kivéve a wumpust, aki til nagy ahoz, hogy beleessen). A wumpus környezetében az egyetlen csábító lehetőség, hogy egy halom aranyat lehet találni. Habár a wumpus világ meglehetősen unalmas a modern számítógépes játékokhoz képest, azonban a játék kiváló tesztkörnyezet az intelligens ágensek számára. Michael Genesereth volt az első, aki javasolta ennek a környezetnek az alkalmazását.

Egy példa wumpus világ látható a 7.2. ábrán. A példakörnyezet pontos definícióját, ahogyan a 2. fejezetben javasoltuk, a TKCSÉ-leírással adjuk meg:

- Teljesítménymérték:** +1000 az arany felvétele, -1000 a csapdába esés vagy ha a wumpus felfal, -1 minden végrehajtott cselekvés, -10 a nyíl használata.



7.2. ábra. Egy tipikus wumpus világ. Az ágens a bal alsó sarokban van.

- Környezet:** Egy szobákból álló  $4 \times 4$ -es háló. Az ágens minden az [1,1]-gyel jelölt négyzetből indul, arccal jobbra nézve. Az arany és a wumpus elhelyezkedése véletlenszerűen, a kiinduló négyzeten kívüli négyzetek közül egyenletes eloszlás szerint van megválasztva. Ezen kívül még bármely, a kiinduló négyzeten kívüli négyzet 0.2 valószínűsséggel lehet csapda.
- Cselekvések:** Az ágens mozoghat előre, fordulhat balra  $90^\circ$ -kal, vagy fordulhat jobbra  $90^\circ$ -kal. Az ágens szörnyűséges halált hal, ha belép egy négyzetbe, ahol csapda van vagy egy élő wumpus található. (Biztonságos, habár meglehetősen rossz illatú egy olyan négyzetbe belépni, amelyben egy halott wumpus van.) Az előrelépésnek nincs hatása, ha egy fal van az ágens előtt. A *Megragad* cselekvést lehet arra használni, hogy az ágens felvegyen egy tárgyat, amely vele azonos szobában van. A *Lövés* cselekvést lehet használni egy nyílnak abban az irányban történő kilövésre, amerre az ágens éppen áll. A nyíl addig repül, amíg el nem találja (és egyben meg nem öli) a wumpust, vagy falnak nem ütközik. Az ágensnek csak egy nyila van, így csak egy *Lövés* cselekvésnek van hatása.
- Érzékelők:** Az ágensnek öt érzékelője van, mindegyik egyetlen bitnyi információt ad:
  - A wumpust tartalmazó négyzetben és a közvetlenül (nem átlósan) szomszédos négyzetekben az ágens búzt érez.
  - A csapdával közvetlenül szomszédos négyzetekben az ágens szellőt érzékel.
  - A négyzetben, ahol az arany található, az ágens csilllogást érzékel.
  - Ha az ágens falnak megy, akkor ütést érzékel.
  - Ha a wumpust megölték, akkor egy elkeseredett sikolyt hallat, amit a barlangban bárhol hallani lehet.
 Az érzeteket az ágens egy öt szimbólumot tartalmazó lista formájában kapja meg; például ha búz és szellő van egy négyzetben, de nincs ütés, csilllogás vagy sikoly, akkor az ágens egy [*Búz*, *Szellő*, *Nincs*, *Nincs*, *Nincs*] érzetet kap.

A 7.1. feladatban definiálni kell egy wumpus környezetet a 2. fejezetben megadott külböző dimenziók mentén. Az alapvető nehézség az ágens számára, hogy kezdetben semmit sem tud a környezet konfigurációjáról. Úgy tűnik, hogy logikai következtetésre

van szüksége az ágensnek ahhoz, hogy felülkerekedhessen a tudatlanság okozta hátrányon. A wumpus világok legtöbb példányában az ágens számára lehetséges az arany biztonságos megszerzése. Néhány környezetben azonban az ágensnek választania kell, hogy hazamegy-e üres kézzel, vagy kockázatot vállal, ami vagy az aranyhoz vagy a halálhoz vezet. Es a környezetek 21%-a teljesen tisztességtelen (mivel az arany egy csapdában van vagy csapdákkal körülvett mezőben).

Nézzünk meg egy tudásbázisú wumpus ágenst, hogy hogyan fedezí fel a 7.2. ábrán látható környezetet. Az ágens kezdeti tudásbázisa a környezetet leíró, az előzőkben felsorolt szabályokat tartalmazza. Nevezetesen tudja, hogy az [1, 1]-ben tartózkodik és hogy az [1, 1] biztonságos hely. Látni fogjuk, hogy hogyan bővül az ágens tudása, amint új érzékelések érkeznek és cselekvések történnek.

Az első érzékelés a [Nincs, Nincs, Nincs, Nincs, Nincs], amiből az ágens arra tud következtetni, hogy a szomszédos négyzetek biztonságosak. A 7.3. (a) ábra mutatja az ágens tudásának állapotát ezen a ponton. Az ábrán a tudásbázis néhány mondatát soroljuk fel, betűket használva a megfelelő négyzetekben, mint az Sz (szellő) és az OK (biztonságos, nincs se csapda, se wumpus). A 7.2. ábra ezzel szemben magát a világot írja le.

Abból a tényből, hogy nem volt se bűz, se szellő az [1, 1]-ben, az ágens kikövetkezetheti, hogy az [1, 2] és [2, 1] négyzetek veszélytelenek. Ennek jelzsére a megfelelő négyzetekbe OK-t írunk. Egy óvatos ágens csak olyan négyzetbe lép, amelyről tudja, hogy OK. Feltételezzük, hogy az ágens úgy dönt, hogy a [2, 1]-be megy, előállítva a 7.3. (b) ábrán látható helyzetet.

Az ágens detektálja a szellőt a [2, 1]-ben, tehát egy csapdának kell lennie valamelyik szomszédos négyzetben. A csapda nem lehet az [1, 1]-ben a játék szabályai szerint, így csapdának kell lennie a [2, 2]-ben vagy a [3, 1]-ben vagy mindenki között. A Cs? jelölés egy lehetséges csapdát jelez a mezőkben a 7.3. (b) ábrán. Ezen a ponton csak egy olyan ismert négyzet van, ami OK, és amit még nem látogatott meg. Így a megfontolt ágens visszafordul, visszamegy az [1, 1]-be és az [1, 2]-be halad tovább.

Az új érzet az [1, 2]-ben a [Bűz, Nincs, Nincs, Nincs, Nincs], ami a 7.4. (a) ábrán látható helyzetet eredményezi. A bűz az [1, 2]-ben azt jelenti, hogy a wumpusnak a közel-

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1 OK	4,1

(a)

**A** = Ágens  
**Sz** = Szellő  
**R** = Ragyogás,  
 Arany  
**OK** = Biztonságos  
 négyzet  
**Cs** = Csapda  
**B** = Bűz  
**M** = Meglátogatott  
**W** = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 M OK	2,1 Sz OK	3,1 Cs? OK	4,1

(b)

7.3. ábra. Az ágens első lépése a wumpus világban. (a) A kezdeti helyzet a [Nincs, Nincs, Nincs, Nincs, Nincs] érzékelése után. (b) Az első lépés után, érzékelve a [Nincs, Szellő, Nincs, Nincs, Nincs]-et.

ben kell lennie. De a wumpus a játék szabályai szerint nem lehet az [1, 1]-ben és nem lehet a [2, 2]-ben sem (mert akkor az ágens érezte volna a bűzt, amikor a [2, 1]-ben járt). Így az ágens kikövetkeztetheti, hogy a wumpus az [1, 3]-ban van. A *W!* jelölés ezt mutatja. Még érdekesebb, hogy a *Szellő* érzet hiánya az [1, 2]-ben azt jelenti, hogy nincs csapda a [2, 2]-ben. De mi már kikövetkeztettük, hogy vagy a [2, 2]-ben, vagy a [3, 1]-ben van egy csapda, ami tehát azt jelenti, hogy a csapdának a [3, 1]-ben kell lennie. Ez egy viszonylag nehéz következetetés, mivel különböző időpontokban és különböző helyeken gyűjtött tudást használ fel, és egy érzet hiányára támaszkodva végez el egy fontos lépést. Ez a következetés meghaladja a legtöbb állat képességeit, de tipikusan jellemzi azt a fajta következetést, amit egy logikai ágens végez.

Az ágens így bebizonyította maga számára, hogy nincs se csapda, se wumpus a [2, 2]-ben, így a mozgás ebbe a négyzetbe *OK*. Nem mutatjuk be az ágens tudását a [2, 2]-ben, csak feltételezzük, hogy fordul és átlép a [2, 3]-ba, ami a 7.4. (b) ábrán látható. A [2, 3]-ban az ágens detektálja a csillagást, így megragadja az aranyat, és ezzel véget ér a játék.

1,4	2,4	3,4	4,4
1,3 <i>W!</i>	2,3	3,3	4,3
1,2 A B OK	2,2	3,2	4,2
1,1 M OK	2,1 B M OK	3,1 Cst	4,1

(a)

**A** = Ágens  
**Sz** = Szellő  
**R** = Rágogás,  
Arany  
**OK** = Bíztonságos  
négyzet  
**Cs** = Csapda  
**B** = Bűz  
**M** = Meglátogatott  
**W** = Wumpus

1,4	2,4	3,4	4,4
1,3 <i>W!</i>	2,3 A B R Sz	3,3 Cs?	4,3
1,2 B B OK	2,2 B OK	3,2	4,2
1,1 M OK	2,1 B M OK	3,1 Cst	4,1

(b)

7.4. ábra. Két későbbi helyzet az ágens előrehaladása során. (a) A harmadik lépést követően, miután [Bűz, Nincs, Nincs, Nincs]-et érzékelt. (b) Az ötödik lépés és a [Bűz, Szellő, Csillagás, Nincs, Nincs] érzékelése után

**Bármely esetben, amikor az ágens következetéseket von le a rendelkezésre álló információkból, a következmény garantáltan helyes lesz, ha a rendelkezésre álló információk helyesek.** Ez alapvető jellegzetessége a logikai következetéseknek. A fejezet hátralevő részében megmutatjuk, hogyan építhetünk olyan logikai ágenseket, amelyek képesek reprezentálni a szükséges információkat, és következetéseket vonnak le, ahogy azt az eddigi fejezetekben leírtuk.

## 7.3. A LOGIKA

Ez az alfejezet áttekintést nyújt a logikai reprezentáció és következetés alapvető fogalmairól. A logika bármely speciális formájára vonatkozó technikai részletek bemutatását a következő fejezetre halasztjuk. Ehelyett egyszerű példákat fogunk használni a wumpus

világóból vagy ismerős aritmetikai területekről. Azért választjuk ezt az igen rendhangyó megközelítést, mivel a logika fogalmai messze általánosabbak és szébbek, mint azt általában feltételezik.

A 7.1. fejezetben említettük, hogy a tudásbázis mondatokból áll. Ezeket a mondatokat a reprezentációs nyelv **szintaxisa** (*syntax*) szerint fejezzük ki, amely specifikálja az összes jó formált, nyelvtanilag helyes mondatot. A szintaxis fogalma elég tiszta a szokásos aritmetikai műveleteknél: „ $x + y = 4$ ” egy jó formált mondat, míg az „ $x2y + =$ ” nem az. A logikai nyelvek szintaxisait (és a matematikáét is egyébként) rendszerint cikkek és könyvek írása céljából terveztek. A szó szoros értelmében tucatjai léteznek a különböző szintaxisoknak, néhány tele görög betűkkel és egzotikus matematikai szimbólumokkal, néhány inkább vizuálisan követhető, nyílkat és buborékokat tartalmazó diagramkból áll. minden esetben azonban, az ágens tudásbázisában a mondatok az ágensnek magának vagy az ágens egy részének valodi fizikai konfigurációi. A következetés ezeknek a konfigurációknak a létrehozását vagy manipulálását fogja jelenteni.

A logikának a nyelv **szemantikáját** (*semantics*) is definiálnia kell. Egyszerűen fogalmazva a szemantika a mondatok „jelentéséről” szól. A logikában a definíció pontosabb. A nyelv szemantikája definiálja minden mondat igazságát (*truth*) minden egyes lehetséges világra (*possible world*) vonatkozóan. Például egy szokásos aritmetikához választott szemantika meghatározza, hogy az „ $x + y = 4$ ” mondat igaz abban a világban, ahol  $x$  értéke 2 és  $y$  értéke 2, de hamis abban a világban, ahol  $x$  értéke 1 és  $y$  értéke is 1.<sup>1</sup> A standard logikákban minden mondat vagy igaz, vagy hamis minden lehetséges világban, és nem lehet „valahol az igaz és hamis között”<sup>2</sup>.

Amikor szükséges, hogy pontosak legyünk, a **modell** (*model*) kifejezést fogjuk használni a „lehetséges világ” helyén. (Szintén használni fogjuk az „ $m$  modellje  $\alpha$ -nak” kifejezést, ami azt jelenti, hogy az  $\alpha$  mondat igaz az  $m$  modellben.) Miután a lehetséges világokat úgy képzelhetjük el, mint (potenciálisan) valós környezeteket, amelyekben az ágens ott lehet vagy nem lehet ott, a modellek olyan matematikai absztrakciók, amelyek csak rögzítik az igazság vagy hamisság értékét minden releváns mondatnak. Például, tegyük fel, hogy  $x$  és  $y$  a férfiak és nők száma, akik egy kártyaasztal körül ülnek és bridzset játszanak, és az  $x + y = 4$  mondat igaz, amikor négyen vannak összesen. Formálisan, a lehetséges modellek nem mások, mint minden lehetséges hozzárendelés az  $x$  és  $y$  változókhöz. minden ilyen hozzárendelés bármely olyan aritmetikai mondat igazságát rögzíti, amely az  $x$  és  $y$  változókat tartalmazza.

Most, hogy van egy képünk az igazság fogalmáról, tudunk beszélni a logikai következetettsérről. Ennek része a mondatok közötti logikai **vonzat** (*entailment*) reláció, annak kifejezése, hogy egy mondat logikusan következik egy másik mondatból. Matematikai jelöléssel ezt így írjuk:

$$\alpha \models \beta$$

<sup>1</sup> Az olvasó minden bizonnal észrevette a hasonlóságot a mondatok igazságának fogalma és az 5. fejezetben bemutatott kényszerek kielégítésének fogalma között. Ez nem véletlen, a kényszerelvek valójában logikák, és a kényszerproblémák megoldása a logikai következetés egy formája.

<sup>2</sup> A **fuzzy logika** (*fuzzy logics*), amelyet a 14. fejezetben mutatunk be, megengedi az igazság fokának kezelését.

aminek jelentése, hogy az  $\alpha$  mondat maga után vonzza a  $\beta$  mondatot. A vonzat formális definíciója a következő:  $\alpha \models \beta$  akkor és csak akkor, ha minden modellben, amelyben  $\alpha$  igaz,  $\beta$  szintén igaz. Közvetlenebbül azt mondhatjuk, hogy  $\beta$  igazságát „tartalmazza”  $\alpha$  igazsága. A vonzat reláció ismerős az aritmetikából is, örömmel vehetjük észre, hogy az  $x + y = 4$  mondat maga után vonzza a  $4 = x + y$  mondatot. Nyilvánvaló, hogy bármely modellben, ahol  $x + y = 4$ , mint például az a modell, amelyben  $x$  és  $y$  is 2 értékű, a  $4 = x + y$  is fenn áll. Hamarosan látni fogunk, hogy a tudásbázist tekintetük egy kijelentésnek, és gyakran beszélhetünk arról, hogy egy tudásbázis maga után vonz egy mondatot.

Alkalmazhatunk hasonló elemzést az előző részben bemutatott wumpus világbeli következtetési példára is. Tekintstük a 7.3. (b) ábrán látható szituációt: az ágens nem észlelt semmit az [1, 1]-ben és szellőt észlelt a [2, 1]-ben. Ezek az érzetek, kombinálva az ágensnek a wumpus világ szabályaira vonatkozó tudásával (a 250–251. oldalon található TKCSÉ-leírás), alkotja a  $TB$ -t. Az ágenst (más dolgok mellett) az érdeklő, hogy vajon a szomszédos [1, 2], [2, 2], [3, 1] négyzetek tartalmaznak-e csapdát. Bármelyik a három négyzet közül tartalmazhat vagy éppen nem tartalmaz csapdát, így a példa esetében  $2^3 = 8$  lehetséges modell létezik. Ezt mutatja a 7.5. ábra.<sup>3</sup>

A  $TB$  hamis azokban a modellekben, amelyek ellentmondanak annak, amit az ágens tud. Például a  $TB$  hamis minden modellben, ahol az [1, 2] tartalmaz csapdát, mivel nincs szellő az [1, 1]-ben. Valójában csak három olyan modell van, amelyben a  $TB$  igaz, ezeket a 7.5. ábra a modellek egy részhalmazaként mutatja. Most tekintsünk két lehetséges következményt:

$$\begin{aligned}\alpha_1 &= \text{„Nincs csapda az [1, 2]-ben”} \\ \alpha_2 &= \text{„Nincs csapda a [2, 2]-ben”}\end{aligned}$$

A 7.5. ábrán megjelöltük az  $\alpha_1$  és  $\alpha_2$  modelleket. Szemrevételezve megállapíthatjuk a következőt:

minden olyan modellben, ahol a  $TB$  igaz, ott  $\alpha_1$  is igaz.

Így  $TB \models \alpha_1$ , és nincs csapda az [1, 2]-ben. Azt is láthatjuk, hogy:

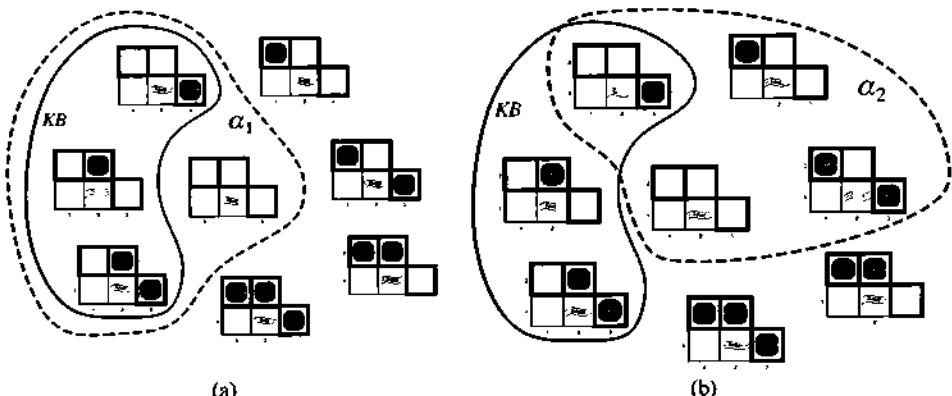
néhány modell, amelyben a  $TB$  igaz, az  $\alpha_2$  hamis.

Így  $TB \not\models \alpha_2$  és az ágens nem tudja kikövetkeztetni, hogy nincs csapda a [2, 2]-ben. (És azt sem tudja kikövetkeztetni, hogy van csapda a [2, 2]-ben.)<sup>4</sup>

Az előző példa nem csak illusztrálja a maga után vonzást, hanem megmutatja, hogy a vonzat definícióját fel lehet használni a következmények levezetésére, azaz, hogy logikai következtetést (logical inference) végezzünk. A 7.5. ábrán illusztrált következtetési algoritmust modelellenőrzésnek (model checking) hívjuk, mivel számba vesz minden lehetséges modellt annak megvizsgálására, hogy  $\alpha$  igaz-e minden modellben, amelyben a  $TB$  igaz.

<sup>3</sup> Habár az ábra a modelleket, mint egy részleges wumpus világot mutatja, ezek valójában nem mások, mint az igaz és hamis értékek hozzárendelései a „csapda van az [1, 2]-ben” mondathoz. A modellek, matematikai értelemben, nem igénylik, hogy szörnyűséges illatú wumpusok legyenek benne.

<sup>4</sup> Az ágens ki tudja számolni, hogy mekkora a valószínűsége annak, hogy van csapda a [2, 2]-ben. A 13. fejezet fogja megmutatni azt, hogy hogyan.



**7.5. ábra.** Lehetséges modelljei a csapda jelenlétének az [1, 2]-[2, 2] és [3, 1]-ben, ha adott a megfigyelés, hogy az [1, 1]-ben semmi és a [2, 1]-ben szellő érezhető. (a) A tudásbázis és  $\alpha_1$ (nincs csapda[1, 2]-ben) modelljei. (b) A tudásbázis és  $\alpha_2$ (nincs csapda[2, 2]-ben) modelljei.

A vonzat és a bizonyítás megértésében segíthet, ha a  $TB$  összes következményeit egy szénakazalnak, az  $\alpha$ -t pedig egy gombostűnek képzeljük el. A vonzat olyan, mintha a gombostű benne volna a kazalban; a bizonyítás pedig nem más, mint megpróbálni megtalálni ezt a tűt. Ez a különbségtétel formálisan a következő megfogalmazásban ölt testet: ha egy  $i$  következtetési algoritmus képes levezetni  $\alpha$ -t a  $TB$ -ből, akkor írhatjuk, hogy

$$TB \vdash_i \alpha$$

amely kimondva: „ $\alpha$  az  $i$  által levezethető  $TB$ -ból” vagy „ $i$  levezeti  $\alpha$ -t a  $TB$ -ból”.

Egy következtetési eljárást, amely csak vonzat mondatokat vezet le, **helyesnek** (*sound*) vagy **igazság tartónak** (*truth-preserving*) nevezzük. A helyesség egy igencsak kívánatos tulajdonság. Egy nem helyes következtetési eljárás kitalál olyan dolgokat ahogy előrehallad, olyan tükk megtalálását jelenti be, amelyek nem is léteznek. Egyszerű belátni, hogy a modellellenőrzés, ha alkalmazható,<sup>5</sup> akkor helyes eljárás.

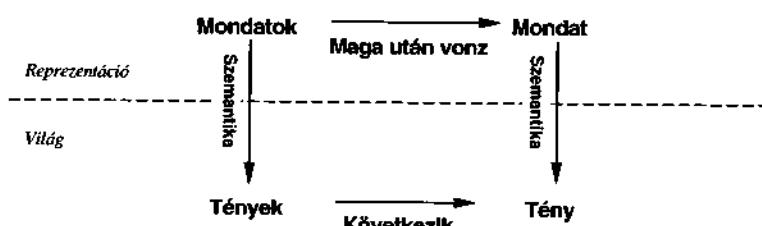
A **teljesség** (*completeness*) tulajdonság szintén kívánatos: egy következtetési eljárás teljes, ha képes levezetni minden vonzatmondatot. Valódi szénakazlak esetében, amelyek véges méretűek, nyilvánvalónak tűnik, hogy szisztematikus kutatással minden előírtható, hogy a tű a kazalban van-e. Sok tudásbázis esetében azonban a konzékvenciák szénakazlának mérete végtelen, és így a teljesség egy fontos kérdéssé válik.<sup>6</sup> Szerencsére léteznak teljes következtetési eljárások a logikához, amelyek megfelelően kifejezők ahhoz, hogy számos tudásbázist kezeljenek.

Egy olyan következtetési folyamatot írtunk le, amelynek következményei bármely világban garantáltan igazak, ahol a premisszák is igazak. Nevezetesen, ha a  $TB$  igaz a valódi világban, akkor bármely, a  $TB$ -ből helyes következtetési eljárással levezetett  $\alpha$

<sup>5</sup> A modellellenőrzés működik, ha a modellek tere véges, mint például egy rögzített méretű wumpus világ esetében. Az aritmetika esetében ezzel szemben a modellek tere végtelen, még akkor is, ha korlátottuk magunkat az egész számokra, mivel végtelen számú  $x, y$  értékpár létezik az  $x + y = 4$  egyenlethez.

<sup>6</sup> Hasonlítsuk össze a 3. fejezet végtelen keresési tereinek esetével, ahol látható, hogy a mélységi keresés nem teljes.

*mondat szintén igaz a valódi világban.* Így, míg a következtetési folyamat a „szintaxison” működik – belső fizikai konfigurációkon, mint például regiszterek bitjein vagy az egy elektromos jelzéseinél mintánál –, addig a folyamat megfelel valódi világ viszonyainak. Ennek megfelelően a valódi világ néhány aspektusa lesz az eset,<sup>7</sup> mivel a valódi világ bizonyos más aspektusai jelenleg képezik az esetet. Ezt a megfeleltetést a világ és a reprezentáció között mutatja a 7.6. ábra.



**7.6. ábra.** A mondatok az ágens fizikai konfigurációi, és a következtetés az a folyamat, amely új fizikai konfigurációkat hoz létre régióból. A logikai következtetésnek biztosítania kell, hogy az új konfigurációk olyan aspektusait reprezentálják a világnak, amelyek ténylegesen is következnek azokból az aspektusokból, amelyeket a régi konfigurációk reprezentálnak.

Az utolsó kérdés, amivel foglalkoznunk kell a logikai ágensek tárgyalásánál, a **meg-alapozottság (grounding)** kérdése, ami nem más, mint a kapcsolat, ha egyáltalán létezik ilyen, a logikai következtetési folyamat és a valódi környezet között, amelyben az ágens létezik. Nevezetesen *hogyan tudhatjuk meg, hogy a TB igaz-e a valódi világban?* (Ezután a TB már csak „szintaxis” az ágens fejében.) Ez egy filozófiai kérdés, amelyről sok-sok könyvet írtak (lásd 26. fejezet). Egy egyszerű válasz az, hogy az ágens érzékelői létesítik a kapcsolatot. Például a mi wumpus világbeli ágenseinknek van egy szagló érzékelője. Az ágensprogram létrehoz egy megfelelő mondatot minden, ha van illat. Így bármikor, ha ez a mondat a tudásbázisban van, ez igaz a valódi világban is. Ezáltal az érzetmondatok jelentését és igazságát az őket létrehozó érzékelő és mondatkonstruáló folyamatok határozzák meg. És mi a teendő az ágens tudásának egyéb részeivel, mint az a meggyőződése, hogy a wumpus rossz illatot terjeszt a szomszédos négyzetekben? Ez nem egy közvetlen reprezentációja egy egyedi érzetnek, hanem egy általános szabály, például érzékelési tapasztalatokból levezetve, de nem azonos magával a tapasztalatnak a kijelentésével. Az ilyen általános szabályokat a **tanulásnak (learning)** nevezett mondatkonstruáló folyamat hozza létre, ami a VI. résznak a tárgya. A tanulás nem tévedhetetlen. Lehet, hogy az eset az, hogy a wumpusok minden rossz illatot árasztanak, kivéve szökőrévekben február 29-én, amikor egyébként megfürödnek. Így lehet, hogy a TB nem igaz a valódi világban, de jó tanuló eljárásokkal van ok az optimizmusra.



<sup>7</sup> Mint azt Wittgenstein (1922) írta híres művében, a *Tractatusban*: „A világ minden, aminek az esete fennáll.”

## 7.4. AZ ÍTÉLETKALKULUS: EGY NAGYON EGYSZERŰ LOGIKA

Most egy nagyon egyszerű logikát, az ítéletkalkulust (**propositional logic**) mutatjuk be.<sup>8</sup> Áttekinjük az ítéletkalkulus szintaxisát, majd a szemantikáját – annak a módját, ahogy a mondatok igazságát meghatározzuk. Azután megnézzük a maga után vonzást – a relációt egy adott mondat és azon mondat között, amelyik az előbbiből következik –, és megnézzük, hogy hogyan vezet ez egy egyszerű logikai következetés algoritmushoz. Mindez természetesen a wumpus világban fog lejátszódni.

### Szintaxis

Az ítéletkalkulus szintaxisa meghatározza a lehetséges mondatokat. Az **atomi mondatok** (**atomic sentences**) – oszthatatlan szintaktikai elemek – egyetlen ítéletszimbólumból (**proposition symbol**) állnak. minden ilyen szimbólum egy kijelentés, ami igaz vagy hamis lehet. Nagybetűs neveket fogunk használni a szimbólumok jelölésére:  $P$ ,  $Q$ ,  $R$  és így tovább. A nevek tetszőlegesek, de gyakran úgy választjuk őket, hogy a nevek bizonyos jelentéssel is rendelkezzenek. Például használhatjuk a  $W_{1,3}$ -t annak az állításnak a kifejezésére, hogy a wumpus az [1, 3]-ban van. (Ne felejtük el, hogy a  $W_{1,3}$  *atomic*, így a  $W$ , az 1 és a 3 nem jelentéssel bíró részei a szimbólumnak.) Létezik két ítéletszimbólum, amelyeknek rögzített az értelmezése: az *Igaz* egy minden igaz állítás, és a *Hamis* egy minden hamis állítás.

**Összetett mondatok** (**complex sentences**) létrehozhatók egyszerűbb mondatokból logikai összekötőjelek (**logical connectives**) felhasználásával. Őt elterjedten használt összekötőjel van:

- ¬ (nem). Egy mondatot, mint a  $\neg W_{1,3}$ -t, **negációnak** (**negation**) nevezünk. Egy **literál** (**literal**) vagy egy atomi mondat (egy pozitív literál), vagy egy negált atomi mondat (egy negatív literál).
- ∧ (és). Egy mondatot, amelynek fő kötőszava a  $\wedge$ , mint például a  $W_{1,3} \wedge C_{1,3}$  **konjunkció**-nak (**conjunction**) nevezünk; ennek részei a **konjunktok** (**conjuncts**). (Az  $\wedge$  jel hasonlít egy A-ra, az angol *And* (és) szóból.)
- ∨ (vagy). Egy mondat, amely használja a  $\vee$  összekötőjelet, mint a  $(W_{1,3} \wedge C_{1,3}) \vee W_{2,2}$  egy **diszjunkció** (**disjunction**), méghozzá a  $(W_{1,3} \wedge C_{1,3})$  és a  $W_{2,2}$  **diszjunktoknak** (**disjuncts**) a diszjunkciója. (Történelmileg a  $\vee$  jel a latin *vel* kifejezésből származik, ami „vagy”-ot jelent. A legtöbb ember számára könnyebb megjegyezni úgy, mint egy fejjel lefelé álló és jelet, vagy a magyar olvasók számára, mint a *vagy* szó első betűjét.)
- ⇒ (implikáció). Egy mondatot, mint amilyen a  $(W_{1,3} \wedge C_{1,3}) \Rightarrow \neg W_{2,2}$  **implikáció**-nak (**implication**) (vagy feltételes mondatnak) nevezünk. Ennek **premisszája** (**premise**) vagy **előzménye** (**antecedent**) a  $(W_{1,3} \wedge C_{1,3})$ , **konklúziója** (**conclusion**) vagy **következménye** (**consequent**) pedig a  $\neg W_{2,2}$  **implikáció szabályként** (**rule**) vagy **ha–akkor**

<sup>8</sup> Az ítéletkalkulust **Boole-logikának** (**Boolean logic**) is nevezik, a logikával foglalkozó George Boole (1815–1864) után.

(if–then) állításként is ismert. Az implikációt más könyvekben időnként a  $\supset$  vagy a  $\rightarrow$  szimbólumokkal jelölik.

$\Leftrightarrow$  (akkor és csakis akkor). A  $W_{1,3} \Leftrightarrow \neg W_{2,2}$  mondat egy **ekvivalencia (biconditional)**.

A 7.7. ábra mutatja az ítéletkalkulus formális nyelvtanát; ha nem ismeri a BNF jelölésrendszerét, akkor nézze meg az 1112. oldalt.

*Mondat*  $\rightarrow$  *AtomiMondat* | *KomplexMondat*

*AtomiMondat*  $\rightarrow$  **Igaz** | **Hamis** | *Szimbólum*

*Szimbólum*  $\rightarrow$  **P** | **Q** | **R** | ...

*KomplexMondat*  $\rightarrow$   $\neg$ *Mondat*

| (*Mondat*  $\wedge$  *Mondat*)

| (*Mondat*  $\vee$  *Mondat*)

| (*Mondat*  $\Rightarrow$  *Mondat*)

| (*Mondat*  $\Leftrightarrow$  *Mondat*)

7.7. ábra. Ítéletkalkulus-beli mondatok BNF (Backus–Naur-forma) nyelvtana

Vegyük észre, hogy a nyelvtan nagyon szigorú a zárójelezésnél: minden mondatot, amelyet bináris összekötőjellel hozunk létre, zárójelek közé kell tenni. Ez biztosítja, hogy a szintaxis teljesen egyértelmű. Ez azt is jelenti például, hogy  $((A \wedge B) \Rightarrow C)$ -t kell írunk  $A \wedge B \Rightarrow C$  helyett. Az olvashatóság javítása céljából gyakran elhagyjuk a zárójeleket, megbízva ehelyett az összekötőjeleknek egy precedencia-sorrendjében. Ez hasonló az aritmetikában alkalmazott precedenciához – például az  $ab + c$ -t  $((ab) + c)$ -nek olvassuk, és nem  $a(b + c)$ -nek, mert a szorzásnak magasabb a precedenciája, mint az összeadásnak. Az ítéletkalkulus precedencia-sorrendje (a legmagasabbtól a legalacsonyabb felé:  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$  és  $\Leftrightarrow$ ). Így a mondat:

$$\neg P \vee Q \wedge R \Rightarrow S$$

ekvivalens a következő mondattal:

$$((\neg P) \vee (Q \wedge R)) \Rightarrow S$$

A precedencia nem oldja fel a többértelműséget az olyan mondatoknál, mint az  $A \wedge B \wedge C$ , amelyet olvashatunk  $((A \wedge B) \wedge C)$ -ként vagy  $(A \wedge (B \wedge C))$ -nek. Mivel a mondatnak ez a két olvasása ugyanazt jelenti a következő részben definiálandó szemantika szerint, az olyan mondatok, mint az  $A \wedge B \wedge C$  megengedettek. Szintén megengedjük az  $A \vee B \vee C$  és az  $A \Leftrightarrow B \Leftrightarrow C$  mondatokat. Olyan mondatok, mint az  $A \Rightarrow B \Rightarrow C$  nem megengedettek, mert a két olvasásnak különböző jelentései vannak, ebben az esetben ragaszkodunk a zárójelhez. Végül, néha használni fogunk szögletes zárójelet az egyszerű zárójel helyett, ami a mondatot áttekinthetőbbé teszi majd.

## Szemantika

Most, hogy specifikáltuk az ítéletkalkulus szintaxisát, definiáljuk a szemantikáját is. A szemantika definiálja a szabályokat, amivel meghatározható a mondat igazsága egy bizonyos modellben. Az ítéletkalkulusban a modell egyszerűen az igazságértéket – *igaz* vagy *hamis* – rögzíti minden ítéletszimbólumra. Például ha a tudásbázis mondatai a  $C_{1,2}$ ,  $C_{2,2}$  és  $C_{3,1}$  ítéletszimbólumokat használják fel, akkor egy lehetséges modell:

$$m_1 = \{C_{1,2} = \text{hamis}, C_{2,2} = \text{hamis}, C_{3,1} = \text{igaz}\}$$

Három ítéletszimbólum esetén  $2^3 = 8$  lehetséges modell van – pontosan azok, amelyek a 7.5. ábrán vannak feltüntetve. Vegyük észre azonban, hogy miután elköteleztük magunkat egy szintaxis mellett, a modellek tisztán matematikai objektumokká váltak, amelyeknek nem feltétlenül van kapcsolatuk a wumpus világgal. A  $C_{1,2}$  csak egy szimbólum, jelentheti azt, hogy „csapda van az [1, 2]-ben” vagy azt, hogy „Párizsban vagyok ma és holnap”.

Az ítéletkalkulus szemantikájának meg kell határoznia, hogyan számítsuk ki bármely mondat igazságértékét egy adott modellben. Ez rekurzívan történik. minden mondat atomi mondatokból és az ötféle összekötőjelből lett létrehozva, így meg kell határoz-nunk, hogy hogyan számítsuk ki az atomi mondatok igazságát, és meg kell határoznunk azt is, hogy hogyan számítsuk ki az igazságát az egyes összekötőjelek felhasználásával formált összetett mondatnak. Az atomi mondatok esete egyszerű:

- *Igaz* akkor, ha minden modellben igaz, és *Hamis* akkor, ha minden modellben hamis.
- minden más ítéletszimbólumnak az igazságértékét közvetlenül a modellben kell meghatározni. Például a korábban megadott  $m_1$  modellben a  $C_{1,2}$  hamis.

Összetett mondatokra olyan szabályaink vannak, mint

- Bármely  $s$  mondatra és bármely  $m$  modellre, a  $\neg s$  mondat az  $m$ -ben akkor és csak akkor igaz, ha  $s$  hamis  $m$ -ben.

Az ilyen szabályok visszavezetik az összetett mondatok igazságának előtöntését egyszerűbb mondatokra. minden összekötőjelre vonatkozó szabály összefoglalható egy igazságátlában (truth table), amely meghatározza egy összetett mondat igazságértékét a mondat komponenseinek minden lehetséges igazságérték hozzárendelésehez. Az öt logikai összekötőjel igazságátláját mutatja a 7.8. ábra. Ilyen táblák felhasználásával bármely  $s$  mondat igazságértéke egy  $m$  modellre vonatkozóan kiszámolható rekurzív kiértékelések egyszerű folyamatával. Például a  $\neg C_{1,2} \wedge (C_{2,2} \vee C_{3,1})$  mondatot kiértékelve  $m_1$ -ben  $\text{igaz} \wedge (\text{hamis} \vee \text{igaz}) = \text{igaz} \wedge \text{igaz} = \text{igaz}$  értéket kapunk. A 7.3. feladat azt kéri, hogy írjon egy IK-IGAZ?( $s, m$ ) algoritmust, amely kiszámítja az  $s$  ítéletkalkulus mondatnak az igazságértékét egy  $m$  modellben.

Korábban azt mondta, hogy a tudásbázist mondatok halmaza alkotja. Most láthatjuk, hogy egy logikai tudásbázis ilyen mondatok konjunkciója. Tehát ha egy üres  $TB$ -vel kezdünk, és végrehajtjuk a KIELENT( $TB, S_1$ ), ..., KIELENT( $TB, S_n$ ) műveleteket, akkor a  $TB = S_1 \wedge \dots \wedge S_n$  áll elő. Ez azt jelenti, hogy tudásbázisokat és mondatokat egymással felcserélhetően használhatunk.

Az „és”, „vagy” és „nem” igazságátlái eltérnek attól, amit a természetes nyelvi jelentésük alapján gondolnánk. A lehetséges félreérítés legszembetűnöbb pontja, hogy a

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>Hamis</i>	<i>Hamis</i>	<i>Igaz</i>	<i>Hamis</i>	<i>Hamis</i>	<i>Igaz</i>	<i>Igaz</i>
<i>Hamis</i>	<i>Igaz</i>	<i>Igaz</i>	<i>Hamis</i>	<i>Igaz</i>	<i>Igaz</i>	<i>Hamis</i>
<i>Igaz</i>	<i>Hamis</i>	<i>Hamis</i>	<i>Hamis</i>	<i>Igaz</i>	<i>Hamis</i>	<i>Hamis</i>
<i>Igaz</i>	<i>Igaz</i>	<i>Hamis</i>	<i>Igaz</i>	<i>Igaz</i>	<i>Igaz</i>	<i>Igaz</i>

**7.8. ábra.** Az öt logikai összekötőjel igazságátáblája. Amikor a táblát használjuk például a  $P \vee Q$  értékének számítására, ha  $P$  igaz és  $Q$  hamis, akkor először megkeressük azt a sort, amelyben  $P$  igaz és  $Q$  *hamis* (a harmadik sor). Ezután a sorban megkeressük a  $P \vee Q$  alatti oszlopot, hogy megtaláljuk az eredményt: *igaz*. Tekinthetjük a táblázat úgy is, hogy minden sor egy modell, és az egyes oszlopbeli elemek az adott sorban azt mondják meg, hogy a megfelelő mondat igaz-e az adott modellben.

$P \vee Q$  kifejezés igaz akkor is, ha mind  $P$ , mind  $Q$  is igaz. Létezik másik összekötőjel is, a „kizárt vagy”-nak nevezett jel (röviden *xor*), amely hamisat ad, ha minden diszjunkt igaz.<sup>9</sup> Nincs általános egyetértés az „exkluzív vagy” szimbólumát illetően; két jelölés is ismert: a  $\vee$  és a  $\oplus$ .

Az implikáció ( $\Rightarrow$ ) igazságátáblája tejtélyesnek tűnhet első látásra, mivel nem teljesen illeszkedik összetönös megértésünkhez, hogy „ $P$  implikálja  $Q$ -t” vagy „ha  $P$ , akkor  $Q$ ”. Az ítéletkalkulus nem kíván semmilyen ok-okozati relációt vagy relevanciát  $P$  és  $Q$  között. A mondat: „az a tény, hogy 5 páratlan implikálja, hogy Tokió Japán fővárosa” az ítéletkalkulusnak egy igaz mondata (a normális interpretáció szerint), még akkor is, ha ez határozottan furcsa mondatnak tűnik. Egy másik esete a félreérteseknek, hogy bármely implikáció igaz, ha az előzménye hamis. Például az „az az állítás, hogy 5 párós szám, implikálja, hogy Samu okos” mondat igaz, függetlenül attól, hogy Samu okos-e. Ez bizarnak tűnik, de elfogadható, ha a „ $P \Rightarrow Q$ ”-t úgy értelmezzük, hogy „ha  $P$  igaz, akkor azt állítom, hogy  $Q$  is igaz. Egyébként nem állítok semmit”. Az egyetlen eset, amikor ez a mondat *hamis*, ha  $P$  *igaz*, de  $Q$  *hamis*.

A  $P \Leftrightarrow Q$  ekvivalencia igazságátáblája azt mutatja, hogy akkor igaz, ha mind  $P \Rightarrow Q$  és  $Q \Rightarrow P$  igaz. Ezt gyakran úgy írjuk le, hogy „ $P$  akkor és csak akkor, ha  $Q$ ” vagy matematikában szokták jelölni „ $P$  aa  $Q$ ”-nak. A wumpus világ szabályait legjobban az  $\Leftrightarrow$  használatával tudjuk felírni. Például, egy négyzet szellő van, ha a szomszédos négyzetben csapda van, és egy négyzet csak akkor szellő van, ha a szomszédos négyzetben csapda van. Így ekvivalenciákra van szükségünk, mint a

$$S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$$

ahol  $S_{1,1}$  jelenti, hogy szellő van az [1, 1]-ben. Vegyük észre, hogy az egyirányú implikáció

$$S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})$$

igaz a wumpus világban, de nem teljes. Nem szabályozza azokat a modelleket, amelyekben  $S_{1,1}$  hamis és  $C_{1,2}$  igaz, amivel megszegnénk a wumpus világ szabályait. Egy másik mód ennek érzékeltetésére, hogy az implikáció igényli a csapda jelenlétét, ha szellő van, miközben az ekvivalencia szintén megkívánja a csapda hiányát, ha nincs szellő.

<sup>9</sup> A latinak létezik külön szava az „exkluzív vagy” kifejezésére, ez az *aut*.

## Egy egyszerű tudásbázis

Most, hogy definiáltuk az ítéletkalkulus szemantikáját, létre tudunk hozni egy tudásbázist a wumpus világ számára. Az egyszerűség kedvéért csak a csapdákkal fogunk törödni, a wumpust magát feladatként az olvasóra hagyjuk. A tudás, amelyet most leírunk, elégsges ahhoz, hogy elvégezzük a 7.3. alfejezetben nem formálisan már elvégzett következtetést.

Először meg kell választanunk a szótárunkat az ítéletszimbólumainak megnevezéséhez. minden  $i, j$ -re:

- Legyen  $C_{i,j}$  igaz, ha csapda van  $[i, j]$ -ben.
- Legyen  $S_{i,j}$  igaz, ha szellő van  $[i, j]$ -ben.

A tudásbázis tartalmazza a következő mondatokat (mindegyiket felcímkeljük a kényelem kedvéért):

- Nincs csapda az  $[1, 1]$ -ben:

$$Sz_1: \neg C_{1,1}$$

- Egy négyzet akkor és csak akkor szellős, ha csapda van a szomszédos négyzetben. Ezt minden négyzetre vonatkozóan ki kell jelenteni. mi most csak a releváns négyzeteket tekintjük:

$$Sz_2: S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$$

$$Sz_3: S_{2,1} \Leftrightarrow (C_{1,1} \vee C_{2,2} \vee C_{3,1})$$

- Az eddigi mondatok minden wumpus világban igazak. Most hozzáadjuk a szellő érzetet az első két meglátogatott négyzetre abban a specifikus világban, ahol az ágens jelenleg tartózkodik, ami elvezet a 7.3. (b) ábrán látott szituációhoz.

$$Sz_4: \neg S_{1,1}$$

$$Sz_5: S_{2,1}$$

A tudásbázis most  $Sz_1$ -től  $Sz_5$ -ig tartalmaz mondatokat. Tekinthetjük ezt úgy is, mint egyetlen mondatot – az  $Sz_1 \wedge Sz_2 \wedge Sz_3 \wedge Sz_4 \wedge Sz_5$  konjunkciót –, mivel ez azt is kijelenti egyben, hogy minden egyes mondat is igaz.

## Következtetés

Emlékezzünk, hogy a logikai következtetés célja, hogy előntsük, hogy  $TB \models \alpha$  bizonyos  $\alpha$  mondatokra. Például, hogy a tudásbázis maga után vonzza-e a  $C_{2,2}$  állást. Az első algoritmusunk a következtetésre a vonzat definíciójának közvetlen megvalósítása lesz: vegyük sorba a modelleket, és ellenőrizzük, hogy  $\alpha$  igaz-e minden modellben, amelyben a  $TB$  igaz. Az ítéletlogikában a modellek az *igaz* és *hamis* értékek hozzárendelései minden egyes ítéletszimbólumhoz. Visszatérve a mi wumpus világbeli példánkhöz, a releváns ítéletszimbólumok az  $S_{1,1}, S_{1,2}, C_{1,1}, C_{1,2}, C_{2,1}, C_{2,2}, C_{3,1}$ . A 7 szimbólum  $2^7 = 128$  lehetséges modellt jelent, háromban ezek közül a  $TB$  igaz (7.9. ábra). Ebben a három modellben  $\neg C_{1,2}$  igaz, így nincsen csapda az  $[1, 2]$ -ben. Viszont a  $C_{2,2}$  a három mo-

debből kettőben igaz és egyben hamis, így még nem tudjuk megmondani, hogy van-e csapda [2, 2]-ben.

A 7.9. ábra precízebb formában megisméli a 7.5. ábrán illusztrált következetést. A 7.10. ábra egy általános algoritmust mutat a maga után vonzás eldöntésére az ítéletkalkulusban. Mint a VISSZALÉPÉSES-KERESÉS algoritmus a 115–116. oldalon, az IT-VONZAT? egy rekurzív felsorolást végez a változó hozzárendelések véges terén. Az algoritmus helyes, mivel közvetlenül a vonzat definícióját valósítja meg, és teljes, mivel bármely  $TB$ -on és  $\alpha$  mondaton működik, és minden sikeresen véget ér, hiszen véges számú modellett kell megvizsgálni.

$S_{1,1}$	$S_{1,2}$	$C_{1,1}$	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$	$C_{3,1}$	$Sz_1$	$Sz_2$	$Sz_3$	$Sz_4$	$Sz_5$	$TB$
$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$igaz$	$igaz$	$hamis$	$hamis$
$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$igaz$	$hamis$	$igaz$	$hamis$	$hamis$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$hamis$	$igaz$	$hamis$	$hamis$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$hamis$	$igaz$	$igaz$	$hamis$
$hamis$	$igaz$	$hamis$	$hamis$	$igaz$	$hamis$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$
$hamis$	$igaz$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$
$hamis$	$igaz$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$
$hamis$	$igaz$	$hamis$	$hamis$	$hamis$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$igaz$	$hamis$	$igaz$	$igaz$	$hamis$	$igaz$	$hamis$

**7.9. ábra.** A tudásbázis alapján épített igazságtábla látható az ábrán. A  $TB$  igaz, ha  $Sz_1$ -től és  $Sz_5$ -ig igaz, amely a 128 sorból csak 3-ban fordul elő. Mind a 3 sorban  $C_{1,2}$  hamis, tehát nincsen csapda az [1, 2]-ben. Viszont lehet, hogy van csapda [2, 2]-ben (bár lehet, hogy nincs).

```

function IT-VONZAT?(TB,  $\alpha$ ) returns igaz vagy hamis
  input: TB, a tudásbázis, egy ítéletkalkulus mondat
           $\alpha$ , a lekérdezés, egy ítéletkalkulus mondat

  szimbólumok  $\leftarrow$  TB-ban és  $\alpha$ -ban szereplő szimbólumok listája
  return IT-ELLENŐRIZ-MIND(TB,  $\alpha$ , szimbólumok, [])



---


function IT-ELLENŐRIZ-MIND(TB,  $\alpha$ , szimbólumok, modell) returns igaz vagy hamis
  if ÜRES(szimbólumok) then
    if IK-IGAZ?(TB, modell) then return IK-IGAZ?( $\alpha$ , modell)
    else return igaz
  else do
    P  $\leftarrow$  ELSÓ(szimbólumok); maradék  $\leftarrow$  MARADÉK(szimbólumok)
    return IT-ELLENŐRIZ-MIND(TB,  $\alpha$ , maradék, KIEGÉSZÍT(P, igaz, modell) and
          IT-ELLENŐRIZ-MIND(TB,  $\alpha$ , maradék, KIEGÉSZÍT(P, igaz, modell)))

```

**7.10. ábra.** Egy igazságátbla felsoroló algoritmus ítéletkalkulus állítás vonzatának elődtöntésére. Az IT az igazságátbláját jelöli. A **IK-IGAZ?** igazat ad vissza, ha a mondatot tartalmazza a modell. A **modell** változó reprezentál egy részleges modellt, egy hozzárendelést a változók egy részéhez. A **Kiegészít( $P$ , igaz, model)** függvény egy új részleges modellt ad vissza, amelyben  $P$  értéke igaz.

 Természetesen a „véges számosság” nem minden jelenti a „néhányat”. Ha a  $TB$  és az  $\alpha$  mondat összesen  $n$  szimbólumot tartalmaz, akkor  $2^n$  modell létezik. Így az algoritmus időigénye  $O(2^n)$ . (A tárígyene csak  $O(n)$ , mivel a felsorolás mélységi jellegű.) A fejezet későbbi részében fogunk látni olyan algoritmusokat, amelyek sokkal hatékonyabbak a gyakorlatban. Sajnos minden ismert, az ítéletlogikára vonatkozó következtetési algoritmusnak a legrosszabb esetre vonatkozó komplexitása exponenciálisan függ a bemenetek számától. Nem remélhetjük, hogy ennél hatékonyabbak is lehetünk, mivel az ítéletlogikában a maga után vonzás co-NP-teljes (lásd A) függelék).

## Ekvivalencia, érvényesség és kielégíthetőség

Mielőtt belemerülnénk a logikai következtetés részleteinek tárgyalásába, szükségünk van néhány további, a vonzattal kapcsolatos fogalomra. Mint a vonzat, ezek a fogalmak is a logika minden formájára alkalmazhatók, de legjobban egy konkrét logikán – mint amilyen az ítéletkalkulus – lehet illusztrálni őket.

Az első fogalom a **logikai ekvivalencia (logical equivalence)**: két mondat, az  $\alpha$  és a  $\beta$  mondatok logikailag ekvivalensek, ha ezek a mondatok a modellek ugyanazon hal-mazán igazak. Ezt úgy jelöljük, hogy  $\alpha \Leftrightarrow \beta$ . Például (igazságáblákat használva) könnyen megmutathatjuk, hogy  $P \wedge Q$  és  $Q \wedge P$  logikailag ekvivalensek; további ekvivalenciák láthatók a 7.11. ábrán. Az ekvivalenciák nagyon hasonló szerepet játszanak a logikában, mint az aritmetikai identitások a közönséges matematikában. Az ekvivalencia alternatív definíciója a következő: bármely két  $\alpha, \beta$  mondatra,

$$\alpha \equiv \beta \text{ akkor és csak akkor, ha } \alpha \models \beta \text{ és } \beta \models \alpha$$

(Emlékeztetőül, a  $\models$  a maga után vonzást jelöli.)

A második fogalom, amelyre szükségünk lesz az **érvényesség (validity)**. Egy mondat érvényes, ha igaz minden modellben. Például a  $P \vee \neg P$  mondat érvényes. Az érvényes mondatokat **tautológiáknak (tautologies)** is szokták nevezni, ezek szükségszerűen igazak és így feleslegesek. Mivel az *Igaz* mondat igaz minden modellben, minden érvényes mondat logikailag ekvivalens az *Igaz* mondattal.

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	$\wedge$ kommutativitás
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	$\vee$ kommutativitás
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	$\wedge$ asszociativitás
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	$\vee$ asszociativitás
$\neg(\neg \alpha) \equiv \alpha$	kettős negáció kiküszöbölés
$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$	kontrapozíció
$(\alpha \Rightarrow \beta) \equiv (\neg \beta \vee \alpha)$	implikáció kiküszöbölés
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	ekvivalencia kiküszöbölés
$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	$\wedge$ disztributivitás
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	$\vee$ disztributivitás

7.11. ábra. Standard logikai ekvivalenciák. Az  $\alpha, \beta, \gamma$  szimbólumok tetszőleges ítéletkalkulus mondatokat jelölnek.

Mire jók akkor az érvényes mondatok? A vonzatokra vonatkozó definícióból alapján levezethetjük a **dedukcióelméletet (deduction theorem)**, amelyet már az ókori görögök is ismertek:

**Bármely  $\alpha$  és  $\beta$  mondatra  $\alpha \models \beta$  akkor és csakis akkor, ha az  $(\alpha \Rightarrow \beta)$  mondat érvényes.**

(A 7.4. feladatban ennek a bizonyítását kérjük.) A 7.10. ábrán látható következtetési algoritmust úgy tekinthetjük, mint a  $(TB \Rightarrow \alpha)$  érvényességének ellenőrzését. Másik oldalról nézve minden érvényes implikáció leír egy legitim következtést.



Az utolsó fogalom, amelyre szükségünk lesz a **kielégíthetőség (satisfiability)**. Egy mondat kielégíthető, ha igaz *néhány* modellben. Például a korábban bemutatott tudásbázis, az  $(Sz_1 \wedge Sz_2 \wedge Sz_3 \wedge Sz_4 \wedge Sz_5)$ , kielégíthető, mert van három olyan modell, amelyben igaz, ahogy ezt a 7.9. ábrán megrutattuk. Ha egy  $\alpha$  mondat igaz az  $m$  modellben, akkor azt mondjuk, hogy  $m$  **kielégíti (satisfies)  $\alpha$ -t**, vagy  $m$  **egy modellje  $\alpha$ -nak**. A kielégíthetőség ellenőrizhető úgy, hogy a lehetséges modellek felsoroljuk mindenkorban, amíg nem találunk egyet, amely kielégíti a mondatot. A mondatok kielégíthetőségének meghatározása az ítéletlogikában az első olyan probléma volt, amelyről bebizonyították, hogy NP-teljes.

Számos probléma a számítástechnikában valójában kielégíthetőségi probléma. Például a kényszerkielégítési problémák az 5. fejezetben alapvetően arra kérdeznek rá, hogy a kényszerek kielégíthetők-e néhány hozzárendeléssel. Megfelelő transzformációk után a keresési problémák szintén megoldhatók a kielégíthetőség ellenőrzésével. Az érvényesség és a kielégíthetőség természetesen kapcsolatban vannak:  $\alpha$  érvényes akkor és csakis akkor, ha  $\neg\alpha$  nem kielégíthető; és fordítva,  $\alpha$  akkor és csakis akkor kielégíthető, ha  $\neg\alpha$  nem érvényes. A következő hasznos eredményt is ismerjük:



**$\alpha \models \beta$  akkor és csakis akkor, ha az  $(\alpha \wedge \neg\beta)$  nem kielégíthető**

A  $\beta$  mondat bizonyítása  $\alpha$  alapján, az  $(\alpha \wedge \neg\beta)$  kielégíthetetlenségének ellenőrzésével, pontosan megfelel a szokásos matematikai bizonyítási technikának a *redukcio ad absurdumnak* (szó szerint „redukció egy abszurd dologra”). Szokták ezt **megcáfolás (refutation)** általi bizonyításnak is nevezni vagy bizonyítás **ellentmondás (contradiction)** által. Feltételezzük, hogy a  $\beta$  mondat hamis, és megrutatjuk, hogy ez a feltételezés ellentmondásra vezet az ismert  $\alpha$  axiómákkal. Ez az ellentmondás pontosan azt jelenti, mint amikor azt mondjuk, hogy az  $(\alpha \wedge \neg\beta)$  mondat kielégíthetetlen.

## 7.5. AZ ÍTELETKALKULUS KÖVETKEZTETÉSI MINTÁI

Ez az alfejezet végigveszi a következtetés standard mintáit, amelyek alkalmazhatók arra, hogy következmények láncolatait vezethessük le, amelyek elvezetnek a kívánt célt. Ezeket a következtetési mintákat **következtetési szabályoknak (inference rules)** hívjuk. A legjobban ismert szabály a **Modus Ponens** és a következőképpen írható le:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

A jelölés azt jelenti, hogy ha bármikor adott egy  $\alpha \Rightarrow \beta$  formájú mondat és adott egy  $\alpha$ , akkor a  $\beta$  mondat ebből következik. Például a  $(WumpusElőrefelé \wedge WumpusEl) \Rightarrow Lövés$  és a  $(WumpusElőrefelé \wedge WiumpusEl)$  adott, akkor a *Lövés* kikövetkeztethető.

Egy másik hasznos következtetési szabály az **És-kiküszöbölés (And-Elimination)**, ami azt mondja ki, hogy egy konjunkcióból bármely konjunkt kikövetkeztethető:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Például abból, hogy (*WumpusElőrefelé*  $\wedge$  *WumpusÉl*) a *WumpusÉl* kikövetkeztethető.

Tekintettel az  $\alpha$  és  $\beta$  lehetséges igazságértékeire, könnyen megmutatható, hogy a Modus Ponens és az És-kiküszöbölés helyes egyszer és mindenkorra. Ezek a szabályok felhasználhatók bármely konkrét esetben, ahol alkalmazhatók, helyes következtetéseket eredményezve anélkül, hogy fel kellett volna sorolnunk a modelleket.

A 7.11. ábrán található összes logikai ekvivalencia használható következtetési szabályként. Például az ekvivalencia kiküszöbölés két következtetési szabályt eredményez:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{és} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Nem minden következtetési szabály működik minden irányban, mint ezek. Például nem futtathatjuk a Modus Ponent ellenkező irányban, hogy megkapjuk  $\alpha \Rightarrow \beta$ -t és  $\alpha$ -t a  $\beta$ -ból.

Nézzük meg, hogyan használhatjuk ezeket a következtetési szabályokat és ekvivalenciákat a wumpus világban. Az Sz<sub>1</sub>, ..., Sz<sub>5</sub> szabályokból álló tudásbázisból indulunk ki, és megmutatjuk, hogy hogyan bizonyíthatjuk a  $\neg C_{1,2}$ -t azaz, hogy nincs csapda az [1, 2]-ben. Először alkalmazzuk az ekvivalencia kiküszöbölést Sz<sub>2</sub>-re, és így kapjuk, hogy

$$Sz_6: (S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})) \wedge ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

Ezután alkalmazzuk az És-kiküszöbölést az Sz<sub>6</sub>-ra, és így kapjuk, hogy

$$Sz_7: ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

A kontrapozíció logikai ekvivalenciát alkalmazva:

$$Sz_8: (\neg S_{1,1} \Rightarrow \neg(C_{1,2} \vee C_{2,1}))$$

Most alkalmazhatjuk a Modus Ponent az Sz<sub>8</sub>-ra és az Sz<sub>4</sub> érzetre (például a  $\neg S_{1,1}$ ) és így kapjuk:

$$Sz_9: \neg(C_{1,2} \vee C_{2,1})$$

Végül alkalmazzuk a De Morgan-szabályt, amely a konklúziót adja:

$$Sz_{10}: \neg C_{1,2} \wedge \neg C_{2,1}$$

Tehát sem az [1, 2], sem a [2, 1] négyzet nem tartalmaz csapdát.

Az előző levezetést – következtetési szabályok egy sorozatát – **bizonyításnak (proof)** nevezzük. A bizonyítás megtalálása pontosan olyan, mint megoldást találni egy keresési problémára. Valójában ha a következtetési szabályok összes lehetséges alkalmazásának generálására egy új állapotámenet-függvényt definiálnánk, akkor minden, a 3. és 4. fejezetbeli kereső algoritmust felhasználhatnánk a bizonyítás megtalálására. A bizonyítás keresése tehát egy alternatívája a modellek felsorolásának. A keresés haladhat előrefelé a kezdeti tudásbázisból kiindulva, alkalmazva a következtetési szabályokat a célmon-

dat levezetéséhez, vagy mehet visszafelé a célmondatból, megpróbálva megtalálni a következtetési szabályoknak olyan láncolatát, amely a kiindulási tudásbázisra alkalmazható szabályokból indul. A fejezetben később bemutatunk két olyan algoritmuscsaládot, amelyek ezeket a technikákat használják.

Az a tény, hogy a következtetés az ítéletlogikában NP-teljes, azt sugallja, hogy a legrosszabb esetet tekintve a bizonyítások keresése sem hatékonyabb a modellek felsorolásánál. Számos gyakorlati esetben azonban, *a bizonyítás megtalálása sokkal hatékonyabb lehet, egyszerűen azért, mert képes figyelmen kívül hagyni az irreleváns állításokat, figyeltenél attól, hogy hány van belőlük*. Például az előző bizonyítás, amely elvezetett a  $\neg C_{1,2} \wedge \neg C_{2,1}$  mondathoz, nem említi az  $S_{2,1}$ ,  $C_{1,1}$ ,  $C_{2,2}$  vagy a  $C_{1,2}$  állításokat. Ezeket azért lehet figyelmen kívül hagyni, mert a  $C_{1,2}$  célállítás csak az  $Sz_2$ -ben jelenik meg, az  $Sz_2$ -ben szereplő egyéb állítások pedig csak az  $Sz_2$ -ben és az  $Sz_4$ -ben, így az  $Sz_1$ ,  $Sz_3$  és  $Sz_5$  szabályoknak nincs kihatásuk a bizonyításra. Ugyanez maradna a helyzet, ha még millió szabályt hozzáadnánk a tudásbázishoz, miközben az igazságátábla algoritmust ezzel ellentében elárasztaná a modellek exponenciális robbanása.

A logikai rendszereknek ez a tulajdonsága valójában egy sokkal alapvetőbb jellegzetességgükől, a **monotonitásból (monotonicity)** következik. A monotonitás azt mondja ki, hogy a vonzatmondatok halmaza csak bővülhet, ha a tudásbázishoz információt adunk hozzá.<sup>10</sup>

ha       $TB \models \alpha$       akkor       $TB \wedge \beta \models \alpha$

Például feltételezzük azt, hogy a tudásbázis tartalmaz egy új  $\beta$  állítást, amely azt mondja ki, hogy pontosan 8 csapda van a világban. Ez a tudás segítheti az ágenst további konklúziók levezetésében, de nem teheti érvénytelenné egyik korábban kikövetkeztetett  $\alpha$  konklúziót sem – így azt a konklúziót sem, hogy nincsen csapda az [1, 2]-ben. A monotonitás azt jelenti, hogy a következtetési szabályok bármikor alkalmazhatók, ha a megfelelő premisszák megtalálhatók a tudásbázisban – a szabály konklúziójának következnie kell, *figyeltenél attól, hogy mi más is van még a tudásbázisban*.

## Rezolúció

Megmutattuk, hogy az eddig ismertetett következtetési szabályok helyesek, de nem tárgyalunk az ezeket használó következtetési algoritmusok *teljességének* kérdését. A keresési algoritmusok, mint az iteratívan mélyülő keresés (118. oldal) teljesek abban az értelemben, hogy meg fogják találni az elérendő célt. Ha azonban a rendelkezésre álló szabályok hiányosak, akkor a cél nem érhető el – nem létezik olyan bizonyítás, amely ezeket a szabályokat használja. Például ha kivennénk az ekvivalencia kiküszöbölés szabályt, az előző fejezetbeli bizonyítás nem futna végig. Ez a fejezet egyetlen következtetési szabályt mutat be, a **rezolúciót (resolution)**, amelynek alkalmazása, párosítva bármelyik keresési módszerrel, egy teljes következtetési algoritmust eredményez.

<sup>10</sup> A **nemmonoton logikák (nonmonotonic logics)**, amelyek megszegik a monotonitás tulajdonságot, az emberi érvelésnek azt a szokványos tulajdonságát jelentik meg, amikor valaki meggondolja magát. Ezeket a 10.7. alfejezetben tárgyaljuk.



Először a rezolúciós szabály egy egyszerű változatát fogjuk használni a wumpus világban. Nézzük meg a 7.4. (a) ábrához vezető lépéseket: az ágens visszafordul a [2, 1]-ből az [1, 1]-be, és innen megy az [1, 2]-be, ahol szellőt érez, de bűzt nem. A következő tényeket adjuk hozzá a tudásbázishoz:

$$Sz_{11}: \neg S_{1,2}$$

$$Sz_{12}: S_{1,2} \Leftrightarrow (C_{1,1} \vee C_{2,2} \vee C_{1,3})$$

Ugyanazzal a folyamattal, amely az  $Sz_{10}$ -hez vezetett korábban, most le tudjuk vezetni, hogy nincs csapda a [2, 2]-ben és az [1, 3]-ban (emlékezzünk, hogy már tudjuk, hogy az [1, 1] csapdamentes):

$$Sz_{13}: \neg C_{2,2}$$

$$Sz_{14}: \neg C_{1,3}$$

Az  $Sz_3$ -ra is alkalmazhatjuk az ekvivalencia kiküszöbölést, amelyet egy Modus Ponens követ az  $Sz_5$ -re, hogy megkapjuk azt a tényt, hogy csapda van az [1, 1], a [2, 2] vagy a [3, 1] négyzetekben:

$$Sz_{15}: C_{1,1} \vee C_{2,2} \vee C_{3,1}$$

Most következik a rezolúció szabály első alkalmazása: a  $\neg C_{2,2}$  literál az  $Sz_{13}$ -ban rezolvál a  $C_{2,2}$  literállal az  $Sz_{15}$ -ben, amely ezt adja:

$$Sz_{16}: C_{1,1} \vee C_{3,1}$$

Magyaráról, ha van egy csapda az [1, 1], [2, 2], [3, 1] négyzetek egyikében, és ez a csapda nem a [2, 2]-ben van, akkor ez az [1, 1]-ben vagy a [3, 1]-ben van. Hasonlóan a  $\neg C_{1,1}$  literál az  $Sz_1$ -ben rezolvál a  $C_{1,1}$  literállal az  $Sz_{16}$ -ban, amiből adódik:

$$Sz_{17}: C_{3,1}$$

Tehát, ha van egy csapda az [1, 1]-ben vagy a [3, 1]-ben, és ez a csapda nem az [1, 1]-ben van, akkor ez a [3, 1]-ben van. Ez az utolsó két következtetési lépés példa az **egységrezolúció (unit resolution)** következtetési szabályra.

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k}$$

ahol  $\ell$  egy literál,  $\ell_i$  és  $m$  pedig **kiegészítő literálok (complementary literals)** (például az egyik negálja a másiknak). Tehát az egységrezolúció vesz egy **klózt (clause)** – literálok diszjunkcióját – meg egy literált, és létrehoz egy új klózt. Vegyük észre, hogy egy egyedi literált tekintethetünk egy literál diszjunkciójának, amit szoktak **egységlőznek (unit clause)** is nevezni.

Az egységrezolúció szabálya általánosítható teljes **rezolúciós (resolution)** szabályá,

$$\frac{\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

ahol  $\ell_i$  és  $m_j$  kiegészítő literálok. Ha csak kettő hosszúságú klózokkal foglalkozunk, akkor ezt írhatjuk:

$$\frac{\ell_1 \vee \ell_2, \neg \ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Azaz a rezolúció vesz két klózt, és létrehoz egy új klózt, amely tartalmaz minden literált az eredeti két klózból, kivéve a kiegészítő literálokat. Például:

$$\frac{P_{1,1} \vee P_{3,1}, \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Van még egy technikai aspektusa a rezolúciós szabálynak: az eredményklóznak minden literálnak csak egy példányát kell tartalmaznia.<sup>11</sup> A literálok többszörös példányainak kivonását **faktorálás (factoring)** hívják. Például ha rezolváljuk az  $(A \vee B)$ -t a  $(A \vee \neg B)$ -vel, akkor  $(A \vee A)$ -t kapunk, amelyet redukálhatunk egyszerűen  $A$ -ra.

A rezolúciós szabály *helyessége* egyszerűen belátható, ha megvizsgáljuk az  $\ell_i$  literált. Ha  $\ell_i$  igaz, akkor  $m_j$  hamis, és így  $m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n$ -nek igaznak kell lennie, mert  $m_1 \vee \dots \vee m_n$  adott. Ha  $\ell_i$  hamis, akkor  $\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k$  igaz kell, hogy legyen, mivel  $\ell_i \vee \dots \vee \ell_k$  adott. Így  $\ell_i$  akár igaz, akár hamis, az egyik vagy a másik konklúzió áll, pontosan úgy, ahogy a szabály ezt kimondja.

Ami még meglepőbb a rezolúciós szabályal kapcsolatban, hogy a rezolúció alapjául szolgál teljes következtetési algoritmusok egy családjának. *Bármely keresési algoritmus, a rezolúciós szabályt alkalmazva, az ítéletlogikában képes levezetni bármilyen konklúziót, amely vonzata a tudásbázisnak.* Egy figyelmezhetős: a rezolúció csak egy speciális értelemben teljes. Ha adott, hogy  $A$  igaz, akkor nem tudjuk a rezolúciót arra használni, hogy levezzük az  $A \vee B$  konzekvenciát. Azonban tudjuk arra használni a rezolúciót, hogy megválaszoljuk azt a kérdést, hogy  $A \vee B$  igaz-e. Ezt megcáfolási teljességnak (refutation completeness) nevezik, ami azt jelenti, hogy a rezolúció minden használható arta, hogy megerősítsünk vagy megcáfoljunk egy mondatot, de nem alkalmazható az igaz mondatok felsorolására. A következő két alfejezet azt magyarázza el, hogy a rezolúció hogyan oldja ezt meg.

### Konjunktív normál forma

A rezolúciós szabály csak literálok diszjunkciójára alkalmazható, így úgy tűnhet, hogy ez csak olyan tudásbázis és lekérdezés esetében érdekes, amelyek ilyen diszjunkciókat tartalmaznak. Akkor hogyan vezet ez egy teljes következtetési eljáráshoz az egész ítéletkalkulus számára? A válasz az, hogy minden ítéletkalkulus mondat logikailag ekvivalens literálok diszjunkciójának konjunktívával. Egy mondatot, amelyet literálok diszjunkciójának konjunktívával fejezünk ki, konjunktív normál formájúnak (conjunctive normal form) vagy CNF formájúnak nevezünk. A későbbiekben az is hasznos lesz, ha ennek egy korlátosabb családját, a  $k$ -CNF mondatokat tekintjük. Egy mondat a  $k$ -CNF-be tartozik, ha pontosan  $k$  literál van a klózokban.

$$(\ell_{1,1} \vee \dots \vee \ell_{1,k}) \wedge \dots \wedge (\ell_{n,1} \vee \dots \vee \ell_{n,k})$$

<sup>11</sup> Ha a klózt literálok halmazának tekintjük, akkor ezt a korlátozást automatikusan figyelembe vettük. A halmazt jelölést alkalmazva a klózokra a rezolúciós szabály sokkal tisztább lesz egy újabb jelölés bevezetésének árán.

Ki fog derülni, hogy minden mondat transzformálható 3-CNF mondattá úgy, hogy azonos marad a modellek halmaza.

Ahelyett hogy bizonyítanánk ezeket a kijelentéseket (lásd 7.10. feladat), leírunk egy egyszerű konvertáló eljárást. Az eljárást az Sz2 szabály, az  $S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$  mondat CNF-re konvertálásával illusztráljuk. A lépések a következők:

1. Küszöböljük ki a  $\Leftrightarrow$  összekötőjelet, helyettesítve az  $\alpha \Leftrightarrow \beta$ -t ( $\alpha \Rightarrow \beta$ )  $\wedge (\beta \Rightarrow \alpha)$ -vel:

$$(S_{1,1} \Rightarrow (C_{1,2} \vee C_{2,1})) \wedge ((C_{1,2} \vee C_{2,1}) \Rightarrow S_{1,1})$$

2. Kiküszöböljük az  $\Rightarrow$  összekötőjelet, kicserélve  $\alpha \Rightarrow \beta$ -t  $\neg\alpha \vee \beta$ -ra:

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge (\neg(C_{1,2} \vee C_{2,1}) \vee S_{1,1})$$

3. A CNF megkívánja, hogy a  $\neg$  csak literálokra vonatkozzon, így most „a-t beljebb mozgatjuk” a következő 7.11. ábrán felírt ekvivalenciák ismételt alkalmazásával:

$$\neg(\neg\alpha) \equiv \alpha \quad \text{kettős negáció kiküszöbölés}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$

A példában csak az utolsó szabály alkalmazására van szükségünk:

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge ((\neg C_{1,2} \wedge \neg C_{2,1}) \vee S_{1,1})$$

4. Most van egy mondatunk, amelyben egymásba ágyazott  $\wedge$  és  $\vee$  operátorok vannak alkalmazva a literálokra. A 7.11. ábrán bevezetett disztributivitás szabályt alkalmazzuk a  $\vee$  operátorokra az  $\wedge$  felett mindenhol, ahol lehetséges.

$$(\neg S_{1,1} \vee C_{1,2} \vee C_{2,1}) \wedge (\neg C_{1,2} \vee S_{1,1}) \wedge (\neg C_{2,1} \vee S_{1,1})$$

Az eredeti mondat most már CNF-ben van, három klóz konjunkciójaként. Bár sokkal nehezebb olvasni ezt a formát, de használható a rezolúciós eljárás bemeneteként.

## A rezolúció algoritmus

A rezolúción alapuló következtetési eljárások az ellentmondásokra vezető bizonyítások elvén működnek, ahogy azt a 7.4. alfejezet végén tárgyaltuk. Tehát annak megmutatásához, hogy  $TB \models \alpha$ , azt mutatjuk meg, hogy a  $(TB \wedge \neg\alpha)$  kielégíthetetlen. Ezt az ellentmondás bizonyításával végezzük el.

A rezolúció algoritmust mutatja a 7.12. ábra. Először a  $(TB \wedge \neg\alpha)$ -t konvertáljuk CNF formára. Majd a rezolúciós szabályt alkalmazzuk a létrejövő klózokra. minden egyes párt, amely kiegészítő literálokat tartalmaz, rezolválunk, hogy egy új klózot hozunk létre, amelyet hozzádunk a halmazhoz, ha még nem volt jelen. A folyamat addig folytatódik, amíg a következő két dolog közül valamelyik meg nem történik:

- nincs több új klóz, amit hozzá lehet adni, ilyen esetben  $\alpha$  nem vonzza maga után  $\beta$ -t.
- a rezolúció alkalmazása egy üres klózra vezet, amely esetben  $\alpha$ -nak vonzata  $\beta$ .

Az üres klóz – egy diszjunkt nélküli diszjunkció – ekvivalens a *Hamis* értékkel, mert a diszjunkció akkor igaz csak, ha legalább az egyik diszjunkt igaz. Úgy is beláthatjuk,

```

function IK-REZOLVÁLÁS( $TB, \alpha$ ) returns igaz vagy hamis
input:  $TB$ , a tudásbázis, egy ítéletkalkulus mondat
 $\alpha$ , a lekérdezés, egy ítéletkalkulus mondat

 $klózok \leftarrow a TB \wedge \neg\alpha$  mondatot reprezentáló CNF formájú klózok halmaza
 $új \leftarrow \{\}$ 
loop do
  for each  $C_i, C_j$  in  $klózok$  do
     $rezolvensek \leftarrow$  IK-REZOLVÁLÁS( $C_i, C_j$ )
    if  $rezolvensek$  tartalmazzák az üres klózt then return igaz
     $új \leftarrow új \cup rezolvensek$ 
    if  $új \subseteq klózok$  then return hamis
   $klózok \leftarrow klózok \cup új$ 

```

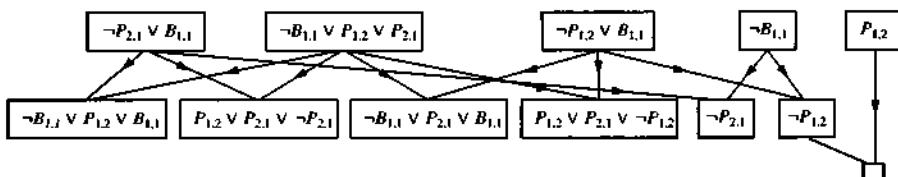
7.12. ábra. Egy egyszerű rezolúciós algoritmus az ítéletkalkulushoz. Az IK-REZOLVÁLÁS a két bemenetként megkapott állítás rezolválásából származó összes lehetséges klóz halmazát adja vissza.

hogy az üres klóz ellentmondást reprezentál, hogy megfigyeljük azt, hogy az üres klóz két kiegészítő egységeklóz, mint amilyen az  $S$  és  $\neg S$ , rezolválásából származik.

Alkalmazzuk a rezolúciós eljárást egy nagyon egyszerű következtetésre a wumpus világban. Amikor az ágens az [1,1]-ben van, akkor nincs szellő, tehát nincs csapda a szomszédos négyzetekben. Az ennek megfelelő tudásbázis:

$$TB = S_{\bar{z}_2} \wedge S_{\bar{z}_4} = (S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})) \wedge \neg S_{1,1}$$

és mi bizonyítani szeretnénk  $\alpha$ -t, ami mondjuk  $\neg C_{1,2}$ . Amikor konvertáljuk a  $(TB \wedge \neg\alpha)$ -t CNF formára, akkor a 7.13. ábra felső részén látható klózokat kapjuk. Az ábra második sora mutatja az összes klózt, amit az első sor párhajainak rezolválásából kaptunk. És akkor, amikor a  $C_{1,2}$ -t rezolváljuk  $\neg C_{1,2}$ -vel megkapjuk az üres klózt, amit egy kis négyzet jelöl. A 7.13. ábra megvizsgálása azt mutatja, hogy számos rezolúciós lépés felesleges. Például az  $S_{1,1} \vee \neg S_{1,1} \vee C_{1,2}$  ekvivalens az *Igaz*  $\vee C_{1,2}$ -vel, ami ekvivalens az *Igaz*-val. Levezetni, hogy az *Igaz* az *igaz*, nem igazán hasznos. Így bármely olyan klóz, amelyben két kiegészítő klóz szerepel, figyelmen kívül hagyható.



7.13. ábra. Az IK-REZOLÚCIÓ algoritmus részleges alkalmazása egy egyszerű wumpus világbeli következtetésre. A felső sor első négy klóza alapján származtatjuk a  $\neg P_{1,2}$ -t.

### A rezolúció teljessége

Hogy befejezzük a rezolúció tárgyalását, most megmutatjuk, hogy az IK-REZOLÚCIÓ algoritmus teljes. Ahhoz, hogy ezt megtehessük, hasznos bevezetni az  $S$  klózok halma-

zának rezolúciós lezártját (**resolution closure**),  $RC(S)$ -t, ami az összes olyan klóz halmaza, amely levezethető a rezolúció ismételt alkalmazásaival az  $S$  halmazbeli elemekből és ezek leszármazottaiból. Az IK-REZOLÚCIÓ a rezolúciós lezárás halmaz elemeit sorolja fel a klózok változóban. Könnyű belátni, hogy  $RC(S)$ -nek végesnek kell lennie, mert véges sok különböző klóz lehetséges konstruálni az  $P_1, \dots, P_k$  szimbólumokból, amelyek elemei  $S$ -nek. (Vegyük észre, hogy ez nem volna igaz a faktorálási lépés nélkül, amely megszünteti a literálok többszörözését.) Így az IK-REZOLÚCIÓ minden terminálódik.

Az ítéletlogikában a rezolúció teljességi tételeit **alap rezolúciós tételek (ground resolution theorem)** nevezzük.

Ha klózok egy halmaza kielégíthetetlen, akkor ezeknek a klózoknak a rezolúciós lezártja tartalmazza az üres klózt.

A tételek az ellentéjének demonstrálásával bizonyítjuk: ha az  $RC(S)$  lezártja nem tartalmaz üres halmazt, akkor  $S$  kielégíthető. Valójában egy modellt konstruálunk  $S$ -nek a megfelelő  $P_1, \dots, P_k$  igazság értékekkel. A konstrukciós eljárás a következő:

Minden  $i$ -re 1-től  $k$ -ig,

- Ha létezik egy klóz  $RC(S)$ -ben, amely tartalmazza a  $\neg P_i$  literált úgy, hogy minden más literál hamis a választott  $P_1, \dots, P_{i-1}$  hozzárendelés szerint, akkor rendeljünk *hamis* értéket  $P_i$ -hez.
- Egyébként rendeljünk *igaz* értéket  $P_i$ -hez.

Az maradt hátra, hogy megmutassuk, hogy ez a  $P_1, \dots, P_k$  hozzárendelés modellje  $S$ -nek, feltéve, hogy  $RC(S)$  zárt a rezolúcióra, és nem tartalmazza az üres klózt. Ennek bizonyítását meghagyjuk feladatnak.

## Előre- és hátrafelé láncolás

A rezolúciót a teljesség tulajdonsága igen fontos következtetési módszerre teszi. Számos gyakorlati esetben azonban, a rezolúció teljes erejére nincs szükség. A valósvilág-beli tudásbázisok gyakran a klózoknak csak egy, **Horn-klózoknak (Horn clause)** nevezett korlátozabb fajtáját tartalmazzák. A Horn-klóz literálok olyan diszjunkciója, amelyek közül legfeljebb egy pozitív. Például a  $(\neg P_{1,1} \vee \neg Szellő \vee S_{1,1})$  klóz, ahol  $P_{1,1}$  jelenti, hogy az ágens pozíciója az [1, 1], egy Horn-klóz, míg az  $(S_{1,1} \vee C_{1,2} \vee C_{2,1})$  nem az.

A korlátozás, hogy csak egy pozitív literál lehet, egy kissé önkényesnek és érdektelennek tűnhet, de valójában nagyon fontos, három ok miatt is:

1. minden Horn-klóz felírható egy implikációként is, amelynek premisszája pozitív literálok konjunkciója, és konklúziója egyetlen pozitív literál (lásd 7.12. feladat). Például a  $(P_{1,1} \vee Szellő \vee S_{1,1})$  Horn-klóz átírható a  $(P_{1,1} \wedge Szellő) \Rightarrow S_{1,1}$  implikációra. Ebben az utóbbi formájában sokkal könnyebb olvasni a mondatot: azt mondja ki, hogyha az ágens az [1, 1]-ben van, és szellő érzékelhető ott, akkor az [1, 1] szellős. Számos területen az emberek számára könnyű a tudást ilyen mondatok formájában olvasni és leírni.

A Horn-klózokat, mint amilyen az előbbi, vagyis amelyben *pontosan* egy pozitív literál van, **határozott klózoknak (definite clauses)** nevezik. A pozitív literál a fej (**head**), míg a negatív literálok alkotják a klóz testét (**body**). Egy negatív literálok

nélküli határozott klóz egyszerűen kijelent egy adott állítást, amit gyakran **ténynek** (*fact*) neveznek. A határozott klózok formálják a **logikai programozás** (*logic programming*) alapját, amelyet a 9. fejezetben tárgyalunk. Egy pozitív literálok nélküli Horn-klóz felírható, mint egy olyan implikáció, amelynek a konklúziója a *Hamis* literál. Például a  $(\neg W_{1,1} \vee \neg W_{1,2})$  klóz – a wumpus nem lehet mind az [1,1], mind az [1,2] mezőben, ekvivalens a  $W_{1,1} \wedge W_{1,2} \Rightarrow \text{Hamis}$  mondattal. Az ilyen mondatokat **integritás kényszereknek** (*integrity constraints*) nevezik az adatbázis-kezelés területén, ahol adathibák jelzésére használják. A következőkben bemutatásra kerülő algoritmusokban az egyszerűség kedvéért feltételezzük, hogy a tudásbázis csak határozott klózokat tartalmaz, és nincsenek benne integritáskényszerek. Azt mondjuk, hogy ezek a tudásbázisok Horn-formában vannak.

2. A Horn-klózokon történő következtetés történhet az **előrefelé láncolás** (*forward chaining*) vagy a **hátrafelé láncolás** (*backward chaining*) algoritmusokkal, amelyeket a következőkben elmagyarázunk. Mindkét algoritmus igen természetes, a következtetési lépések nyilvánvalóak és egyszerűen követhetők az emberek számára.
3. A maga után vonzás kérdésének eldöntéséhez szükséges idő lineárisan függ a tudásbázis méretétől.

```

function IK-EL-VONZAT?(TB, q) returns igaz vagy hamis
inputs: TB, a tudásbázis, ítéletkalkulus Horn-klózok halmaza
         q, a lekérdezés, egy ítéletkalkulus szimbólum
lokális változók: számláló, egy tábla, klózokkal indexelve, kezdetben a premisszák száma
                      következmények, egy tábla, szimbólumokkal indexelve, kezdetben minden
                      elem hamis
                      agenda, szimbólumok lista, kezdetben azok a TB-beli szimbólumok, amelyekről
                      ismerjük, hogy igazak

while agenda nem üres do
  p ← ÉLEJE(agenda)
  if p = q then return igaz
  unless következmények [p] ← igaz
    következmények [p] ← igaz
    for each c Horn-klóz, amelyben a p premissza szerepel do
      csökkent számláló [c]
      if számláló [c] = 0 then
        Hozzáad(FEJ[c], agenda)
  return hamis

```

**7.14. ábra.** Előrefelé láncolás algoritmus az ítéletkalkulus számára. Az *agenda* tartalmazza azokat a szimbólumokat, amelyek ismerten igazak, de még nem „dolgozták fel” őket. A *számláló* tábla követi, hogy az egyes implikációknak hány premisszája ismeretlen. Ha az agendáról egy új *p* szimbólumot feldolgozunk, minden olyan implikáció számlálója csökken egyelőre, amelynek premisszájában *p* megjelenik. (Ez megoldható konstans időben, ha a *TB* megfelelően indexelt.) Ha a számláló eléri a nullát, azaz az implikáció minden premisszája ismert, akkor az implikáció konklúzióját hozzá lehet adni az agendához. Végül szükségünk van arra, hogy kövessük melyik szimbólumot dolgoztuk már fel; egy kikövetkeztetett szimbólumot nem kell hozzáadni az agendához, ha azt már korábban sikeresen feldolgoztuk. Így elkerülhető a redundáns munka, és megakadályozza a végtelen ciklusok kialakulását, amelyet olyan implikációk okozhatnak, mint a  $P \Rightarrow Q$  és  $Q \Rightarrow P$ .

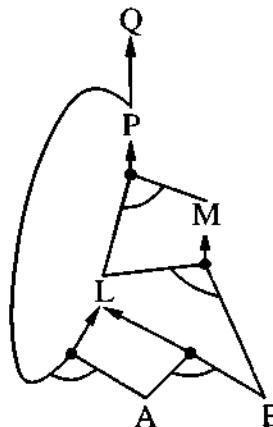
Ez az utolsó tény egy kellemes meglepetés. Azt jelenti, hogy a logikai következetés nágyon oleső számos, a gyakorlatban előforduló ítéletkalkulus tudásbázis esetében.

Az IK-EL-VONZAT?( $TB, q$ ) előrefelé láncolási algoritmus meghatározza, hogy egy  $q$  ítéletkalkulus szimbólum – a lekérdezés – vonzata-e egy Horn-klózokat tartalmazó tudásbázisnak. Az algoritmus a tudásbázisban található ismert tényekből (pozitív literálok) indul ki. Ha egy implikáció minden premisszája ismert, akkor a konklúzióját hozzáadjuk az ismert tények halmazához. Például, ha ismert a  $P_{1,1}$ , és a Szellő és a  $(P_{1,1} \wedge \text{Szellő}) \Rightarrow S_{1,1}$  a tudásbázisban van, akkor  $S_{1,1}$ -t hozzáadhatjuk ehhez. Ez a folyamat folytatódik, amíg vagy a  $q$  lekérdezést hozzá tudjuk adni az ismert tények halmazához, vagy már nem tudunk további következetést végezni. A részletes algoritmust a 7.14. ábra mutatja, a legfontosabb tulajdonság, amire emlékeznünk kell, hogy ez lineáris időben fut.

Az algoritmust a legjobban úgy érhetjük meg, ha megnézünk egy példát és egy hozzá tartozó ábrát. A 7.15. (a) ábra egy egyszerű Horn-klózokból álló tudásbázist mutat be, amely az  $A$  és  $B$  ismert tényeket tartalmazza. A 7.15. (b) ábra ugyanez a tudásbázist egy ÉS-VAGY gráfkként (AND-OR graph) ábrázolja. Az ÉS-VAGY gráfokban a konjunkciókat ívekkel összefogott kapcsolatok – minden egyes kapcsolatot bizonyítani kell – jelölik, míg a diszjunkciókat ívek nélkül összefutó kapcsolatok – elég valamelyik kapcsolatot bizonyítani – jelölik. Könnyű áttekinteni, hogyan is működik az előrefelé láncolás a gráfon. Az ismert levelek (itt  $A$  és  $B$ ) adottak, és a következetés addig halad felfelé a gráfon, amíg lehetséges. Bárhol megjelenik egy konjunkció a folyamatban, a továbbterjesztés megáll addig, amíg az összes konjunkt ismertté nem válik, majd ezután halad tovább. Javasoljuk az olvasónak, hogy vegye át részletesen a példát.

Könnyű belátni, hogy az előrefelé láncolás helyes: minden következetés valójában a Modus Ponens egy alkalmazása. Az előrefelé láncolás teljes is: minden vonzat atomi mondatot vezet le. A legegyszerűbben úgy érhetjük meg ezt, ha a kikövetkeztetett állá-

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ A \\ B \end{aligned}$$



(a)

(b)

7.15. ábra. (a) Egyszerű Horn-klózokból álló tudásbázis. (b) Ugyanez a tudásbázis ÉS-VAGY gráfkként.

pot tábla végső állapotát tekintjük (miután az algoritmus elérte egy **fix pontot (fixed point)**), ahol már újabb következtetés nem lehetséges). A tábla *igaz* értéket tartalmaz minden, a folyamat során kikövetkeztetett szimbólumra, és *hamis* értéket a többi szimbólumra. A táblát úgy tekinthetjük, mint egy logikai modellt, ráadásul *minden határozott klóz az eredeti TB-ben igaz ebben a modellben*. Ennek belátására feltételezzük az ellenkezőjét, nevezetesen, hogy valamely  $a_1 \wedge \dots \wedge a_k \Rightarrow b$  klóz hamis a modellben. Ez azt jelenti, hogy  $a_1 \wedge \dots \wedge a_k$ -nak igaznak, míg  $b$ -nek hamisnak kell lennie a modellben. Ez viszont ellentmond annak a feltételezésünknek, hogy az algoritmus elérte a fix pontot! Így levonhatjuk azt a következtetést, hogy a kikövetkeztetett atomi mondatok halmaza a fix pontban az eredeti *TB* egy modelljét definiálja. Ezenkívül, bármely  $q$  atomi mondatnak, amely vonzata a *TB*-nek, igaznak kell lennie ennek minden modelljében, így ebben az említett modellben is. Ennél fogva az algoritmusnak ki kell következtetnie minden  $q$  vonzat mondatot.

Az előrefelé láncolás egy példája az általános **adatvezéreltnek (data-driven)** nevezett következtetési elvnek, amely olyan következtetéseket jelent, ahol a figyelem fókusza kezdetben az ismert adatokon van. Ez a megközelítés alkalmazható az ágenseknél a bejövő érzetekből történő következtetések vezetésére, gyakran anélküli is, hogy valamilyen kérdés lebegne a szemünk előtt. Például a wumpus ágens *KUELEMENT*-heti az érzeteit a tudásbázisnak, felhasználva egy inkrementális előrefelé következtető algoritmust, amelyben új tényeket lehet felvenni egy előjegyzési listára, hogy további következtetés inicializálódjon. A minden nap életben, jelentős számban fordul elő adatvezérelt következtetés, amikor új információk érkeznek. Például ha bent vagyok a szobában és hallom, hogy elkezd esni, akkor felmerülhet ben nem, hogy a tervezett kirándulás elmarad. Ugyanakkor az valószínűleg nem fog megfordulni a fejemen, hogy a szomszéd kertjében a legnagyobb rózsa tizenhetedik szirma nedves lesz. Az emberek óvatos kontroll alatt tartják az előrefelé láncolást, nehogy elárasszák őket az irreleváns következmények.

A visszafelé láncolás algoritmus, ahogy azt a neve is sugallja, visszafelé működik a lekérdezésből indulva. Ha a  $q$  lekérdezésről tudjuk, hogy igaz, akkor nem szükséges további munka. Egyébként az algoritmus megtalál minden olyan implikációt a tudásbázisban, amelynek a következménye  $q$ . Ha valamelyik ilyen implikáció az összes premisszáját be lehet bizonyítani (visszafelé láncolással), akkor  $q$  igaz. Amikor a 7.15. ábrán a  $Q$  lekérdezésre alkalmazzuk a módszert, akkor ez az ábrán követve vissza, lefelé működik, addig, amíg el nem éri az ismert tények halmazát, ami a bizonyítás alapja. A részletes algoritmust meghagyjuk feladatnak, és mint az előrefelé láncolásnál, egy hatékony implementáció itt is lineáris időben működik.

A visszafelé láncolás a **célorientált következtetés (goal-oriented reasoning)** egy formája. Hasznos-e megválaszolni olyan kérdéseket, mint hogy „Mit tegyek most?”, vagy hogy „Hol vannak a kulcsaim?”. Gyakran a visszafelé láncolás költsége *sokkal kisebb*, mint a tudásbázis méretétől lineárisan függő költség, mert a folyamat csak releváns tényeket érint. Általánosságban, egy ágensnek meg kell osztania a munkát az előrefelé és hátrafelé láncolás között, korlátozva az előrefelé következtetés alkalmazását azokra a tényekre, amelyek valószínűleg relevánsak azon lekérdezésekhez, amelyeket az ágens visszafelé láncolással fog megoldani.



## 7.6. HATÉKONY ÍTÉLETKALKULUS KÖVETKEZTETÉS

Ebben a szakaszban az ítéletkalkulus következtetés két hatékony családját írjuk le, amelyek a modellellenőrzésen alapulnak: az egyik megközelítés a visszalépéses keresésen alapul, a másik a hegmászó keresésen. Ezek az algoritmusok az ítéletkalkulus „technológiájának” részei. Ez a szakasz átlapozható a fejezet első olvasásakor.

Az algoritmusok, amelyeket leírunk, a kielégíthetőséget ellenőrzik. Már megfigyeltük a kapcsolatot egy logikai mondatot kielégítő modell megtalálása és egy kényszerkielégítési probléma megoldásának megtalálása között, így talán nem meglepő, hogy a két algoritmus család igen hasonlít az 5.2. alfejezetben bemutatott visszalépéses algoritmusokra és az 5.3. alfejezet lokális keresési algoritmusaira. Ezek az algoritmusok azonban rendkívül fontosak saját maguk jogán is, mert számos kombinatorikai probléma a számítástechnikában visszavezethető egy ítéletkalkulus mondat kielégíthetőségének ellenőrzésére. A kielégíthetőségi algoritmusok terén elérte bármilyen haladás általánosságban is óriási hatással van a képességeinkre a komplexitás kezelésében.

### Egy teljes visszalépéses algoritmus

Az első algoritmus, amit megnézünk Martin Davis és Hilary Putnam (1960) jelentős konferenciájukkal alapján, az úgynevezett **Davis–Putnam-algoritmus**. Ez az algoritmus valójában egy változata annak, amelyet Davis, Logemann és Loveland (1962) publikált, így DPLL-nek fogjuk hívni szerzőik neveinek kezdőbetűi alapján. A DPLL bemenetként egy konjunktív normál formájú mondatot vesz – a klózok egy halmazát. Mint a VISSZALÉPÉSES-KERESÉS vagy az IK-VONZAT?, ez is alapvetően rekurzív, mélyiségi felsorolását végzi a lehetséges modelleknek. Az algoritmus három továbbfejlesztést tartalmaz az IK-VONZAT? egyszerű sémájához képest:

- **Korai leállás:** Az algoritmus észreveszi, ha egy mondat már biztos igaz vagy hamis, még részben elkészült modell alapján is. Egy klóz igaz, ha bármelyik literál igaz, még akkor is, ha a többi literálnak még nincs igazság értéke. Ennél fogva a mondatot mint egészet igaznak ítélik, még mielőtt a modell teljes volna. Például az  $(A \vee B) \wedge (A \vee C)$  igaz, ha  $A$  igaz, függetlenül  $B$  és  $C$  értékétől. Hasonlóan, egy mondat hamis, ha bármelyik klóz hamis, ami akkor fordul elő, ha ennek a klóznak minden literálja hamis. Ismételten, ez a modell teljessé válásánál sokkal korábban is előfordulhat. A korai leállás a keresési tér egész részfáinak átvizsgálását kerüli el.
- **Tiszta szimbólum heurisztika:** Egy **tiszta szimbólum** (pure symbol) egy olyan szimbólum, amely mindenkor ugyanolyan „előjellel” szerepel minden klózban. Például a következő három mondatban,  $(A \vee \neg B)$ ,  $(\neg B \vee \neg C)$  és az  $A$  szimbólum tiszta, mivel csak a pozitív literálja jelenik meg, a  $B$  tiszta, mivel csak a negatív literálja jelenik meg, és  $C$  nem tiszta. Könnyű belátni, hogyha egy mondatnak van modellje, akkor létezik tiszta szimbólumokat tartalmazó modellje is, amelyben a tiszta szimbólumok értéke úgy van megválasztva, hogy literáljai igazak legyenek, hiszen így egyetlen klózt sem teszünk hamissá. Vegyük észre, hogy egy szimbólum tiszta-ság tulajdonságának meghatározásakor az algoritmus figyelmen kívül hagyhatja azokat a klózokat, amelyekről már tudjuk, hogy igazak a modell eddigi konstruálása alapján.

- Például ha a modellünk tartalmazza a  $B = \text{hamis}$  hozzárendelést, akkor a  $(\neg B \vee \neg C)$  klóz már igaz, és  $C$  tisztává válik, mert csak a  $(C \vee A)$ -ban jelenik meg.
- Egyésgklóz heurisztika:* Az *egységlőzöt* már definiáltuk korábban, mint egy olyan klóz, amelynek egy literálja van. A DPLL esetében, ez azt jelenti, hogy egy kivételelvet minden literál *hamis* értéket kapott a modellben. Például ha a modell tartalmazza a  $B = \text{hamis}$  hozzárendelést, akkor a  $(B \vee \neg C)$  egység klózzá válik, mivel ekvivalens a  $(\text{Hamis} \vee \neg C)$ -vel, azaz a  $\neg C$ -vel. Természetesen ahhoz, hogy ez a klóz igaz legyen,  $C$ -nek *hamis* értéket kell adni. Az egységlőzöt heurisztika elvégzi ezeket a hozzárendeléseket, mielőtt elágazna a maradékra. Egy fontos konzekvenciája ennek a heurisztikának, hogy bármely, már a tudásbázisban található literára vonatkozó (cáfolt általi) bizonyítási kísérlet azonnal sikeres lesz (7.16. feladat). Vegyük észre azt is, hogy értéket adva egy egységlőzohoz újabb egységlőzöt hozhatunk létre. Például amikor  $C$ -nek *hamis* értéket adunk, akkor a  $(C \vee A)$  egységlőzzá válik, ami az igaz  $A$ -hoz való hozzárendelését eredményezi. Ezt az „egymást követő” kikényszerített hozzárendelések sorozatát **egységtérjesztésnek (unit propagation)** nevezik. Használ a folyamat a Horn-klózokkal történő előrefelé láncolásra, és valóban, abban az esetben ha a CNF formájú kifejezés csak Horn-klózokat tartalmaz, akkor a DPLL lényegében lemásolja az előrefelé láncolást (lásd 7.17. feladat).

A DPLL algoritmust mutatja a 7.16. ábra. Az ábrán a lényeges elemeket tartalmazó vázat adtuk meg, ami bemutatja magát a keresési folyamatot. Nem szerepel az adatstruktúra leírása, amelyet fenn kell tartani, hogy a keresési lépések hatékonyan lehessen el-

```
function DPLL-KIELÉGÍTHETŐSÉG?(s) returns igaz vagy hamis
  inputs: s, egy ítéletkalkulus mondat
```

```
  klózok ← az s mondat CNF formájú klózainak halmaza
  szimbólumok ← az s mondat szimbólumainak listája
  return DPLL(klózok, szimbólumok, [])
```

```
function DPLL(klózok, szimbólumok, modell) returns igaz vagy hamis
```

```
  if minden klóz a klózok halmazban igaz a modell modellben then return igaz
  if néhány klóz a klózok halmazban hamis a modell modellben then return hamis
  P, érték ← TISZTA-SZIMBÓLUM-KERESÉS(klózok, szimbólumok, modell)
  if P nem nulla then return DPLL(klózok, szimbólumok-P, KIEGÉSZÍT(P, érték, modell))
  P, érték ← EGYSÉG-KLÓZ-KERESÉS(klózok, modell)
  if P nem nulla then return DPLL(klózok, szimbólumok-P, KIEGÉSZÍT(P, érték, modell))
  P ← ELSŐ(szimbólumok); maradék ← MARADÉK(szimbólumok);
  return DPLL(klózok, maradék, KIEGÉSZÍT(P, igaz, modell)) or
         DPLL(klózok, maradék, KIEGÉSZÍT(P, hamis, modell))
```

**7.16. ábra.** A DPLL algoritmus ítéletkalkulus mondatok kielégíthetőségének ellenőrzése. A TISZTA-SZIMBÓLUM-KERESÉS és az EGYSÉG-KLÓZ-KERESÉS eljárásokat a szövegben elmagyaráztuk; minden kettő egy szimbólummal vagy nullával tér vissza, és a szimbólumhoz hozzárendelendő igazságértékkel. Mint az IT-VONZAT? eljárás, ez is részleges modellekben dolgozik.

végezni, sem azoknak a trükköknek a leírása, amelyeket a teljesítmény fokozása céljából az alapalgoritmushoz hozzá lehet adni: klóztanulás, változó választási heurisztika és a véletlenszerű újraindítások technikája. Ha ezeket is beépítjük az algoritmusba, akkor a DPPLL az egyik leggyorsabb kielégíthetőségi algoritmus, antikvitása ellenére is. A CHAFF implementáció millió változós hardververifikációs problémák megoldására szolgált.

## Lokális keresés algoritmus

Számos lokális keresés algoritmust láttunk már a könyvben, beleértve a HEGYMÁSZÓ algoritmust (155. oldal) és a SZIMULÁLT-LEHŰTÉS-t (158. oldal). Ezek az algoritmusok alkalmazhatók közvetlenül is kielégíthetőségi problémákra, feltéve, hogy megfelelő kiértékelő függvényt választunk. Mivel a cél egy olyan hozzárendelés megtalálása, amely kielégít minden klózt, megfelel számunkra egy olyan kiértékelő függvény választása, amely a kielégítetlen klózok számát számolja. Valóban, ez pontosan az a mérték, amit a MIN-KONFLIKTUS algoritmus használt kényszerkielégítési problémánál (197. oldal). Minden ilyen algoritmus a teljes hozzárendelések terében végez lépéseket, cserélgetve egy-egy lépésben egy-egy szimbólum igazságértekét. A tér rendszerint számos lokális minimumot tartalmaz, amelyekből a meneküléshez a véletlenszerű lépések különféle formái szükségesek. Az utóbbi években nagyon sokat foglalkoztak azzal a kérdéssel, hogy hogyan lehet egy jó egyensúlyt találni a mohóság és a véletlenszerűség között.

Az egyik legegyszerűbb és leghatékonyabb algoritmus, amely ezen munkák között felbukkant a WALKSAT (7.17. ábra). minden iterációban az algoritmus vesz egy kielégítetlen klózt és egy szimbólumot a klózból, amelynek értékét ellenkezőre cseréli. Az értéket cserélő szimbólum kiválasztása a következő két módszer közül – véletlenszerűen választva – az egyikkel történik: (1) a „min-konfliktus” lépés, amely minimalizálja a kielégítetlen klózokat az új állapotban, és a (2) „véletlen-bejárás”, amely véletlenszerűen választja a szimbólumot.

Működik-e egyáltalán a WALKSAT? Az egyértelmű, hogy ha visszaad egy modellt, akkor a bemeneti mondat valóban kielégíthető. Mi van akkor, ha *kudarc*tal tér vissza?

```

function WALKSAT(klózok, p, max_csere) returns egy kielégítő modell vagy kudarc
  inputs: klózok, ítéletkalkulus klózok halmaza
           p, a „véletlen lépés” valószínűségének értéke, általában 0,5 körüli
           max_csere, a megengedett cserék száma, mielőtt feladná az algoritmus

  modell  $\leftarrow$  igaz/hamis értékek véletlen hozzárendelései a klózok szimbólumaihoz
  for i = 1 to max_csere do
    if modell kielégítíti klózok then return modell
    klózok  $\leftarrow$  véletlenszerűen választott klóz a klózok elemei közül, amely hamis a modell-ben
    with probability p érték csere egy véletlenszerűen választott klóz-ban a modell-ben
    else bármelyik szimbólum a klóz-ban, amely maximalizálja a kielégített klózok számát
  return kudarc
```

**7.17. ábra.** A WALKSAT algoritmus, kielégíthetőségek ellenőrzése véletlenszerűen cserélgetetett változó értékekkel. Számos változata ismert az algoritmusnak.

Sajnos ebben az esetben nem tudjuk megmondani, hogy a mondat kielégíthetetlen vagy az algoritmus számára több időt kellene hagyni a keresésre. Megpróbálhatjuk beállítani a *max\_csere* értékét végtelenre. Erre az esetre könnyű megmutatni, hogy a WALKSAT végül vissza fog adni egy modellt (ha létezik ilyen), feltéve, hogy a  $p$  valószínűségre igaz a  $p > 0$ . Ez azért van így, mert minden létezik egy olyan cserélési sorozat, amely elvezet egy kielégítő hozzárendeléshez, és előbb-utóbb a véletlen bejárás lépések elő fogják állítani ezt a sorozatot. Persze ha a *max\_csere* végtelen, és a mondat kielégíthetetlen, akkor az algoritmus soha nem fog leállni!

Ezek az eredmények azt sugallják, hogy a lokális keresési algoritmusok, mint a WALKSAT, akkor a leginkább hasznosak, ha feltételezhetjük, hogy létezik megoldás – mint például a 3. és 5. fejezetben tárgyalt problémáknak általában van megoldásuk. Viszont a lokális keresés nem képes detektálni a *kielégíthetetlenséget*, amely szükséges a maga után vonzás kérdésének az eldöntéséhez. Például egy lokális keresést alkalmaszó ágens nem tudja *megbízhatóan* bizonyítani, hogy egy négyzet biztonságos a wumpus világban. Ehelyett azt tudja mondani, hogy „Egy órája gondolkodom a kérdésen, és nem tudtam olyan lehetséges világot találni, amiben a négyzet nem biztonságos.” Ha a lokális keresés algoritmus általában igazán gyors egy modell megtalálásában, ha létezik ilyen, akkor az ágenst tekinthetjük igazolhatónak, feltételezve, hogy a kudarc a kielégíthetetlenséget jelzi. Ez természetesen nem azonos egy bizonyítással, és az ágensnek kétszer is meg kell gondolnia, mielőtt erre bízza az életét.

## Nehéz kielégíthetőségi problémák

Most megnézzük, hogyan is teljesít a DPLL és a WALKSAT a gyakorlatban. Minket elsősorban a *nehéz* problémák érdekelnek, mert a *könnyű* problémák megoldhatóak bármely régi algoritmussal is. Az 5. fejezetben láttunk néhány meglepő felfedezést bizonyosfajta problémákkal kapcsolatban. Például az  $n$ -királynő probléma – amely igen kemény feladat volt a visszalépéses keresés számára – triviálisan egyszerűnek bizonyult az olyan lokális keresési módszerek számára, mint például a min-konfliktus. Ennek az az oka, hogy a megoldások nagyon sűrűn oszlanak el a hozzárendelések terében, és bármely kezdeti hozzárendeléshez garantáltan találhatunk egy megoldást a közelben. Tehát az  $n$ -királynő könnyű probléma, mert **alulhatározott** (*under-constrained*).

Amikor konjunktív normál formában levő kielégíthetőségi problémákat tekintünk, alulhatározott probléma az, amelyben viszonylag kevés a változókat korlátozó klóz szerepel. Például itt van egy véletlenszerűen létrehozott<sup>12</sup> 3-CNF mondat öt szimbólummal és öt klózzal:

$$\begin{aligned} & (\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \\ & \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C) \end{aligned}$$

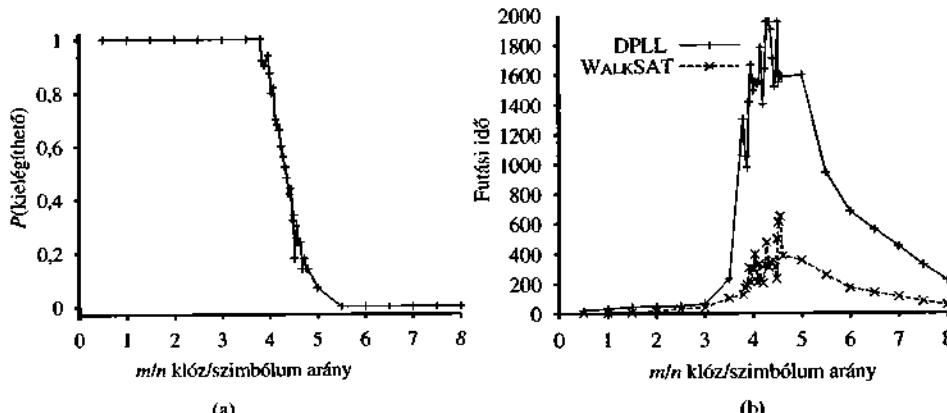
A 32 lehetséges hozzárendelésből 16 modellje a mondatnak, így átlagosan csak két véletlenszerű találhatásra van szükség egy modell megtalálásához.

<sup>12</sup> minden klóz három véletlenszerűen kiválasztott szimbólumot tartalmaz, amelyek 50% valószínűséggel negáltak.

Akkor melyek a nehéz problémák? Feltételezhető, hogyha *növeljük* a klózok számát, miközben a szimbólumok számát rögzítve tartjuk, a problémát erősebben határozotttá tesszük, és a megoldás megtalálása egyre nehezebbé válik. Legyen  $m$  a klózok száma és  $n$  a szimbólumok száma. A 7.18. (a) ábra mutatja annak a valószínűségét, hogy egy véletlenszerűen választott 3-CNF mondat kielégíthető-e a klóz/szimbólum arány ( $m/n$ ) függvényében rögzített  $n = 50$  mellett. Ahogy vártuk, kis  $m/n$  aránynál a valószínűség közel van 1-hez, és nagy  $m/n$  aránynál közel van a 0-hoz. A valószínűség viszonylag élesen esik le az  $m/n = 4,3$  értéknel. Azok a CNF mondatok, amelyek közel vannak ehhez a **kritikus ponthoz** (**critical point**), „alig kielégíthetőnek” vagy „alig kielégíthetlennek” jellemzhetők. Ez volna az, ahol a nehéz problémák vannak?

A 7.18. (b) ábra mutatja a DPLL és a WALKSAT futási időit ennek a pontnak a környezetében, csak a kielégíthető problémákat véve figyelembe. Hárrom dolog tiszta: először is, a kritikus pont körüli problémák *sokkal* nehezebbek, mint a többi probléma. Másodsor, a DPLL igen hatékony még a legnehezebb problémákra is – átlagosan néhány ezer lépést tesz, összehasonlítva az igazságtábla felsorolás  $2^{50} \approx 10^{15}$  számú lépésigényével. Harmadszor, a WALKSAT a teljes tartományban sokkal gyorsabb, mint a DPLL.

Természetesen ezek az eredmények csak a véletlenszerűen generált problémákra vonatkoznak. A valós problémáknak nem feltétlenül hasonló a struktúrájuk – a pozitív és negatív állítások aránya, a klózok közti kapcsolatok sűrűsége, és egyéb más vonatkozások – mint a véletlen problémáknak. Mégis, a gyakorlatban a WALKSAT és az ehhez kapcsolódó, hasonló algoritmusok nagyon jók valós problémák megoldásában is – gyakran ugyanolyan jók, mint az adott feladatra készített legjobb speciális célú algoritmus. Több ezer szimbólumot és milliónyi klózt tartalmazó problémákat megoldókkal szokás kezelni, mint amilyen a CHAFF. Ezek a megfigyelések azt sugallják, hogy a min-konfliktus és a véletlen bejárás algoritmusok valamelyen kombinációja általános célú képességet biztosít a legtöbb szituáció elemzéséhez, ahol kombinatorikai következtetés szükséges.



**7.18. ábra.** (a) A grafikon a klóz/szimbólum arány függvényében annak valószínűségét mutatja, hogy  $n = 50$  mondat közül véletlenszerűen választott 3-CNF mondat kielégíthető-e. (b) A grafikon a DPLL és a WALKSAT algoritmusok futási időinek középtériként mutatja 100 kielégíthető 3-CNF mondaton  $n = 50$  mellett, a kritikus  $m/n$  arány mellett szűk sávban.

## 7.7. ÍTÉLETLOGIKÁT ALKALMAZÓ ÁGENSEK

Ebben a szakaszban, összeszedünk minden, amit eddig tanultunk, hogy olyan ágenseket konstruálunk, amelyek az ítéletlogika felhasználásával működnek. Kétfajta ágenst fogunk megvizsgálni: olyat, amely következtetési algoritmust és egy tudásbázist használ, mint a 7.1. ábra általános tudásbázisú ágense, és olyat, amely a logikai kifejezések kiértékelését közvetlenül, áramkörökkel alkalmazva végzi. Mindkétfajta ágenst a wumpus világban fogjuk demonstrálni, és látni fogjuk, hogy minden komoly hátrányuktól szabad.

### Csapdák és wumpusok megtalálása logikai következtetés felhasználásával

Kezdjünk egy ágenssel, amely képes logikusan érvelni a csapdák és wumpusok helyével és a biztonságos négyzetekkel kapcsolatban. A feladat megoldása a tudásbázis felírásával kezdődik, amely reprezentálja a wumpus világ „fizikáját”. Az ágens tudja, hogy az  $[1, 1]$  nem tartalmaz se csapdát, se wumpust, azaz  $\neg C_{1,1}$  és  $\neg W_{1,1}$ . Az ágens minden  $[x, y]$  négyzetre ismer egy mondatot, ami azt jelenti ki, hogy a szellő honnan ered:

$$S_{x,y} \Leftrightarrow (C_{x,y+1} \vee C_{x,y-1} \vee C_{x+1,y} \vee C_{x-1,y}) \quad (7.1)$$

Az ágens minden  $[x, y]$  négyzetre ismer egy mondatot, ami azt jelenti ki, hogy a bűz hogyan ered:

$$B_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}) \quad (7.2)$$

Végül tudja, hogy pontosan egy wumpus van. Ezt két részben fejezzük ki. Először ki kell jelenteniünk, hogy *legalább* egy wumpus van:

$$W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,3} \vee W_{4,4}$$

Ezután azt kell mondanunk, hogy *legfeljebb* egy wumpus van. Az egyik lehetséges módja ennek, hogy minden két mezőre kijelentjük, hogy a két mező egyikének wumpusmentesnek kell lennie. Ha  $n$  négyzet van, akkor  $n(n - 1)/2$  mondatot kapunk, olyanokat mint a  $\neg W_{1,1} \vee \neg W_{1,2}$ . Egy  $4 \times 4$ -es világ esetében, így összesen 64 különböző szimbólumot tartalmazó 155 mondattal kezdünk.

Az ágensprogram, amelyet a 7.19. ábra mutat, KIJELENTI a tudásbázisnak az összes szellő és bűz érzetet. (Ezenkívül frissít néhány hagyományos változójának értékét is, amelyek azt őrizik, hogy az ágens merre van most és merre járt korábban – erre később még visszatérünk). Ezután a program kiválasztja, hogy melyik cellát nézze meg a peremen lévők közül – olyan négyzetek közül, amelyek szomszédosak valamelyik megláthatottal. Egy peremen lévő  $[x, y]$  cella *bizonyíthatóan biztonságos*, ha a  $(\neg C_{i,j} \wedge \neg W_{i,j})$  mondat vonzata a tudásbázisnak. A „második legjobb dolog”, ha egy cella *lehetséges, hogy biztonságos*, amikor az ágens nem tudja bizonyítani, hogy ott egy csapda vagy egy wumpus van – azaz, hogy a  $(C_{i,j} \vee W_{i,j})$  nem vonzat.

A KÉRDEZ eljárásban levő maga után vonzás számítása implementálható a fejezetben korábban bemutatott módszerek közül bármelyikkal. Az IT-VONZAT? (7.10. ábra) nyilván nem praktikus, mivel fel kellene sorolni a  $2^{64}$  sort. A DPLL (7.16. ábra) néhány milliszekundum alatt végrehajtja a kívánt következtést, leginkább az egység-

```

function IK-WUMPUS-ÁGENS(érzet) returns egy cselekvés
  inputs: érzet, egy lista, [bűz, szellő, ragyogás]
  static: TB, egy tudásbázis, kezdetben a wumpus világ „fizikáját” tartalmazza
    x, y, orientáció, az ágens pozíciója (kezdetben 1, 1) és orientációja (kezdetben jobbra)
    meglátogatott, egy tömb, meglátogatott négyzetek megmutatása, kezdetben hamis
    cselekvés, az ágens által végrehajtott utolsó cselekvés, kezdetben nulla
    terv, egy cselekvéssorozat, kezdetben üres

  frissít x, y, orientáció, meglátogatott a végrehajtott cselekvés alapján
  if bűz then KUELENT(TB,  $B_{x,y}$ ) else KUELENT(TB,  $\neg B_{x,y}$ )
  if szellő then KUELENT(TB,  $S_{x,y}$ ) else KUELENT(TB,  $\neg S_{x,y}$ )
  if ragyogás then cselekvés  $\leftarrow$  megragad
  else if terv nem üres then cselekvés  $\leftarrow$  ELSÓ(terv)
  else if valamely szélső [i, j] négyzetre, KÉRDEZ(TB, ( $\neg C_{i,j} \wedge \neg W_{i,j}$ ) igaz or
    valamely szélső [i, j] négyzetre, KÉRDEZ(TB, ( $C_{i,j} \vee W_{i,j}$ ) hamis then do
      cselekvés  $\leftarrow$  A*-GRÁF-KERESÉS(ÚTVONAL-PROBLÉMA([x, y], orientáció, ([i, j], meglátogatott)))
      cselekvés  $\leftarrow$  ELSÓ(terv)
  else cselekvés  $\leftarrow$  véletlen választott lépés
  return cselekvés

```

**7.19. ábra.** Egy wumpusvilág-beli ágens programja, amely ítéletkalkulust használ a csapdák, a wumpusok és a biztonságos négyzetek azonosítására. Az ÚTVONAL-PROBLÉMA szubrutin létrehoz egy keresési problémát, aminek megoldása egy cselekvéssorozat, amely elvezet az [x, y] négyzetből az [i, j]-be, és csak korábban már látogatott négyzeteken vezet keresztül.

propagáció heurisztikának köszönhetően. A WALKSAT szintén használható, a teljesség hiányából származó gyengeségeivel. A wumpus világokban, a modell megtalálásának kudarca, 10 000 cserét feltételezve, praktikusan megfelel a kielégíthetetlenségnek, így nem valószínű, hogy hiba fordulhatna elő a teljesség hiánya miatt.

Az IK-WUMPUS-ÁGENS igen jól működik kisméretű wumpus világban. Azonban mégis van valami erősen nem megnyugtató az ágens tudásbázisával kapcsolatban. A TB tartalmazza a fizikai leírást a (7.1) és (7.2) alakú egyenletek formájában minden egyes négyzetre. Minél nagyobb a környezet, annál nagyobb kezdeti tudásbázisra van szükség. Sokkal inkább azt szeretnénk, hogy csak két mondatunk lenne, amelyek kimondják, hogy hogyan is lehetkezhet a szellő vagy a bűz bármely kockában. Ez már túl van az ítéletkalkulus kifejezési képességén. A következő fejezetben látni fogunk egy nagyobb kifejezővel rendelkező logikai nyelvet, amelyben egyszerű lesz ilyen mondatot kijelenteni.

## A hely és az irány nyomkövetése

A 7.19. ábra ágensprogramja „csal”, mert a helyzetének nyomkövetését a tudásbázison kívül oldja meg, ahelyett hogy logikai következtést végezne.<sup>13</sup> Hogy „helyesen” oldjuk meg a nyomkövetést, szükségünk lesz az elhelyezkedéssel kapcsolatos állításokra.

<sup>13</sup> A figyelmes olvasó észreveszi, hogy ez megengedte számunkra azt, hogy ügyeskedjünk a nyers érzet, mint a Szellő, és a helyspecifikus állítás, mint az  $S_{1,1}$ , közötti kapcsolattal.

Első hallásra hajlamosak lehetünk felvenni egy  $H_{1,1}$  szimbólumot, ami azt reprezentál-ná, hogy az ágens az [1, 1]-ben van. Így a kezdeti tudásbázis ilyen mondatokat tartalmazhatna:

$$H_{1,1} \wedge ArccalJobbra \wedge Halad \Rightarrow H_{2,1}$$

Azonnal láthatjuk, hogy ez nem fog működni. Ha az ágens kiindul az [1, 1]-ből arccal jobbra és halad abban az irányban, akkor a tudásbázisból következni fog a  $H_{1,1}$  (az eredeti elhelyezkedés) és a  $H_{2,1}$  (az új elhelyezkedés). Mégsem lehet azonban minden két állítás igaz! A probléma az, hogy az elhelyezkedés állításoknak két különböző időpontra kell vonatkozniauk. Szükségünk van egy  $H_{1,1}^1$  szimbólumra, ami azt jelentené, hogy az ágens az [1, 1]-ben van az 1 időpontban, és egy  $H_{2,2}^2$  szimbólumra, ami azt, hogy az ágens a [2, 1]-ben van a 2 időpontban és így tovább. Az irány és az cselekvés állításoknak szintén függeniük kell az időtől. Így a helyes mondatok:

$$\begin{aligned} H_{1,1}^1 \wedge ArccalJobbra^1 \wedge Halad^1 &\Rightarrow H_{2,1}^2 \\ ArccalJobbra^1 \wedge FordulBalra^1 &\Rightarrow ArccalFelfelé^2 \end{aligned}$$

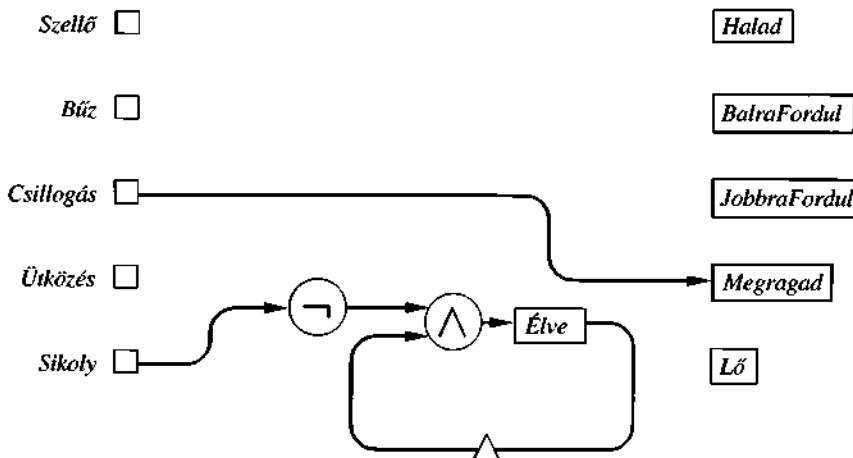
és így tovább. Az látható, hogy igen ügyesnek kell lennünk egy teljes és helyes tudásbázis felépítéséhez, hogy minden nyomon kövessünk a wumpus világban; ennek teljes tárgyalását elhalasztjuk a 10. fejezetig. A lényeg, amire itt rá akartunk mutatni, hogy a kezdeti tudásbázis tartalmazni fog olyan mondatokat, mint az előző két példa, minden  $t$  időpontra és minden helyre is. Ami azt jelenti, hogy minden  $t$  időpontra és  $[x,y]$  helyre a tudásbázis tartalmaz egy következő formájú mondatot:

$$H_{x,y}^t \wedge ArccalJobbra^t \wedge Haladt \Rightarrow H_{x+1,y}^{t+1} \quad (7.3)$$

Még akkor is, ha felső korlátot szabunk a megengedett lépések számának – 100-at talán – mondatok tízezreihez jutunk. Ugyanez a probléma merül fel, ha minden időlépéshez is hozzáadunk mondatokat, „ahogy szükséges”. A klózoknak ez az elburjánzása olvashatatlanná teszi számunkra a tudásbázist, de a gyors ítéletkalkulus megoldók még így is könnyen tudják kezelni a  $4 \times 4$ -es wumpus világot (korlátaikat a  $100 \times 100$ -as méret körül érik el). Az áramkörre alapuló ágens, amit a következő alfejezet mutat be, részleges megoldást kínál a klózok elburjánzásának problémájára, de a teljes megoldásra várnunk kell, míg nem alkalmazzuk az elsőrendű logikát majd a 8. fejezetben.

## Az áramkörön alapuló ágens

**Az áramkörön alapuló ágens (circuit-based agent)** a reflexív ágens egy állapottal rendelkező speciális fajtája, mint ahogy ezt a 2. fejezetben definiáltuk. Az érzetek egy sorrendi áramkör (**sequential circuit**) bemenetei, amely sorrendi áramkör egy **kapukból** (**gates**) álló hálózatból, amelyek mindenike egy logikai kapcsolatot valósít meg, és **regiszterekből** (**registers**) épül fel, amelyek mindenike az egyes állítások igazságértékeit tárolja. Az áramkör kimenetei olyan regiszterek, melyek cselekvéseknek felelnek meg – például a *Megragad* kimenet *igaz*-ra van állítva, ha az ágens meg akar ragadni valamit. Ha a *Csillagás* bemenetet közvetlenül összekötjük a *Megragad* kimenettel, akkor az ágens meg fogja ragadni a célt bármikor, ha látja (lásd 7.20. ábra).



**7.20. ábra.** Egy áramkóralapú, wumpusvilág-beli ágens egy része, a bemeneteket és a kimeneteket mutatva, az arany megragadását megvalósító áramkör, és az áramkör, amely meghatározza, hogy a wumpus él-e. A regisztereket négyzet jelöli, és a késleltetőket kis háromszögek mutatják.

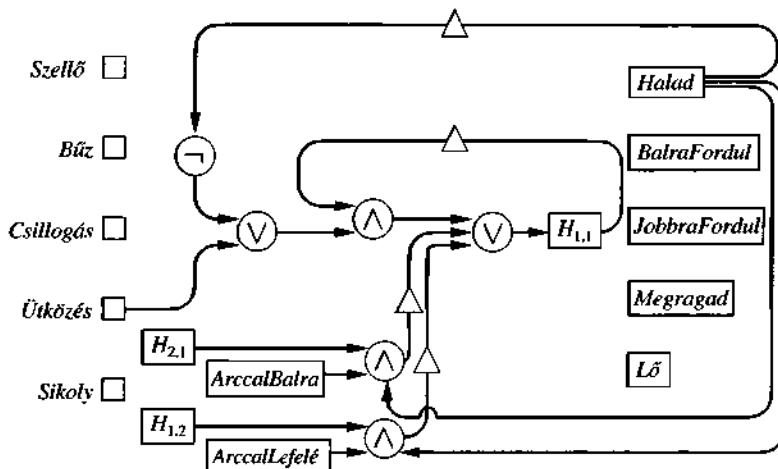
Az áramköröket az **adatfolyamok** (**dataflow**) mintájára értékeljük ki, minden időlépéssel beállítjuk a bemenetet, és a jeleket végigterjesztjük a hálózaton. Ha egy kapunak minden bemenete rendelkezésre áll, létrehoz egy kimenetet. Ez a folyamat közeli kapcsolatban van az előrefelé láncolással és az ÉS-VAGY gráfokkal, mint amilyen a 7.15. (b) ábrán látható.

Az előző szakaszban azt állítottuk, hogy az áramkörön alapuló ágens az időt kielégítőbb módon kezeli, mint az ítéletkalkulus következtetés alapú ágens. Ez azért lehet így, mert egy regiszterben tárolt változó értéke adja meg a megfelelő ítéletkalkulus szimbólumnak az igazságértékét a *pillanatnyi t időpontban*, ahelyett hogy másolatok léteznének minden egyes időlépéshoz. Például lehetne egy *Élve* regiszterünk, amelynek *igaz* értéket kell tartalmaznia, ha a wumpus életben van, és *hamis*-at, ha halott. Ez a regiszter az *Élve<sup>t</sup>* ítéletkalkulus szimbólumnak felel meg, így minden egyes időpillanatban egy különböző ítéletállításra vonatkozik. Az ágens belső állapotai – például a memóriája – megőrizhetők a regiszterek kimenetének **egy késleltető soron** (**delay line**) keresztül történő viszszacsatolásával. Ez a regiszter az előző lépéshoz tartozó értéket szolgáltatja. A 7.20. ábra egy példát mutat. Az *Élve* értékét a *Sikoly* negáltjának és az *Élve* késleltetett értékének a konjunkciója adja. Állításokban megfogalmazva ez a hálózat az *Élve* fogalomra megvalósítja a következő ekvivalenciát:

$$\text{Élve}^t \Leftrightarrow \neg\text{Sikoly}^t \wedge \text{Élve}^{t-1} \quad (7.4)$$

amely mondat azt mondja ki, hogy a wumpus a  $t$  időpontban akkor és csak akkor él, ha nem észleltünk sikolyt a  $t$  időpillanatban (egy  $t - 1$  időpontbeli sikoly alapján) és él a  $t - 1$  időpillanatban. Feltételezzük, hogy az áramkörben az *Élve* regiszter *igazra* van inicializálva. Így az *Élve* igaz marad mindaddig, amíg nincs egy sikoly, amikor is hamissá válik és a későbbiekben hamis marad. Ez pont az, amit mi akarunk.

Az ágens helyzete nagyon hasonló módon kezelhető, mint a wumpus halála. Szukségnünk van egy  $H_{x,y}$  regiszterre minden  $x$ -hez és  $y$ -hoz, és ennek az értéke *igaz* kell, hogy



7.21. ábra. Egy áramkör, amely meghatározza, hogy az ágens az [1, 1]-ben van-e. minden pozíció és orientáció regiszterhez egy hasonló áramkör csatlakozik.

legyen, amikor az ágens az  $[x, y]$ -ban van. Az áramkör, amelyik beállítja  $H_{x,y}$  értékét azonban sokkal bonyolultabb, mint az Élve áramköre. Például az ágens az [1, 1]-ben van a  $t$  időpontban, ha (a) az ott volt a  $t - 1$ -ben és vagy nem haladt semerre, vagy megpróbálta, de falnak ütközött; vagy (b) az [1, 2]-ben volt arccal lefelé és haladt előrefelé; vagy (c) a [2, 1]-ben volt arccal balra és haladt előre:

$$H_{1,1}^t \Leftrightarrow (H_{1,1}^{t-1} \wedge (\neg \text{Halad}^{t-1} \vee \text{FalnakÜtközik}^t)) \quad (7.5) \\ \vee (H_{1,2}^{t-1} \wedge (\text{ArccalLefelé}^{t-1} \wedge \text{Halad}^{t-1})) \\ \vee (H_{2,1}^{t-1} \wedge (\text{ArccalBalra}^{t-1} \wedge \text{Halad}^{t-1}))$$

A  $H_{1,1}$  hálózata a 7.21. ábrán látható. minden elhelyezkedés regiszterhez egy hasonló áramkör tartozik. A 7.13. (b) feladatban azt kérjük, hogy készítsen áramkört az irányra vonatkozó állításokhoz.

A 7.20. és 7.21. ábrákon levő áramkörök fenntartják az Élve és a  $H_{x,y}$  igazságértékét minden időpillanatban. Egy ilyen áramkör által reprezentált állítások szokatlanok azonban abban az értelemben, hogy a helyes igazságértékeket minden pillanatban meg tudjuk határozni. Tekintsük az előző helyett az  $S_{4,4}$  állítást: a [4, 4] négyzet szellős. Habár az állításnak az igazságértéke rögzítve van valamelyen értéken, az ágens nem ismerheti az igazságértéket mindaddig, amíg meg nem látogatja a [4, 4] mezőt (vagy ki nem következeti, hogy van egy szomszédos csapda). Az ítéletlogikát és az elsőrendű logikát igaz, hamis és ismeretlen értékű állítások automatikus reprezentálására terveztek, de az áramkörökkel nem: az  $S_{4,4}$  regiszternek tartalmaznia kell valamely értéket, vagy igaz vagy hamis értéket, még mielőtt az igazságot feltárták. A regiszterbeli érték lehet helytelen, és ez helytelen irányba is vezetheti az ágenst. Más szavakkal, szükségünk volna háromféle lehetséges állapot reprezentálására ( $S_{4,4}$  ismert igaz, ismerten hamis vagy nem ismert), és csak egy bitünk van arra, hogy megoldjuk ezt.

A megoldás erre a problémára az, hogy használunk két bitet egy helyett. Az  $S_{4,4}$ -et reprezentálja két regiszter, amelyeket  $K(S_{4,4})$ -nek és  $K(\neg S_{4,4})$ -nek nevezünk, ahol  $K$  je-

löli azt, hogy „ismert”. (Emlékezzünk, hogy ezek még mindig csak összetett nevekkel rendelkező szimbólumok, még akkor is, ha strukturált kifejezésnek tűnnek!) Ha minden  $K(S_{4,4})$  és  $K(\neg S_{4,4})$  hamis, akkor ez azt jelenti, hogy az  $S_{4,4}$  igazságértéke nem ismert. (Ha mindenkető igaz, akkor hiba van a tudásbázisban!) Ebben a reprezentációban bármikor is használnánk az  $S_{4,4}$ -et a hálózat valamely részében, a  $K(S_{4,4})$ -et használjuk helyette; és bármikor használnánk az  $\neg S_{4,4}$ -et, akkor a  $K(\neg S_{4,4})$ -et használjuk. Általánosságban, minden potenciális határozatlan állítást két **tudásállítással (knowledge propositions)** reprezentálunk, kijelentve, hogy a vonatkozó állítás ismerten igaz vagy ismerten hamis.

Hamarosan látni fogunk egy példát arra, hogyan kell alkalmaznunk a tudásállításokat. Először is, szükségünk van annak kidolgozására, hogy hogyan határozzuk meg maguknak a tudásállításoknak az igazságértékét. Végül észre, hogy miközben az  $S_{4,4}$ -nek rögzített igazságértéke volt, a  $K(S_{4,4})$  és  $K(\neg S_{4,4})$  változik, miközben az ágens egyre többet tud meg a vilagról. Például a  $K(S_{4,4})$  hamis értékről indul, és igazzá válik, amint az  $S_{4,4}$ -ről meghatározható, hogy igaz – azaz, amikor az ágens a [4, 4]-be lép, és detektálja a szellőt. Majd igaz marad ezután. Így megállapítható, hogy

$$K(S_{4,4})^t \Leftrightarrow K(S_{4,4})^{t-1} \vee (H_{4,4}^t \wedge \text{Szellő}^t) \quad (7.6)$$

Hasonló egyenlet írható fel  $K(\neg S_{4,4})^t$ -re is.

Most, hogy az ágensnek van már ismerete a szellős négyzetekről, foglalkozhat a csapdákkal is. Egy csapda hiánya az adott négyzetben akkor és csak akkor határozható meg, ha valamelyik szomszédos négyzetéről ismert, hogy nem szellős. Például ha létezik

$$K(\neg C_{4,4})^t \Leftrightarrow K(\neg S_{3,4})^t \vee K(\neg S_{4,3})^t \quad (7.7)$$

Egy csapda adott négyzetben való *jelenlétének* a meghatározása sokkal nehezebb – szellőnek kell lennie egy szomszédos négyzetben, de olyan szellőnek, amely nem egy másik csapda miatt keletkezett:

$$\begin{aligned} K(C_{4,4})^t &\Leftrightarrow (K(S_{3,4})^t \wedge K(\neg C_{2,4})^t \wedge K(\neg C_{3,3})^t) \\ &\vee (K(S_{4,3})^t \wedge K(\neg C_{4,2})^t \wedge K(\neg C_{3,3})^t) \end{aligned} \quad (7.8)$$

Miközben egy kicsit zavaros a csapdák jelenlétének és hiányának bizonyítása az áramkörök használata esetében, az áramkör *konstans számú kapui használ minden egyes négyzethez*. Ez a tulajdonság alapvetően fontos ahhoz, hogy olyan áramkörön alapuló ágenst építsünk, amelynek bonyolultsága kezelhető módon nő a probléma méretével. Ez valójában magának a wumpus világak a jellemzője; azt mondjuk, hogy egy környezet **lokalitás (locality)** tulajdonságot mutat fel, ha igaz, hogy bármelyik vizsgálandó állítás igazságának meghatározásához csak egy konstans számú másik állítás megvizsgálása szükséges. A lokalitás tulajdonság nagyon érzékeny a környezet pontos „fizikájára”. Például az aknakereső tárgyterület (7.11. feladat) nemlokális, mert az akna egy adott négyzetbeli létezésének eldöntése igényelheti, hogy akármilyen távol levő négyzetet is megnézzük. Nemlokális tárgyterületekre az áramkörre alapuló ágens alkalmazás nem minden praktikus.

Van egy másik probléma is, amely mellett eddig óvatosan elmentünk: a **hurokmentesség (acyclicity)** kérdése. Az áramkört hurokmentesnek tekintjük, ha minden olyan ág, amely egy regiszter kimenetét visszacsatolja a bemenetére, tartalmaz legalább egy közöbölő késleltető elemet. Megköveteljük, hogy minden áramkör hurokmentes legyen,

mivel a hurkokat tartalmazó áramkörök, mint fizikai eszközök, nem működnek! Az ilyen áramkör instabil oszcillációba kerülhet határozatlan értékeket eredményezve. Példaként a hurkot tartalmazó áramkörökre, tekintsük a (7.6) egyenletnek a következő kibővítését:

$$K(S_{4,4})^t \Leftrightarrow K(S_{4,4})^{t-1} \vee (H_{4,4}^t \wedge Szellő^t) \vee K(C_{3,4})^t \vee K(C_{4,3})^t \quad (7.9)$$

A hozzáadott diszjunktok, a  $K(C_{3,4})^t$  és a  $K(C_{4,3})^t$  lehetővé teszik az ágens számára, hogy a szellő jelenlétét meghatározza a szomszédos négyzetekben levő csapdák jelenlétéből, ami teljesen ésszerűnek tűnik. Így viszont sajnos a szellő tulajdonság függ a szomszédos csapdáktól, és a csapdák függenek a szomszédos négyzetek szellő tulajdonságától, ahogy azt a (7.8) egyenlet leírja. Ezáltal a teljes áramkör tartalmazna hurkot.

A nehézség nem az, hogy a (7.9) egyenlet *nem helyes*. A probléma inkább az, hogy az egyenletek által reprezentált függőségek összekapcsolása nem oldható fel az igazságértékek propagálásának egyszerű mechanizmusával a vizsgált logikai áramkörökben. A (7.6) egyenletet alkalmazó hurokmentes verzió, amely a szellősséget a közvetlen megfigyelésekkel határozza meg, *nem teljes* abban az értelemben, hogy bizonyos pontokon az áramkóralapú ágens kevesebbet tud, mint a teljes következtetési eljárást alkalmazó következtetésalapú ágens. Például ha van szellő az [1, 1]-ben, a következtetésalapú ágens képes arra a megállapításra jutni, hogy van egy szellő a [2, 2]-ben, míg a (7.6) egyenletet használó, hurokmentes áramkörre alapuló ágens erre nem képes. Lehetséges építeni egy teljes áramkört – végül is, szekvenciális áramkörök képesek emulálni bármilyen digitális számítógépet –, de ez jelentősen bonyolultabb volna.

## Összehasonlítás

A következtetésalapú és az áramkörön alapuló ágensek a deklaratív és procedurális megközelítés szélsőséges esetei az ágens tervezésében. Számos szempont szerint összehasonlíthatjuk őket:

- *Tömörség.* Az áramkörön alapuló ágensnek, ellentétben a következtetésalapú ágenssel, nincs szüksége a „tudásának” önálló másolataira minden időlépésben. Ehelyett, minden csak a pillanatnyi és a megelőző időlépésre hivatkozik. A „fizika” leírásához minden ágensnek szüksége van minden egyes négyzetre vonatkozó másolatokra (mondatok vagy áramkörök formájában), és így nem könnyű nagyobb környezeteket leírni. Azokban a környezetekben, ahol számos objektum összetett kapcsolatait kell leírni, a kijelentések száma eláraszt bármilyen ítéletkalkulus ágenst. Ezek a környezetek már igénylik az elsőrendű logika kifejezőjére (lásd 8. fejezet). Mindkét típusú ítéletkalkulus ágens rosszul alkalmazható egy biztonságos négyzettel szomszédos helyre vezető útvonal megtalálásának kifejezésére vagy megoldására. (Emiatt az IK-WUMPUS-ÁGENS keresési algoritmusokat alkalmaz.)
- *Számítási hatékonyság.* A legrosszabb esetben a következtetés időigénye exponenciálisan függ a szimbólumok számától, míg az áramkör futtatása az áramkör méretétől lineárisan függő időt igényel (vagy a mélységtől lineárisan függő időt, ha fizikai esz-

közként megvalósítjuk). Láthattuk azonban, hogy a *gyakorlatban* a DPLL igen gyorsan elvégezte a szükséges következtetéseket.<sup>14</sup>

- *Teljesség.* Azt sugalltuk korábban, hogy az áramkóralapú ágens lehet, hogy nem teljes a ciklusmentességgel kapcsolatos korlátozás miatt. A teljesség hiányának okai azonban még alapvetőbbek. Először is, emlékezzünk, hogy az áramkör a méretével lineárisan arányos időben oldja meg a feladatot. Ez azt jelenti, hogy bizonyos környezetek esetében egy teljes áramkörnek (egy olyannak, amely minden meghatározható állításhoz kiszámítja az igazságértéket) exponenciálisan nagyobbnak kell lennie, mint a következtetésalapú ágens tudásbázisának. Egyébként lenne módszerünk arra, hogy hogyan oldjuk meg az ítéletkalkulus vonzat problémáját kisebb mint exponenciális idő alatt, ami nagyon valószínűtlen. A második érv az ágens belső állapotainak a jellege. A következtetésalapú ágens emlékezik minden érzetre, és implicit vagy explicit módon ismer minden mondatot, ami az érzetekből és a kezdeti adatbázisból következik. Például ha adott  $S_{1,1}$  és ismeri a  $C_{1,2} \vee C_{2,1}$  állítást, abból  $S_{2,2}$  következik. Ezzel szemben az áramkóralapú ágens elfelejt minden korábbi érzetet és csak a regiszterekben tárolt egyedi állításokra emlékszik. Ekképpen az első érzet után a  $C_{1,2}$  vagy a  $C_{2,1}$  önmagában ismeretlen marad, és így semmilyen következtetés nem vonható le  $S_{2,2}$ -re vonatkozóan.
- *A létrehozás egyszerűsége.* Ez egy nagyon fontos kérdés, amelyet nehéz pontosan megítélni. A könyv szerzője számára a „fizika” leírása bizonyára sokkal egyszerűbb deklaratív úton, miközben kisméretű, hurokmentes, majdnem teljes áramkörök tervezése a csapdák megtalálására igen nehéznek bizonyult.

Összefoglalva, úgy tűnik, hogy létezik kompromisszum a számítási hatékonyság, a tömörség, a teljesség és a könnyű létrehozás között. Ha az érzetek és a cselekvések közötti kapcsolat egyszerű – mint a *Ragyogás* vagy a *Megrágad* közötti kapcsolat esetében – az áramkör optimálisnak tűnik. Ennél bonyolultabb kapcsolatok esetében a deklaratív megközelítés jobb lehet. Egy olyan területen, mint a sakk például, a deklaratív szabályok tömörek és könnyen kódolhatók (legalábbis az elsőrendű logikában), de a tábla állapotaiból a lépéseket közvetlenül számító hálózat elképzelhetetlenül hatalmas lenne.

Eltérő dolgokat látunk ezekkel a megközelítésekkel kapcsolatban az állatok világában. Az egyszerű idegrendszerrel rendelkező kisebb állatok valószínűleg hálózatai párak, míg a magasabb rendű állatok úgy tűnik, hogy explicit reprezentációkon következtetéseket végeznek. Ez lényegesen bonyolultabb ágensfunkciók számítását teszi lehetővé számukra. Az emberek szintén rendelkeznek áramkörökkel, amelyek a reflexeket valósítják meg, és talán képesek **lefordítani (compile)** deklaratív ismereteket áramkörökké, amikor bizonyos következtetések rutinná válnak. Ily módon egy **hibrid ágens (hybrid agent)** tervezése (lásd 2. fejezet) lehet a legjobb mindenkit világ számára.

<sup>14</sup> Valójában minden olyan következtetés, amely megvalósítható egy áramkörrel, megoldható lineáris időben a DPLL-lel. Ez azért van így, mert egy áramkör kiértékelése hasonlít az előrefelé láncoláshoz, amely emulálható DPLL-lel az egység propagáció szabály alkalmazásával.

## 7.8. ÖSSZEFOLGLALÁS

Bevezettük a tudásbázisú ágens ötletét, és megmutattuk, hogy hogyan tudunk olyan logikát definiálni, amellyel az ágens képes következtetni a világról. A legfontosabb pontok a következők:

- Az intelligens ágensnek szüksége van tudásra a világról, hogy jó döntéseket hozhasson.
- A tudást az ágens egy **tudásbázisban** (**knowledge base**) egy **tudásreprezentációs nyelv** (**knowledge representation language**) mondatainak (**sentences**) formájában tárolja.
- Egy tudásbázisú ágenst a tudásbázis és a következtetési mechanizmus alkotja. Működésének lényege, hogy a tudásbázisában a világot leíró mondatokat tárol és következtetési mechanizmust alkalmaz új mondatok következtetésére, majd felhasználja ezeket cselekvések meghatározására.
- Egy reprezentációs nyelvet **szintaxisa** (**syntax**) és **szemantikája** (**semantics**) definiál. A szintaxis a mondatok struktúráját határozza meg, a szemantika a mondatok igazságát (**truth**) határozza meg minden lehetséges világban (**world**) vagy modellben (**model**).
- A mondatok közötti **vonzat** (**entailment**) kapcsolatnak alapvetően fontos szerepe van a következtetés megértésében. Egy  $\alpha$  mondat maga után vonz egy másik  $\beta$  mondatot, ha  $\beta$  igaz minden világban, ahol  $\alpha$  igaz. Ezzel ekvivalens definíciók az  $\alpha \Rightarrow \beta$  mondat érvényességét (**validity**) és a  $\alpha \wedge \neg\beta$  mondat kielégíthetetlenségét (**unsatisfiability**) meghatározó definíciók.
- A következtetés az a folyamat, amivel új mondatok vezethetők le régiemből. A helyes (**sound**) következtetési algoritmusok csak vonzat mondatokat vezetnek le; a teljes (**complete**) algoritmus az összes következményt levezeti.
- Az ítéletkalkulus (**propositional logic**) egy nagyon egyszerű nyelv, amely ítéletszimbólumokból (**proposition symbols**) és logikai összekötőjelekből (**logical connectives**) áll. Képes kezelni olyan állításokat, amelyekről tudjuk, hogy igazak, tudjuk, hogy hamisak, vagy teljesen ismeretlen az igazságértekük.
- Ha adott egy ítéletkalkulus szimbólumszótár, akkor a lehetséges modellek száma véges, így a vonzatok ellenőrzése történhet a modellek felsorolásával is. Hatékony ítéletkalkulus **modellellenőrző** (**model checking**) algoritmusok többek között viszszalépéses vagy lokális keresési eljárásokat alkalmaznak, amelyekkel gyakran nagyméretű problémákat is nagyon gyorsan meg tudnak oldani.
- A következtetési szabályok (**inference rules**) helyes következtetési minták, amelyeket felhasználhatunk bizonyítások megtalálásához. A **rezolúció** (**resolution**) szabály teljes következtetési algoritmust biztosít olyan tudásbázisokhoz, amelyek konjunktív normál formában (**conjunctive normal form**) vannak kifejezve. Az előrefelé láncolás (**forward chaining**) és a hátrafelé láncolás (**backward chaining**) igen természetes érvényű algoritmusok Horn-formában (**Horn form**) adott tudásbázisokon.
- Kétfajta ágenst lehet építeni az ítéletkalkulus alkalmazására: a következtetésalapú ágens (**inference-based agent**) következtetési algoritmusokat használ a világ eseményeinek követésére, és képes rejttett jellemzőket is levezeti, míg az áramkörön alapuló ágens (**circuit-based agent**) az állításokat regiszterek bitjeiként reprezentálja, és jelek logikai áramkörökben történő terjesztésével végzi ezek frissítését.

- Az ítéletkalkulus meglehetősen hatékony bizonyos feladatokra egy ágensben alkalmazva, de kezelhetetlen nemkorlátos méretű környezetekben, mivel hiányzik a megfelelő kifejező erő az idő, a tér és az objektumok közötti kapcsolatok általános mintáinak leírására.

## Irodalmi és történeti megjegyzések

John McCarthy cikke a „Programs with Common Sense” (McCarthy, 1958, 1968) tette híressé az ágens fogalmát, egy olyan programét, amely az érzetek és cselekvések összekapcsolására logikai következetést használ. A cikk kitűzte a deklarativizmus zászlóját, megmutatva, hogy szoftverek írásának igen elegáns módszere az, ha a szükséges ismereteket közöljük az ágenssel. Allen Newell cikke (1982) a „The Knowledge Level” tárgyalja azt a megközelítést, hogy racionális ágensek leírhatók és elemzhetők egy absztrakt szinten, ami az általuk birtokolt tudást és nem azt a programot definiálja, amit futtatnak. A mesterséges intelligencia deklaratív és procedurális megközelítéseit hasonlíta össze Boden (Boden, 1977). A vitát, mások mellett Brooks (Brooks, 1991) és Nilsson (Nilsson, 1991) írásai élesztették fel újra.

Magának a logikának az eredete az ósi görög filozófiában és matematikában található. Számos logikai alapelv – a mondatok szintaktikus struktúrájának összekötése az igaz vagy hamis jellegükkel, a jelentéstükkel, a bennük megjelenő argumentumok érvényességével – elszórt helyeken megtalálható Platón műveiben. Arisztotelész készítette az első ismert, rendszerezett munkát a logikáról. Munkáját diákjai gyűjtötték össze halála után, i. e. 322-ben az *Organon* c. tanulmányban. Arisztotelész **szillogizmusai** (*syllogisms*) olyan logikai állítások voltak, amelyeket ma következetési szabályoknak neveznénk. Habár a szillogizmus tartalmazott elemeket, mind a propozíciós, mind az elsőrendű logikából, a rendszer, mint egész, igen gyengének számít a modern kívánlalmak szerint. Nem tette lehetővé tetszőleges komplexitású mondatok létrehozását a következetési mintákban, mint a modern ítéletlogika.

A hasonló Megara és sztoikus iskolák (az i. e. 5. században indultak, és több száz éven keresztül működtek) vezették be az implikációt és más alapvető szerkezeteket, amelyeket ma is használunk a modern ítéletlogikában. Logikai összekötőjelek definíálására vezette be az igazságítábla használatát Philón és Megara. A sztoikusok öt alapvető következetési szabályt használtak, amelyeket igazolás nélkül érvényesnek tartottak, közülük azt a szabályt, amelyet ma Modus Ponensnek nevezünk. Számos szabályt vezettek le ebből az ötből, felhasználva többek között a dedukció elméletének alapelveit (267. oldal), és sokkal tisztábban használták a bizonyítás fogalmát, mint azt Arisztotelész tette. A sztoikusok azt állították, hogy az öt logikájuk teljes abban az értelemben, hogy tartalmaz minden érvényes következetést, de ami fennmaradt munkájukból, az túlságosan töredékes ahhoz, hogy elemezhető legyen. A Megara és a sztoikus logikák történetének, már amennyire ezek ismertek, jó beszámolóját készítette el Benson Mates (Mates, 1953).

Wilhelm Leibniztől (1646–1716) származik a mesterséges formális nyelvi minták létrehozásának ötlete a logikai kapcsolatok tisztázásának és a logikai következetés egy tisztán formális és mechanikus folyamattá való egyszerűsítése céljából. Leibniz saját matematikai logikája azonban igencsak tökéletlen volt, és rá inkább ezeknek a gondo-

latoknak mint célkitűzéseknek a bevezetéséért és nem e célok megvalósítására tett kísérletei miatt emlékezünk.

George Boole vezette be első igazán átfogó és működőképes formális logikai alapú rendszert a *The Mathematical Analysis of Logic* c. könyvében (Boole, 1847). Boole logikájának modellje közel állt a valós számok algebrájához, és elsődleges következetési módszerként a logikailag ekvivalens mondatok helyettesítését használta. Bár Boole rendszere csak töredéke volt a teljes ítéletkalkulusnak, elég közel volt ahhoz, hogy a 19. század szerzői Boole-t követve hamar kitöltsék a hiányzó részeket. Schröder definiáltá (Schröder, 1877) a konjunktív normál formát, míg a Horn-formát sokkal később Alfred Horn (Horn, 1951) vezette be. A modern ítéletkalkulus (és elsőrendű logika) első átfogó bennutatása Gottlob Frege *Begriffschrift* (Az írás fogalma vagy Fogalmi jelölés) c. könyvében (Frege, 1879) található.

Az első mechanikai eszközt, amely logikai következetést végzett, Stanhope harmadik grófja (1753–1816) készítette. A Stanhope Demonstrator képes volt szillogizmusokat kezelní és bizonyos valószínűségi következetéseket végezni. William Stanley Jevons – egyike azoknak, akik továbbfejlesztették és kiegészítették Boole munkáját – 1869-ben a Boole-logikán alapuló következetések végrehajtására megépítette a „logical piano”-ját. Ezeknek és más korai következetésre készített mechanikus eszközöknek szórakoztató és tanulságos története Martin Gardner (Gardner, 1968) munkájában olvasható. Az első logikai következetést végző, publikált számítógépes program a „Logic Theorist” volt, amelyet Newell, Shaw és Simon (Newell, Shaw és Simon, 1957) készítettek. Ezt a programot az emberi gondolkodás modellezésére szánták a szerzők. Bár Martin Davis (Davis, 1957) tervezett egy programot, ami bizonyíthatóan 1954-ben készült, de a Logic Theorist eredményeit korábban publikálták. Mind Davis 1954-es programja, mind a Logic Theorist részben *ad hoc* módszerekre épült, amelyek nem voltak jelentős hatással a későbbi automatikus dedukciós rendszerekre.

Az igazságtabláknak, mint az ítéletkalkulus nyelvében az érvényességnak vagy a mondat kielégíthetetlenségének a tesztjét egymástól függetlenül Ludwig Wittgenstein (1922) és Emil Post (1921) vezették be. A harmincas években jelentős haladást értek el az elsőrendű logika következetési módszereinek terén. Nevezetesen, Gödel megmutatta (Gödel, 1930), hogy az elsőrendű logikában történő következetésre egy teljes eljárást lehet kapni az ítéletlogikára történő redukcióval, felhasználva Herbrand elméletét (Herbrand, 1930). A 9. fejezetben újra áttekintjük ennek történetét, itt most a lényeges pont az, hogy a hatvanas években, a hatékony ítéletkalkulus algoritmusok fejlesztésében a matematikusok érdeklődését jelentős mértékben motiválta az a cél, hogy hatékony tételebizonyítót készítsenek az elsőrendű logika számára. A Davis–Putnam-algoritmus (Davis és Putnam, 1960) volt az első hatásos algoritmus az ítéletkalkulus rezolúció megvalósítására, de a legtöbb esetben ez is sokkal kevésbé hatékony, mint a DPLL visszalépéses algoritmus, amelyet két évvel később mutattak be (1962). A teljes rezolúciós szabály és a rezolúció teljességének bizonyítása J. A. Robinson (Robinson, 1965) nagyhatású tanulmányában jelent meg, ami azt is megmutatta, hogyan lehet elsőrendű logikai következetést végezni ítéletkalkulus technikák igénybevétele nélkül.

Stephen Cook (Cook, 1971) mutatta meg, hogy a kielégíthetőség eldöntése az ítéletlogikában NP-teljes. Mivel a vonzat meghatározása ekvivalens a kielégíthetetlenség eldönthetével, ez is NP-teljes. Az ítéletlogika számos részhalmaza ismert, amelyről tudnuk, hogy bennük a kielégíthetőség problémája polinomiális időben megoldható. A Horn-klózok az

egyik ilyen rézhalmaz. A Horn-klózokon működő, lineáris időben futó előreláncolási algoritmust Dowlingnak és Galliernek (Dowling és Gallier, 1984) köszönhetjük, akik az algoritmusokat egy adatfolyam-eljárásként írták le, hasonlóan a jelek terjedéséhez az áramkörökben. A kielégíthetőség ellenőrzése alapvető módszerré vált a problémák NP-teljességének vizsgálatában; például Kaye (Kaye, 2000), megmutatta, hogy az Aknake-soros játék (lásd 7.11. feladat) NP-teljes.

Számos szerző próbálkozott lokális keresési algoritmusokat használni a kielégíthetőség elődöntésére a nyolcvanas években. minden ilyen algoritmus a kielégíthetetlen klózok számának minimalizálásának elvén (Hansen és Jaumard, 1990) alapult. Különlegesen hatékony algoritmust fejlesztett ki Gu (Gu, 1989), és tőle függetlenül Selman és társai (Selman és társai, 1992), amelyet ez utóbbi szerzők GSAT-nak neveztek el, és megmutatták, hogy az algoritmus képes nehéz problémák széles körét igen gyorsan megoldani. A WALKSAT algoritmust, amelyet ebben a fejezetben bemutattunk, szintén Selma és társai publikálták (1996).

A „fázisátmenet” létezését a véletlen  $k$ -SAT problémák kielégíthetőségében először Simon és Dubois (Simon és Dubois, 1989) figyelték meg. Crawford és Auton tapasztalati eredményei (Crawford és Auton, 1993) azt sejtették, hogy nagyméretű, véletlenszerű 3-SAT problémák esetén ez az átmenet a 4,24-es klóz/változó arány könyékén van. Cikkük szintén bemutat egy hatékony DPLL megvalósítást. Bayardo és Schrag írt egy másik, kényszer-kielégítési technikákat alkalmazó hatékony DPLL megvalósítást (Bayardo és Schrag, 1997). Moskewicz és társai publikálták a CHAFF algoritmust (Moskewicz és társai, 2001), amely millió változós hardververifikációs problémákat képes megoldani, és amely megnyerte a SAT 2002 versenyt. Li és Anbulagan gyors probléma-megoldókat lehetővé tevő egységpropagációt alapuló heurisztikákat tárgyal (Li és Anbulagan, 1997). Cheeseman és társai (1991) számos hasonló problémáról adtak adatokat, és megfogalmazták azt a feltevést, hogy minden NP-teljes problémának van állapotátmenete (Cheeseman és társai, 1991). Kirkpatrick és Selman módszereket mutattak arra (Kirkpatrick és Selman, 1994), hogy a statisztikus fizika technikáit hogyan lehet alkalmazni a fázisátmenetek pontos „alakjának” jobb megértéséhez. Az átmenetek helyének meghatározására vonatkozó elméleti vizsgálatok igen gyengék: az egyetlen, amit sikerült bizonyítani, hogy az átmenet a [3,003 4,598] tartományba esik a véletlen 3-SAT problémáknál. Cook és Mitchell kiváló áttekintést készítettek ezekről és számos más kielégíthetőséggel kapcsolatos téma eredményeiről (Cook és Mitchell, 1997).

Korai elméleti kutatások megmutatták, hogy a DPLL átlagos komplexitása polinomiális a problémák egy bizonyos természetes eloszlású körei. Ez a potenciálisan érdekes tény kevésbé izgalmasá válta, miután Franco és Paull megmutatták, hogy ugyanezek a problémák konstans időben megoldhatók egyszerűen találgtatással, ahol véletlen hozzárendelésekkel dolgozunk (Franco és Paull, 1983). A véletlen generáló módszer, amelyet a fejezetben bemutattunk, sokkal nehezebb problémákat hoz létre. Az ilyen, a problémákon lokális kereséssel elért tapasztalati sikerek által motiválva, Koutsoupias és Papadimitriou megmutatta, hogy egy egyszerű hegymászó algoritmussal nagyon gyorsan megoldható szinte minden kielégíthetőségi probléma példány (Koutsoupias és Papadimitriou, 1992), ami azt sugallja, hogy a nehéz problémák ritkák. Mi több, Schöning bejelentett (Schöning, 1999) egy véletlen elemeket tartalmazó GSAT változatot, amelynek a *legrosszabb esetre* számított várható futási ideje 3-SAT problémákon  $1,333^n$  – ami még mindig exponenciális, de lényegesen gyorsabb, mint a korábbi

legrosszabb esetkorlátok. A kielégíthetőségi algoritmusok ma is igen aktív területét képezik a kutatásoknak; Du és társai cikkgyűjteménye (Du és társai, 1999) jó kiindulási pontot jelent.

Az áramköralapú ágensek gondolata visszavezethető McCulloch és Pitts nagyhatású cikkére (McCulloch és Pitts, 1943), amely a neurális hálózatok témakör kezdetét jellelte. Népszerű vélekedésekkel szemben a cikk valójában logikai áramköralapú ágens-terveknek az agyban történő megvalósításával foglalkozott. Azonban az áramköralapú ágens kis figyelmet kapott a mesterséges intelligenciában. A leginkább megjegyzendő kivétel Stan Rosenschein munkája (Rosenschein, 1985; Kaelbling és Rosenschein, 1990), aki módszert dolgozott ki áramköralapú ágensek lefordítására a feladat környezetének deklaratív leírására. A regiszterekben tárolt állítások frissítésének módszere közel a kapcsolatban van a Reiter által az elsőrendű logikára kifejlesztett követő állapot axiómához (*successor-state axiom*) (Reiter, 1991). Rod Brooks munkája (1986, 1989) demonstrálta az áramköralapú megvalósítások hatékonyságát robotok vezérlésére – a 25. fejezetben foglalkozunk ezzel a témaival. Brooks amellett érvelt (Brooks, 1991), hogy az áramköralapú ágensen kívül más nem is szükséges a mesterséges intelligenciához – mivel más módszerek nehézkesebbek, drágák és szükségtelenek. A mi nézetünk szerint, egyik módszer sem elégsges önmagában.

A wumpus világot Gregory Yob (Yob, 1975) találta ki. Némi iróniával, Yob azért fejlesztette ki, mert unta azokat a játékokat, amelyeket egy rácson játszanak: az eredeti wumpus világ dodekaéder alakú volt, mi helyeztük vissza a régi unalmas négyzetrácsba. Michael Genesereth javasolta először, hogy a wumpus világot az ágensek tesztkörnyezeteként alkalmazzuk.

## Feladatok

- 7.1. Írja le a wumpus világot a 2. fejezetben felsorolt feladatkörnyezetek tulajdonságainak felhasználásával.
- 7.2. Tegyük fel, hogy az ágens eljutott a 7.4. (a) ábra szerinti állapotba, úgy, hogy nem érzett semmit az [1, 1]-ben, szellőt érzett az [2, 1]-ben és búzt az [1, 2]-ben, és most vizsgálja az [1, 3], [2, 2] és [3, 1] négyzeteik tartalmát. Ezek bármelyike tartalmazhat csapdát, és legfeljebb az egyik tartalmazhatja a wumpust. A 7.5. ábra példáját követve hozza létre a lehetséges világokat (32 ilyet kell találnia). Jelölje meg azokat a világokat, amelyekben a *TB* igaz, és azokat, amelyekben a következő mondatok igazak:

$$\alpha_2 = \text{„Nincsen csapda a [2, 2]-ben”}$$

$$\alpha_3 = \text{„Az [1,3]-ban wumpus van”}$$

Mutassa meg ezenkívül, hogy  $TB \models \alpha_2$  és  $TB \models \alpha_3$ .

- 7.3. Tekintsük azt a problémát, hogy hogyan dönthető el egy ítéletkalkulus mondat igazsága egy adott modellben.
  - (a) Írjon egy *IK-IGAZ?*(*s, m*) rekurzív programot, amely akkor és csak akkor ad vissza igazat, ha az *s* mondat igaz az *m* modellben (ahol *m* minden

s szimbólumhoz egy igazságértéket rendel). Az algoritmusnak a mondat méretével lineárisan változó időben kell futnia. (Használhatja ennek a függvénynek egy változatát az online kódírából.)

- (b) Adjon három példát olyan mondatokra, amelyekről meghatározható, hogy igazak vagy hamisak egy részleges modellben, amely nem specifikálja minden szimbólum igazságértékét.
- (c) Mutassa meg, hogy egy részleges modellben egy mondat igazságértéke (ha van ilyen) általánosságban nem határozható meg hatékonyan.
- (d) Módosítsa az IK-IGAZ? algoritmust úgy, hogy néha részleges modell alapján is meg tudjon határozni igazságértékeket és közben tartsa meg a függvény igazságértékét és lineáris futási idejét. Adjon meg három példát olyan mondatokra, amelyek igazságértékét a részleges modellben nem állapítja meg az algoritmus.
- (e) Vizsgálja meg, hogy a módosított algoritmus hatékonyabbá teszi-e az IT-VONZAT? eljárást.

#### 7.4. Bizonyítsa be a következő állításokat:

- (a)  $\alpha$  akkor és csak akkor érvényes, ha  $Igaz \models \alpha$ .
- (b) Bármilyen  $\alpha$ -ra  $Hamis \models \alpha$ .
- (c)  $\alpha \models \beta$  akkor és csak akkor, ha az  $(\alpha \Rightarrow \beta)$  mondat érvényes.
- (d)  $\alpha \equiv \beta$  akkor és csak akkor, ha az  $(\alpha \Leftrightarrow \beta)$  mondat érvényes.
- (e)  $\alpha \models \beta$  akkor és csak akkor, ha az  $(\alpha \wedge \neg\beta)$  mondat kielégíthetetlen.

#### 7.5. Vegyünk egy szótárat, amelyben csak négy ítéletkalkulus állítás létezik, az $A$ , $B$ , $C$ és $D$ . Hány modellje létezik a következő mondatoknak?

- (a)  $(A \wedge B) \vee (B \wedge C)$
- (b)  $A \vee B$
- (c)  $A \Leftrightarrow B \Leftrightarrow C$

#### 7.6. Definiáltunk 4 bináris logikai összekötőjelet.

- (a) Léteznek-e még továbbiak, amelyek hasznosak lehetnek?
- (b) Hány bináris összekötőjel lehetséges?
- (c) Miért nem túlságosan hasznos közülük néhány?

#### 7.7. Egy szabadon választott módszer felhasználásával igazolja a 7.11. ábra ekvivalenciáit.

#### 7.8. Vizsgálja meg a következő mondatokat, és döntse el mindegyikre, hogy érvényesek, kielégíthetetlenek, vagy egyik sem. Igazolja a döntését igazságtáblával, vagy felhasználva a 7.11. ábra ekvivalencia szabályait. Van-e olyan, amit elsőre eltévesztett?

- (a)  $Füst \Leftrightarrow Füst$
- (b)  $Füst \Rightarrow Tűz$
- (c)  $(Füst \Rightarrow Tűz) \Rightarrow (\neg Füst \Rightarrow \neg Tűz)$
- (d)  $Füst \vee Tűz \vee \neg Tűz$
- (e)  $((Füst \wedge Hőség) \Rightarrow Tűz) \Leftrightarrow ((Füst \Rightarrow Tűz) \vee (Hőség \Rightarrow Tűz))$

- (f)  $(Füst \Rightarrow Tűz) \Rightarrow ((Füst \wedge Hőség) \Rightarrow Tűz)$   
 (g)  $Nagy \vee Hallgatag \vee (Nagy \Rightarrow Hallgatag)$   
 (h)  $(Nagy \wedge Hallgatag) \vee \neg Hallgatag$

- 7.9. (Barwise és Etchemendy, 1993 alapján) Be tudja-e bizonyítani, hogy az unikornis mitikus, ha adottak az alábbiak? Mit mondhatunk a mágikusságról? Van-e szarva?

Ha az unikornis mitikus, akkor halhatatlan, de ha nem halhatatlan, akkor egy halandó emlős. Ha az unikornis vagy halhatatlan, vagy emlős, akkor van szarva. Az unikornis mágikus állat, ha van szarva.

- 7.10. minden logikai mondat logikailag ekvivalens azzal az állítással, hogy minden lehetséges világ, amelyben hamis volna, nem az eset. Felhasználva ezt a megfigyelést, bizonyítsa be, hogy minden mondat átírható konjugált normált formára.

- 7.11. Az ismert számítógépes játék, az Aknakereső igen hasonló a wumpus világhoz. Az aknakereső világ egy  $N$  négyzetet tartalmazó négyzetes rács, amelyek közül  $M$  elszórta elhelyezkedő négyzet láthatatlan aknát takar. minden négyzetet megpróbálhat az ágens, de azonnali halállal lakol, ha ott egy aknát talál. Az Aknakereső játék minden kipróbált négyzetben felfedi, hogy hány közvetlenül vagy átlósan szomszédos négyzetben van akna, ezzel mutatva az aknák jelenlétéét. A cél az, hogy minden aknát nem tartalmazó négyzetet megpróbálunk.

- (a) Legyen  $X_{i,j}$  igaz akkor és csak akkor, ha az  $[i, j]$  tartalmaz egy aknát. Írja le az  $X_{i,j}$  szimbólum logikai kombinációját tartalmazó mondattal azt a kijelentést, hogy pontosan két akna szomszédos az  $[1, 1]$ -gel.  
 (b) Általánosítsa az (a)-beli kijelentést megmutatva, hogy hogyan hozható létre konjugált normál formájú mondat annak kifejezésére, hogy az  $n$  szomszédos négyzetből  $k$  négyzet tartalmaz aknát.  
 (c) Magyarázza meg pontosan, hogy hogyan használhat egy ágens DPLL algoritmust annak bizonyítására, hogy egy adott négyzet tartalmaz (vagy nem tartalmaz) aknát, figyelem kivíül hagyva azt a globális korlátot, hogy pontosan  $M$  akna van összesen.  
 (d) Tegyük fel, hogy a globális korlát a (b) feladatrészben megfogalmazott módon van létrehozva. Hogyan függ a klózok száma  $M$ -től és  $N$ -től? Javasoljon egy módosítást a DPLL algoritmushoz, amellyel a globális korlátot nem kell explicit módon reprezentálni.  
 (e) Van-e olyan a (c) feladatrészben definiált eljárás által előállított következmény, amely érvénytelenné válna, ha figyelembe vennénk a globális kényezséget?  
 (f) Mutasson példát olyan lehetséges konfigurációra, amelyek távoli függősségeket idéznek elő, olyanokat, ahol egy ki nem próbált négyzet tartalma távoli négyzetek tartalmáról ad információt. [Segítség: vizsgáljon meg egy  $N \times 1$ -es táblát.]

- 7.12. Ez a feladat a klózok és az implikációs mondatok közötti kapcsolatot vizsgálja.

- (a) Mutassa meg, hogy a  $(\neg P_1 \vee \dots \vee \neg P_m \vee Q)$  klóz logikailag ekvivalens a  $(P_1 \wedge \dots \wedge P_m \Rightarrow Q)$  implikációs mondattal.

- (b) Mutassa meg, minden klóz (függetlenül a pozitív literálok számától) felírható  $(P_1 \wedge \dots \wedge P_m) \Rightarrow (Q_1 \vee \dots \vee Q_m)$  alakban, ahol  $P$ -k és  $Q$ -k ítéletkalkulus szimbólumok. Az ilyen mondatokat tartalmazó tudásbázist **implikatív normál formájúnak (implicative normal form)** vagy **Kowalski formájúnak (Kowalski form)** nevezzük.
- (c) Írja le a teljes rezolúciós szabályt implikatív normál formájú mondatokra.
- 7.13.** Ebben a feladatban tervezze tovább az áramkörön alapuló wumpus ágenst.
- Írjon egy a (7.4) egyenlethez hasonló egyenletet a *Nyíl* kijelentéshez, amelynek igaznak kell lennie, amikor az ágensnek még van nyíla.
  - Ismételje meg az (a) feladatot egy *ArccalJobbra* kijelentésre, ahol használja fel a (7.5) egyenletet modellként.
  - Készítsen változatokat a (7.7) és (7.8) egyenletekre a wumpus megtalálásához, és rajzolja fel az áramkört.
- 7.14.** Elemezze, hogy mit is jelent az *optimális* viselkedés a wumpus világban! Mutassa meg, hogy az IT-WUMPUS-ÁGENS definíciója nem optimális, és tegyen javaslatot a fejlesztésére!
- 7.15.** Bővítse ki az IT-WUMPUS-ÁGENS-t úgy, hogy képes legyen folyamatosan követni a releváns tényeket a tudásbázisban!
- 7.16.** Mennyi ideig tart a DPLL számára a bizonyítása, ahol  $\alpha$  egy literális, amelyet már *tartalmaz* a *TB*? Adjon magyarázatot!
- 7.17.** Kövesse a DPLL viselkedését a 7.15. ábrán leírt tudásbázison, amikor  $Q$ -t próbáljuk meg bizonyítani, és hasonlítsa össze ezt a működést az előrefelé láncolási algoritmussal.

# 8. ELSŐRENDŰ LOGIKA

*Ebben a fejezetben a világ objektumait és ezek kapcsolatait vizsgáljuk meg, és megpróbálunk érvelni ezek felhasználásával.*

A 7. fejezetben megmutattuk, hogy egy tudásbázisú ágens hogyan képes reprezentálni az őt körülvevő világot, és hogyan vezeti le a végrehajtandó cselekvéseket. Reprezentációs nyelvként az ítéletkalkulust alkalmaztuk, mert ez elégséges volt a logika és a tudásbázisú ágensek alapvető tulajdonságainak leírásához. Sajnos az ítéletkalkulus nyelvezte túlságosan korlátos ahhoz, hogy összetett környezetekről szerzett tudást megfelelő módon reprezentáljon. Ebben a fejezetben az elsőrendű logikát<sup>1</sup> (*first-order logic*) vizsgáljuk, amelynek kifejezőereje elégséges ahhoz, hogy a józan ésszel felfogható tudásunk nagy részét reprezentálja. Ezenkívül az elsőrendű logika sok más reprezentációs nyelvet magában foglal, vagy alapját képezi, és hosszú évtizedek óta vizsgálatok tárgya. A 8.1. alfejezetben a reprezentációs nyelvek általános bemutatásával kezdünk; a 8.2. alfejezetben az elsőrendű logika szintaxisát és szemantikáját tárgyaljuk; a 8.3. és 8.4. alfejezet leírja az elsőrendű logika használatát egyszerű reprezentációk esetén.

## 8.1. MÉG EGYSZER A REPREZENTÁCIÓRÓL

Ebben az alfejezetben a reprezentációs nyelvek természetét vizsgáljuk. Megállapításaink be fogják mutatni az elsőrendű logika fejlesztésének szükségességét, ami egy lényesen nagyobb kifejezőerővel rendelkező nyelv, mint a 7. fejezetben bemutatott ítéletkalkulus. Megnézzük majd, hogy az ítéletkalkulus és másfajta nyelvek mire képesek, és mire nem. A kérdést itt csak felületesen tárgyaljuk, néhány bekezdésbe sűrítve évszázadok gondolatait, kísérletezéseit és kudarcait.

A programnyelvek – mint például a C++, a Java vagy a Lisp – a használatban lévő formális nyelvek messze legnagyobb csoportját alkotják. A programok önmagukban, szó szerinti értelemben, csak számítási folyamatokat reprezentálnak. A programokon belüli adatstruktúrák tényeket reprezentálhatnak; például egy program használhatja a  $4 \times 4$ -es négyzetrács-elrendezést a wumpus világ elemeinek reprezentálásához. Így tehát az a programnyelvi állítás, hogy *Világ*[2, 2]  $\leftarrow$  *Csapda* viszonylag természetes módja annak a kijelentésnek, hogy van egy csapda a [2, 2] négyzetben. (Az illesfajta reprezentációkat *ad hoc* jellegűnek vélik; az adatbázisrendszeret pontosan azért fejlesztették ki, hogy általánosabb, tárgyterület-független módját biztosítsák a tények

<sup>1</sup> Elsőrendű predikátumkalkulusnak (*first-order predicate calculus*) is szokták nevezni, és néha ERPK-nak rövidítik.

tárolásának és visszakeresésének.) A programnyelvek hiányossága, hogy nem rendelkeznek általános mechanizmussal arra, hogy tényeket más tényekből levezessenek: az adatstruktúra minden frissítése egy tárgyterület-specifikus eljárás segítségével történik, amelynek részleteit a programozó vezeti le a tárgyterületről meglévő ismereti alapján. Ez a **procedurális (procedural)** megközelítés szembeállítható az ítéletkalkulus **deklaratív (declarative)** természetével, amelyben a tudás és az interferencia különálló fogalmak, a következtetés pedig teljes mértékben tárgyterületfüggő.

A programok adatstruktúráinak (és így az adatbázisoknak is) a másik hátránya annak a képességnak a hiánya, hogy egyszerűen lehessen kijelenteni például azt, hogy „Van egy lyuk a [2, 2]-ben vagy a [3, 1]-ben”, vagy azt, hogy „Ha a wumpus az [1, 1]-ben van, akkor nincs a [2, 2]-ben”. A programok egyetlen értéket tudnak tárolni minden változóhoz, néhány rendszer lehetővé teszi azt is, hogy az érték „ismeretlen” legyen, de hiányzik belőlük a részinformációk kezeléséhez szükséges kifejezőrő.

Az ítéletkalkulus egy deklaratív nyelv, mivel szemantikája mondatok és a lehetséges világok közötti igazságrelációkon alapul. Elégséges a kifejezőreje ahhoz is, hogy részinformációkat kezeljen, diszjunkciók vagy negációk használatával. Az ítéletkalkulusnak ezeken kívül egy harmadik tulajdonsága is van, ami kívánatos a reprezentatív nyelvek esetében, nevezetesen a **kompoziciós képesség (compositionality)**. Egy kompoziciós nyelvnél a mondat jelentése a mondatrészek jelentésének a függvénye. Például, a „ $B_{1,4} \wedge B_{1,2}$ ” kapcsolatban van a „ $B_{1,4}$ ” és a „ $B_{1,2}$ ” jelentésével. Nagyon furcsa lenne, ha a „ $B_{1,4}$ ” azt jelentné, hogy az [1, 4]-es négyzet bűdös, és a „ $B_{1,2}$ ” azt jelentné, hogy az [1, 2]-es négyzet bűdös, de a „ $B_{1,4} \wedge B_{1,2}$ ” már azt jelentné, hogy Franciaország és Lengyelország döntetlen játsszott a múlt heti jégkorong-selejtező mérkőzésen. Egyértelmű tehát, hogy a kompoziciós képesség hiánya nagyban megnehezíti a következető rendszerek működését.

Mint azt a 7. fejezetben láttuk, az ítéletkalkulusnak nincs elégséges kifejezőreje egy sok objektumos környezet **tömör** leírásához. Például kénytelenek voltunk minden egyes négyzetről önálló szabályt írni a csapdákra és a szellőkre:

$$S_{1,1} \Leftrightarrow (C_{1,2} \vee C_{2,1})$$

Magyarul, ezzel szemben, elégégegyszerűnek tűnik egyszer s mindenkorra kijelenteni, hogy „A lyukas négyzetek mellett lévők szellősek.” A természetes nyelvek szintaxisa és szemantikája lehetővé teszik a környezet tömör és összefogott leírását.

Első látásra úgy tűnik, hogy a természetes nyelvek (mint az angol vagy a magyar) valóban nagyon kifejezők. Képesek voltunk csaknem a teljes könyvet természetes nyelven megírni, és csak alkalmanként kellett áttérnünk más nyelvekre (beleértve a logikát, a matematikát és a diagramok nyelvét). Nagy múltú hagyomány a nyelvészettel és a nyelvek filozófiájában, hogy a természetes nyelvet elsősorban deklaratív tudásreprezentációs nyelvnek tekintjük, és megpróbáljuk rögzíteni ennek formális szemantikáját. Ha egy ilyen kutatás sikeres lenne, nagy előrelépést jelenthetne a mesterséges intelligencia számára, mert egy természetes nyelvet (vagy annak valamelyen származékát) használhatnánk a tudásreprezentációhoz vagy a következető rendszerekben.

A modern megközelítés szerint a természetes nyelv kissé más célt szolgál, nevezetesen a nyelv a **kommunikáció (communication)**, és nem annyira a tiszta reprezentáció eszköze. Amikor egy beszélő rámutat valamire, és azt mondja: „Nézd!” a hallgató meg tudja azt, hogy mondjuk, Superman végre megjelent a háztetők felett. Mégsem szeretnénk

azt kijelenteni, hogy a „Nézd!” mondatban benne foglaltatik ez a tény is. A mondat jelentése minden magától a mondattól, minden attól a szövegkörnyezettől (*context*) függ, amelyben a mondat elhangzott. Nyilvánvalóan nem várható, hogy csupán a „Nézd!” mondatot az adatbázisban tárolva visszaadható ennek a jelentése a szövegkörnyezet ismerete nélkül – amiből az a kérdés következik, hogy hogyan tudjuk megállapítani a szövegkörnyezetet reprezentálni. A természetes nyelvekből is hiányzik a kompozíciós képesség – egy olyan mondat jelentése, mint az „És akkor megláttá”, függhet az előtte álló és az utána következő mondatok által alkotott szövegkörnyezettől. Végül pedig a természetes nyelvek egyik gyenge pontja a többértelműség (*ambiguity*), ami nehézségeket okozhatna a következetetben. Amint azt Pinker megállapítja (Pinker, 1995): „Mikor az emberek az égre gondolnak, bizonyára nem zavarja meg őket, hogy vajon az égboltról van-e szó, vagy valamiről, ami lángol – és ha egy szó két gondolatot is megjeleníthet, akkor a gondolatok nem feleltethetők meg szavaknak.”

Azt a megközelítést követjük, hogy építünk az ítéletkalkulus alapjaira felhasználva, hogy ez egy deklaratív, kompozíciós szemantika, ami független a szövegkörnyezettől, és egyértelmű. Majd ezekre az alapokra egy még kifejezőbb logikát építünk azáltal, hogy felhasználunk a természetes nyelvenél alkalmazott reprezentációs megoldásokat, miközben igyekszünk elkerülni ennek a hátrányait. A természetes nyelv szintaxisát vizsgálva a legegyesrőlönben azonosítható összetevők a főnevek és a főnévi kifejezések, amelyek objektumokra (*objects*) utalnak (négyzetek, lyukak, wumpusok), valamint az igék és az igei kifejezés, amelyek az objektumok között fennálló relációkra (*relations*) utalnak (szellős, szomszédos, kilő). Ezeknek a relációknak egy része függvény (*function*) – azaz olyan relációk, amelyeknek csak egyetlen „értékükk” van egy adott „bemenethez”. Könnyen találhatunk példákat objektumokra, relációra és függvényekre:

- Objektumok: emberek, házak, számok, elméletek, Ronald McDonald, színek, baseballmeccs, háborúk, évszázadok...
- Relációk: lehetnek unáris relációk vagy tulajdonságok (*properties*), úgymint piros, kerek, színlelt, elsőrendű, többemeletes..., vagy általánosabb *n*-elemű relációk, úgymint testvére, nagyobb, belsejében, része, színe, utána történt, birtokol, kettő között van...
- Függvények: apja, legjobb barátja, harmadik hivatali ideje, eggyel több mint, kezdete...

Valójában minden állítás elképzelhető úgy, mint ami objektumokra, relációra és függvénykre vonatkozik. Íme néhány példa:

- „Egy plusz kettő egyenlő hárommal.”  
Objektumok: egy, kettő, három, egy meg kettő; Reláció: egyenlő; Függvény: plusz.  
„Egy plusz kettő a neve annak az objektumnak, amelyet úgy kapunk, hogy a „plusz” függvényt alkalmazzuk az „egy” és „kettő” objektumokra. A három egy másik neve ugyanennek az objektumnak.)
- „A wampus helyével szomszédos négyzetek büdösek.”  
Objektumok: wampus, négyzetek; Tulajdonság: büdös; Reláció: szomszédos.
- „A gonosz János király uralkodott Angliában 1200-ban.”  
Objektumok: János, Anglia, 1200; Reláció: uralkodott; Tulajdonságok: gonosz, király.

Az elsőrendű logika (*first order logic*) nyelvezete, amelynek a szintaxisát és szemantikáját a következő alfejezetben adjuk meg, objektumok és relációk köré épül. Az elsőrendű logika azért olyan fontos a matematika, a filozófia és a mesterséges intelligencia

számára, mert ezeket a területeket – amelyek valójában az emberi lét minden napjait írják le – tekinthetjük úgy, mint amelyek objektumokkal és a köztük lévő relációkkal dolgoznak. Az elsőrendű logika ezenkívül tényeket közölhet az univerzum *néhány* vagy *összes* objektumáról. Ez lehetővé teszi, hogy általános szabályokat vagy törvényszerűségeket tudunk megfogalmazni, mint például azt az állítást, hogy „A wumpus helyével szomszédos négyzetek büdösek.”

Az elsődleges különbség az ítéletlogika és az elsőrendű logika között az egyes nyelvek **ontológiai meghatározottságában** (*ontological commitment*) rejlik, abban, hogy mit feltételez a nyelv a *valóság* természetéről.

### A gondolat nyelve

A filozófusok és a pszichológusok sokat töprengtek azon, hogy az emberek és más élőlények hogyan jelenítik meg a tudást. Egyértelmű, hogy a természetes nyelv fejlődése fontos szerepet játszott abban, hogy az emberekben kifejlődött ez a képesség. Másrészt, számos pszichológiai eredmény azt sugallja, hogy az emberek nem alkalmazzák közvetlenül a nyelvet a belső reprezentációkhoz. Például a két mondat közül melyikkel kezdtük a 8.1. alfejezetet?

„Amelyben a világ objektumait és ezek kapcsolatait vizsgáljuk meg...”

„Ebben az alfejezetben a reprezentációs nyelvek természetét vizsgáljuk...”

Wanner (1974) úgy találta, hogy a kísérlet alanyai véletlenszerűen – körülbelül 50%-os gyakorisággal – jól választottak, de inkább az elolvasott szöveg tartalmára emlékeztek, több mint 90%-os biztonsággal. Ez azt sugallja, hogy az emberek a szavakat úgy dolgozzák fel, hogy egy nem verbális reprezentációra alakítják át, amit **emlékezetnek** (*memory*) nevezünk.

Továbbra is nagyon érdekes kérdez, hogy mi a pontos mechanizmusa annak, ahogyan a nyelv lehetővé teszi és formálja a gondolatok reprezentációját az emberekben. A híres **Sapir-Whorf-hipotézis** szerint az általunk beszélő nyelv nagymértékben befolyásolja gondolkodásunk és döntéseiink mikéntjét, különösen abban, ahogy kategóriarendszeret állítunk fel, amelyet felhasználva aztán felosztjuk a világot különböző fajta objektumokra. Whorf szerint (Whorf, 1956), azáltal, hogy az eszkimóknak számos szavuk van a hóra, másképpen is érzékelik a havat, mint más nyelvek beszélői. Néhány nyelvész vitatja ennek az állításnak a tényezőségét – Pullum úgy érvel (Pullum, 1991), hogy az inuitoknak, a yupiknak és más rokon nyelvet beszélőknek ugyanannyi szavuk van a hóval rokonítható fogalmakra, mint az angoloknak – vannak azonban olyanok is, akik támogatják az állítást (Fortescue, 1984). Vitathatatlanul igaznak tűnik, hogy az olyan embercsoportok, amelyeknek több ismeretük van a világnak egy bizonyos részéről, sokkal részletesebb szókincssel rendelkeznek erre a részre vonatkozóan – például a gyakorló rovarszakértők az általunk *bogaraknak* nevezett lényeket fajok százezreire osztják fel, és sokról ezek közül személyes ismeretekkel rendelkeznek. (Az evolúciós biológus J. B. S. Haldane meg is állapította, hogy a Teremtő „Bogarak iránti mértékű szeretete” különös.) Mi több, a tapasztalt sielőknek rengeteg szavuk van a hóra – porhó, latyak, tört krumpli, aludtej, kukorica, cukor, aszfalt, kordbársony, pehely, trutymó és így tovább –, és ezek olyan különbségeket mutatnak, amelyek ismeretlenek egy kívüllálló számára. Az ok-okozi viszony irányában azonban ismeretlen – kérdez, hogy a sielők csak azért ismerik fel a különbségeket, mert megtanulják a szavakat, vagy a különbségek keletkeznek először a személyes tapasztalat alapján, és ezeket címkezik fel aztán a közösségen már ismert fogalmakkal. Ez a kérdez különösen a gyermekek fejlődésének tanulmányozása szempontjából fontos. Egyelőre kevés ismeretünk van arról, hogy milyen mértékben fonódik egymásba a nyelvtanulás és a gondolkodás elsajtítása. Például egy olyan fogalom nevének ismerete, mint például az *aggregény*, megkönnyíti-e bonyolultabb, erre épülő fogalmak előállítását és használatát, amelyek tartalmazzák ezt a nevet – *partiképes aggregény*?

Az ítéletlogika feltételezi például, hogy a definiált tények fennállnak vagy nem állnak fenn a világban. minden ténynek két állapota lehet: igaz vagy hamis.<sup>2</sup> Az elsőrendű logika bonyolultabb feltételezésekre épít, nevezetesen, hogy a világ objektumokból épül fel, amelyek között relációk léteznek, amelyek vagy fennállnak, vagy nem. A speciális célú logikák további ontológiai hozzárendeléseket tesznek; például a **temporális logika (temporal logic)** feltételezi, hogy a tények csak bizonyos *időben* állnak fenn, és hogy ezek az idők – amelyek lehetnek időpontok és időtartamok is – rendezhetők. Így ezek a speciális célú logikák az objektumok egy bizonyos fajtájához (és az azokhoz tartozó axiómához) „első osztályú” státust rendelnek a logikán belül, ahelyett hogy egyszerűen meghatároznák őket a tudásbázison belül. A **magasabb rendű logika (higher-order logic)** úgy tekint az elsőrendű logikában relációknak és függvényeknek nevezettekre, mintha azok maguk is objektumok lennének. Ez lehetővé teszi, hogy minden relációról állításokat alkossunk – például arra, ha valaki azt szeretné meghatározni, mit jelent, hogy egy reláció tranzitív. Elterően más speciális célú logikáktól a magasabb rendű logika egyértelműen nagyobb kifejezőrővel rendelkezik, mint az elsőrendű logika, abban az értelemben, hogy néhány magasabb rendű logikai állítás nem írható le véges számú elsőrendű logikai állítással.

Egy logika jellemezhető még az **ismeretelméleti megállapításaival (epistemological commitments)** is – az egyes tényekhez megengedett tudásállapotokkal. Mind az ítéletkalkulusban, mind az elsőrendű logikában egy állítás egy tényt reprezentál, és az ágens vagy elhiszi, hogy az állítás igaz, vagy hamisnak véli, vagy nincs róla véleménye. Ezekben a logikákban tehát a tudásnak három lehetséges állapota van, bármely állítást is tekintjük. Ezzel szemben azok a rendszerek, amelyek a **valószínűség-elméletet (probability theory)** használják, tartalmazhatnak egy *hiedelemfokozatot*, ami a 0-tól (teljes hitetlenség) az 1-ig tart (teljes elfogadás).<sup>3</sup> Például, egy valószínűíthető wumpusvilágbeli ágens hiheti, hogy a wumpus az [1, 3]-ban van, 0,75-ös valószínűsséggel. Az öt különböző logikai rendszer ontológiai és ismeretelméleti megállapításait a 8.1. táblázatban foglaltuk össze.

Nyelv	Ontológiai meghatározottság (ami a világban létezik)	Ismeretelméleti hozzárendelések (amit az ágens elhisz a tényekről)
Ítéletlogika	Tények	Igaz/hamis/ismertetlen
Elsőrendű logika	Tények, objektumok, relációk	Igaz/hamis/ismertetlen
Temporális logika	Tények, objektumok, relációk, idők	Igaz/hamis/ismertetlen
Valószínűség-elmélet	Tények	Hiedelemfokozat $\in [0, 1]$
Fuzzy logika	Tények igazsági fokkal $\in [0, 1]$	Ismert intervallumérték

**8.1. táblázat.** Formális nyelvek, valamint azok ontológiai és ismeretelméleti hozzárendelései

<sup>2</sup> Ezzel szemben a fuzzy logikában (fuzzy logic) definiált az **igazság foka (degree of truth)**, ami egy 0 és 1 közötti érték. Például az a mondat, hogy „Bécs egy nagy város” a mi világunkban csak 0,6 fokig lehet igaz.

<sup>3</sup> Fontos, hogy ne keverjük össze a valószínűség-számításban alkalmazott hiedelemfokozatot a fuzzy logikában reprezentált igazsági fokozattal. Létezik néhány olyan bizonytalanságot kezelő rendszer, amely megenged határozatlanságot (a hiedelem fokát) az igazsági fokozatokkal kapcsolatban.

A következő alfejezetben elkezdjük az elsőrendű logika részletes tárgyalását. Ahogy egy fizikát tanuló diáknak valamelyen szinten rendelkeznie kell matematikai ismeretekkel, a mesterséges intelligenciát tanulmányozóknak is el kell sajátítaniuk bizonyos jártasságot a logikai jelölési rendszerek használatában. Másrészt az is nagyon fontos, hogy *ne* bonyolódjunk bele a logikai jelölési rendszerek sajátosságaiba – hiszen számos különböző változat létezik. A lényeges az, hogy tudatában legyünk annak, hogy a nyelv miként könnyíti meg tömör reprezentációk létrehozását, és hogy a szemantika miként vezet el helyes következtetési folyamatokhoz.

## 8.2. AZ ELSŐRENDŰ LOGIKA SZINTAXISA ÉS SZEMANTIKÁJA

Az alfejezetet azzal kezdjük, hogy pontosabban meghatározzuk azt a módot, amellyel az elsőrendű logikában előforduló lehetséges világok leírják az objektumokra és relációakra tett ontológiai megállapításokat. Ezután bemutatjuk a nyelv különböző elemeit, és ahogyan haladunk, megmagyarázzuk ezek szemantikáját.

### Az elsőrendű logika modelljei

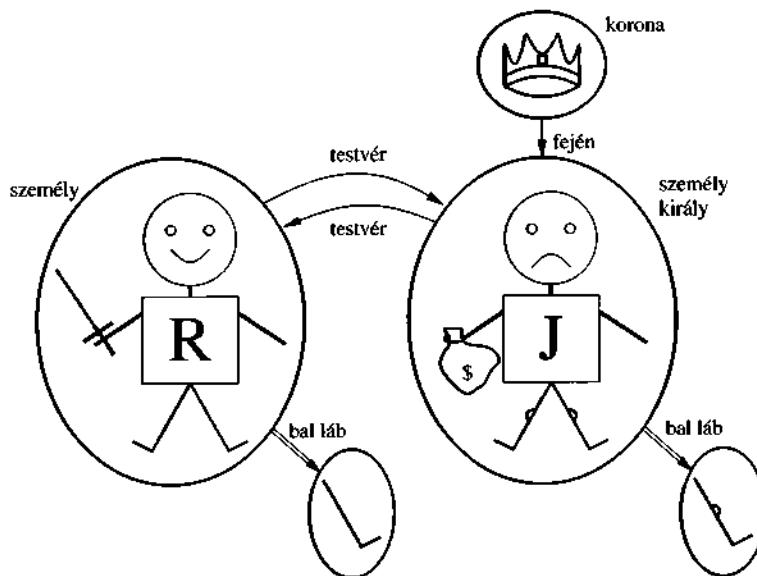
Emlékezzünk arra a 7. fejezetből, hogy a logikai nyelvek modelljei azok a formális struktúrák, amelyek a vizsgálat tárgyát képező lehetséges világokat alkotják. Az ítélet-kalkulus modelljei csupán az ítéletszimbólumok igazságértékeinek halmazai. Az elsőrendű logika modelljei ennél érdekesebbek. Először is, ezek tartalmaznak objektumokat! A modell **tárgyterülete (domain)** azoknak az objektumoknak a halmaza, amelyeket a tárgyterület tartalmaz; ezeket az objektumokat szoktuk a **tárgyterület elemeinek (domain elements)** nevezni. A 8.2. ábra bemutat egy modellt öt objektummal: Oroszlánszívű Richárd, Anglia királya 1189-től 1199-ig; öccse, gorosz János király, aki 1199-től 1215-ig uralkodott; Richárd és János bal lába; végül egy korona.

A modellben található objektumok többféle relációban lehetnek egymással. A 8.2. ábrán Richárd és János testvérek. Formálisan, egy reláció csak egymással kapcsolatban lévő objektumok **n-eleinek (tuple)** halmaza. (Egy *n*-es az objektumok egy rögzített sorrendben felsorolt gyűjteménye, amely objektumokat a ⟨ ⟩ zárójelpár vesz körbe.) Így tehát ebben a modellben a testvéreláció a következő halmaz:

{⟨Oroszlánszívű Richárd, János király⟩, ⟨János király, Oroszlánszívű Richárd⟩} (8.1)

(Itt magyarul neveztük meg az objektumokat, de képzeletben be lehetne helyettesíteni a neveket például képekkel is.) A korona János király fején van, tehát a „fején” relációhoz csak egy *n*-es tartozik (a korona, János király).

A „testvér” és a „fején” relációk bináris relációk – ez azt jelenti, hogy két objektum között állítanak fel kapcsolatot. A modell ezenkívül egyelemű, unáris relációkat vagy tulajdonságokat is tartalmaz: A „személy” tulajdonság egyaránt igaz Richárdra és Jánosra; a „király” tulajdonság csak Jánosra igaz (feltehetően azért, mert Richárd ekkor már halott volt); míg a „korona” tulajdonság csak a koronára igaz.



**8.2. ábra.** Egy öt objektumot – két bináris relációt, három egyelemű relációt (címkékkel jelezve az objektumokon) és egy egyargumentumú függvényt, bal láb – tartalmazó modell

A kapcsolatok egyes típusai leginkább függvényeknek tekinthetők, ahol egy adott objektumnak pontosan csak egy másik objektummal kell ilyen kapcsolatban lennie. Például minden egyes személynek egy bal lába van, tehát a modellben található egy „bal láb” függvény, ami a következő leképezéseket teszi lehetővé:

$$\begin{aligned} \langle \text{Oroszlánszívű Richárd} \rangle &\rightarrow \text{Richárd bal lába} \\ \langle \text{János király} \rangle &\rightarrow \text{János bal lába} \end{aligned} \quad (8.2)$$

Pontosabban megfogalmazva, az elsőrendű logika modelljei **totális függvényeket** (**total functions**) kívának meg, ami azt jelenti, hogy minden bemeneti  $n$ -eshez tartoznia kell egy értéknek. Így tehát a koronához tartoznia kell egy bal lábnak, és ennek így kell lennie minden bal láb esetében. Létezik erre a problémára egy technikai megoldás, amely egy hozzáadott „láthatatlan” objektumot vezet be: egy bal lábat rendel mindenhez, amelynek nincs bal lába, beleértve önmagát is. Szerencsére, amíg valaki nem tesz kijelentéseket a bal lábakról vagy olyan dolgoról, amelyeknek nincs bal lábuk, ezeknek a technikai problémáknak nincs jelentőségük.

## Szimbólumok és interpretációk

Most pedig nézzük a nyelv szyntaxát. A türelmetlen olvasó teljes leírást talál az elsőrendű logika formális nyelvtanáról a 8.3. ábrán.

Az elsőrendű logika alapvető szintaktikai elemei az objektumokat, relációkat és függvényeket megjelenítő szimbólumok. A szimbólumoknak tehát három típusa létezik: **konstansszimbólumok** (**constant symbols**), amelyek az objektumokat jelölik; **predi-**

**kátnaszimbólumok (predicate symbols)**, amelyek a relációkat jelenítik meg; és a **függvényszimbólumok (function symbols)**, amelyekkel függvényekre hivatkozhatunk. Azt a konvenciót fogjuk követni, hogy a szimbólumok minden nagybetűvel kezdődnek. Például használhatjuk a *Richárd* és *János* konstansszimbólumokat; a *Testvér*, *Fején*, *Személy*, *Király* és *Korona* predikátumszimbólumokat, valamint a *BalLáb* függvényszimbólumot. Úgy, mint az ítéletszimbólumoknál, a nevek megválasztása itt is teljes mértékben a felhasználóra van bízva. minden egyes predikátum- és függvényszimbólum együtt jár egy **számossággal (arity)**, ami behatárolja a paraméterek számát.

<i>Mondat</i>	$\rightarrow$	<i>AtomiMondat</i>
		( <i>Mondat Összekötőjel Mondat</i> )
		<i>Kvantor Változó... Mondat</i>
		$\neg$ <i>Mondat</i>
<i>AtomiMondat</i>	$\rightarrow$	<i>Predikátum(Term,...)</i>   <i>Term = Term</i>
<i>Term</i>	$\rightarrow$	<i>Függvény(Term,...)</i>
		<i>Konstans</i>
		<i>Változó</i>
<i>Összekötőjel</i>	$\rightarrow$	$\Rightarrow$   $\wedge$   $\vee$   $\leftrightarrow$
<i>Kvantor</i>	$\rightarrow$	$\forall$   $\exists$
<i>Konstans</i>	$\rightarrow$	<i>A</i>   <i>X<sub>1</sub></i>   <i>János</i>   ...
<i>Változó</i>	$\rightarrow$	<i>a</i>   <i>x</i>   <i>s</i>   ...
<i>Predikátum</i>	$\rightarrow$	<i>Mielőtt</i>   <i>Színes</i>   <i>Esik</i>   ...
<i>Függvény</i>	$\rightarrow$	<i>Anya</i>   <i>BalLába</i>   ...

**8.3. ábra.** Az elsőrendű logika (egyenlőséggel kiegészített) szyntaxa Backus–Naur-forma jelöléseivel. (Lásd az 1112. oldalt, ha ez a jelölés ismeretlen.) A szintaxis szigorú a zárójelezés tekintetében; a zárójelekről és az operátori elsőbbségről a 259. oldalon szereplő megjegyzések ugyanúgy vonatkoznak az elsőrendű logikára is.

A szemantika a mondatokat a modellekhez kapcsolja, és képesnek kell lennie az igazság meghatározására. Hogy ez megtörténjen, szükségünk van egy **interpretációra**, amely pontosan megjelöli, hogy mely objektumok, relációk és függvények felelnek meg a konstans-, predikátum- és függvényszimbólumoknak. Egy lehetséges interpretáció – amit **szándékolt interpretációnak (intended interpretation)** nevezünk – a mi esetünkben a következő:

- *Richárd Oroszlánszívű Richádra*, míg *János* a gonosz János királyra vonatkozik.
- A *Testvér* a testvéri kapcsolatra vonatkozik, ami a (8.1) egyenletben megadott *n*-es halmazban látható; a *Fején* arra a „fején” relációra vonatkozik, ami a korona és János király között áll fenn; a *Személy*, *Király* és *Korona* pedig arra az objektumhalmazra vonatkoznak, amelyek személyek, királyok és koronák.
- A *BalLáb* azt a „bal láb” függvényt jelenti, aminek a leképezését a (8.2) egyenletben adtuk meg.

Sok más lehetséges interpretáció is létezik, ami ezeknek a szimbólumoknak a modellhez való kapcsolatát írja le. Például egy lehetséges interpretációban *Richárd* jelenti a koronát és *János* a János király bal lábat. A modell öt objektumot tartalmaz, így 25 lehetséges interpretáció adható meg csak a *Richárd* és a *János* konstansszimbólumokhoz.

Figyeljük meg, hogy nem minden objektumnak van arra szüksége, hogy elnevezék – például a szándékolt interpretáció nem nevezi meg a koronát vagy a lábakat. Az is lehetséges, hogy egy objektumnak több neve legyen; létezik olyan interpretáció, amelyben a *Richárd* és a *János* is a koronára vonatkozik. Ha ez a lehetőség zavaró, emlékezzünk arra, hogy az ítéletlogikában teljességgel elfogadható egy olyan modell, amelyben a *Felhős* és a *Napos* egyaránt igaz. A tudásbázis feladata az, hogy kizárja azokat a modelleket, amelyek ellentmondásban vannak ismereteinkkel.

Bármely mondat igazságát egy modell és a mondat szimbólumainak az interpretációja határozza meg. Így tehát a vonzat, az érvényesség és más jellemzők az összes lehetséges modell és az összes lehetséges interpretáció termjeinek felhasználásával vannak meghatározva. Fontos megjegyezni, hogy a tárgyterület elemek száma minden egyes modellben korlátlan lehet – a tárgyterület elemei például lehetnek egész vagy valós számok. Ebből következően, a lehetséges modellek száma korlátlan, mint ahogy az interpretációk száma is. A maga után vonzásnak az összes lehetséges modell felsorolásával történő vizsgálata, ami az ítéletlogikában működik, nem lehetséges az elsőrendű logikában. Még ha az objektumok száma korlátos is, a kombinációk száma nagyon nagy lehet. A példánkban szereplő objektumokkal hozzávetőlegesen  $10^{25}$  a kombinációk száma, egy öt objektummal rendelkező tárgyterületben (lásd 8.5. feladat).

## Termek

Egy **term** (**term**) egy objektumra vonatkozó logikai kifejezés. A konstansszimbólumok tehát termek, de nem minden kényelmes önálló szimbólumot alkalmazni minden egyes objektum megnevezésénél. Magyarul inkább a „János király bal lába” kifejezést használjuk, mintsem nevet adjunk a lábának. Ezért használunk függvényszimbólumokat: a konstansszimbólumok használata helyett inkább a *BalLáb(János)*-t alkalmazzuk. Általános esetben egy összetett term egy függvényszimbólumból áll, amelyet a függvényszimbólum argumentumainak zárójelezett listája követ. Fontos megjegyezni, hogy egy összetett term nem más, mint egy bonyolult formájú név, és nem egy „értéket visszaadó szubrutin hívás”. Nem egy *BalLáb* szubrutinról van szó, amely bemenetként megkap egy személyt, és visszaad egy lábat. Tudunk következtetni a bal lábakkal kapcsolatban (pl. kiindulva az általános szabályból, hogy mindenkinél van egy bal lába, és ebből levezethető, hogy Jánosnak is lennie kell) anélkül, hogy valaha is megadnánk a *BalLáb* definícióját. Ez is olyasmi, amit nem tehetünk meg a programozási nyelvek szubrutinjaival.<sup>4</sup>

<sup>4</sup> A  $\lambda$ -kifejezések ( $\lambda$ -expressions) hasznos jelölést biztosítanak, amelyek az új függvényszimbólumokat futási időben hozzák létre. Például a függvény, amely négyzetre emeli a számokat, leírható úgy, hogy  $(\lambda x. x \times x)$ , és alkalmazható az argumentumokra, akár bármely más függvényszimbólum. Egy  $\lambda$ -kifejezést úgy is meghatározhatunk és használhatunk, mint egy predikáumszimbólumot (lásd 22. fejezet). A Lámbda operátor a Lispen pontosan ugyanezt a szerepet játszsa. Vegyük észre, hogy a  $\lambda$  használata ilyen módon nem növeli meg az elsőrendű logika formális kifejezberejét, mert az a mondat, ami tartalmaz egy  $\lambda$ -kifejezést, átirtható úgy, hogy hozzácsatoljuk a paramétereket, ami egy ugyanolyan mondatot eredményez.

A termek formális szemantikája nyilvánvaló. Tekintsük az  $f(t_1, \dots, t_n)$  termet. Az  $f$  függvényszimbólum a modellben található valamely függvényre (nevezzük  $F$ -nek) vonatkozik; az argumentum termek a tárgyterületben fellelhető objektumokra vonatkoznak (nevezzük őket  $d_1, \dots, d_n$ -nek); és a term egészében arra az objektumra vonatkozik, ami a  $d_1, \dots, d_n$ -re alkalmazott  $F$  függvény értéke. Például tételezzük fel, hogy ha a *BalLáb* függvényszimbólum a (8.2) egyenletben mutatott függvényre vonatkozik, és a János vonatkozik János királyra, akkor a *BalLáb(János)* János király bal lábat jelöli. Ezen a módon az interpretáció rögzíti minden termnek a referenciáit.

## Atomi mondatok

Most, hogy vannak objektumokra vonatkozó termeink és relációkra vonatkozó predikátszimbólumaink, összerakhatjuk őket, hogy tényeket kifejező **atomi mondatokat** (*atomic sentences*) állítsunk elő. Egy atomi mondatot egy predikátumszimbólum és az azt követő zárójelezett listában található termek listája alkot. Például a

*Testvér(Richárd, János)*

azt jelenti egy korábban megadott szándékolt interpretáció szerint, hogy Oroszlánszívű Richárd testvére János királynak.<sup>5</sup> Az atomi mondatoknak is lehetnek összetett termek az argumentumai. Így tehát a

*Házas(Apja(Richárd), Anyja(János))*

azt jelenti, hogy Oroszlánszívű Richárd apja feleségül vette János király anyját (ismét csak egy alkalmas interpretációban).

 *Egy atomi mondat igaz (true) egy adott modellben, egy adott interpretáció mellett, ha a reláció, amire a predikátumszimbólum vonatkozik, fennáll az argumentumok által jelölt objektumok között.*

## Összetett mondatok

Összetett mondatok létrehozásához használhatunk **logikai összekötőjeleket** (*logical connectives*) úgy, mint az ítéletkalkulusban. A logikai kötőszavakkal formált mondatok szemantikája azonos az ítéletkalkulusban látott szemantikával. Íme négy mondat, amelyek igazak a 8.2. ábra modelljében, a szándékolt interpretációból:

¬*Testvér(BalLáb(Richárd), János)*

*Testvér(Richárd, János)  $\wedge$  Testvér(János, Richárd)*

*Király(Richárd)  $\vee$  Király(János)*

¬*Király(Richárd)  $\Rightarrow$  Király(János)*

<sup>5</sup> Rendszerint követjük az argumentumok rendezésének konvencióját, amely szerint a  $P(x,y)$ -t úgy értelmezzük, hogy az „ $x$  az  $y$ -nak a  $P$ -je”.

## Kvantorok

Ha egyszer létrehoztunk egy objektumokat is tartalmazó logikát, természetes, hogy objektumok egész gyűjteményeire vonatkozó tulajdonságokat is ki akarunk fejezni anélkül, hogy az objektumokat nevük felhasználásával fel kellene sorolni. Ezt a **kvantorok (quantifiers)** alkalmazása teszi lehetővé. Az elsőrendű logika két standard kvantort tartalmaz, amelyeket *univerzális* és *egzisztenciális* kvantoroknak nevezünk.

### Univerzális kvantor ( $\forall$ )

Emlékezzünk vissza arra, hogy a 7. fejezetben milyen nehézségeink voltak általános szabályok ítéletlogikában történő megfogalmazásával. Az olyan szabályok, mint az „A wumpus mellett található négyzetek büdösek” vagy a „Minden király személy” adják meg a savát-borsát az elsőrendű logikának. Az első szabállyal a 8.3. alfejezetben fogunk foglalkozni. A második szabály, „Minden király személy”, az elsőrendű logikában így írható le:

$$\forall x \text{ Király}(x) \Rightarrow \text{Személy}(x)$$

A  $\forall$ -t általában „Minden ...re”-nek olvassuk ki. A mondat tehát azt jelenti: „Minden  $x$ -re, ha  $x$  egy király, akkor  $x$  egy személy.” Az  $x$  szimbólum neve **változó (variable)**. A konvenció alapján a változókat kisbetűvel írjuk. Egy változó önmagában egy term, és mint olyan, szerepelhet egy függvény argumentumaként – például:  $BalLáb(x)$ . Egy változó nélküli termet **alaptermnek (ground term)** nevezünk.

Egyszerűen a  $\forall x P$  mondat, ahol a  $P$  egy tetszőleges logikai kifejezés, azt mondja ki, hogy a  $P$  igaz minden  $x$  objektumra. Pontosabban,  $\forall x P$  igaz egy adott modellre az interpretáció belül, ha  $P$  igaz minden lehetséges **kiterjesztett interpretációban (extended interpretations)**, amelyet az adott interpretáció felhasználásával hozunk létre, ahol minden egyes kiterjesztett interpretáció meghatároz egy tárgyterületeket, amire az  $x$  vonatkozik.

Ez bonyolultnak hangszik, de valójában csak egy óvatos módja annak, hogy megállapítsuk az univerzális kvantorok intuitív jelentését. Tekintsük a 8.2. ábrát és a hozzá tartható szándékolt interpretációt. Ötféle módon tudjuk kiterjeszteni az interpretációt:

- $x \rightarrow$  Oroszlánszívű Richárd
- $x \rightarrow$  János király
- $x \rightarrow$  Richárd bal lába
- $x \rightarrow$  János bal lába
- $x \rightarrow$  a korona

A  $\forall x \text{ Király}(x) \Rightarrow \text{Személy}(x)$  univerzális kvantort alkalmazó mondat igaz az eredeti interpretációban, ha a  $\text{Király}(x) \Rightarrow \text{Személy}(x)$  mondat igaz mind az öt kiterjesztett interpretációra. Eszerint az univerzális kvantorral ellátott mondat egyenértékű a következő öt mondattal:

Oroszlánszívű Richárd egy király  $\Rightarrow$  Oroszlánszívű Richárd egy személy

János király egy király  $\Rightarrow$  János király egy személy

Richárd bal lába egy király  $\Rightarrow$  Richárd bal lába egy személy

János bal lába egy király  $\Rightarrow$  János bal lába egy személy

A korona egy király  $\Rightarrow$  a korona egy személy

Nézzük meg alaposan ezt az állításhalmazt. Mivel a mi modellünkben János király az egyetlen király, a második mondat kijelenti, hogy ő egy személy, ahogy azt reméltek is. De mi a helyzet a többi négy mondattal, amelyek azért szerepelnek, hogy állításokat fogalmazzanak meg lábakról és koronáról? Része ez annak a jelentésnek, hogy „Minden király személy”? Valójában, a többi négy állítás is igaz a modellben, de egyáltalán nem állítanak semmit a lábak, koronák vagy akár Richárd személyes tulajdonságairól. Ez azért van, mert ezen objektumok egyike sem király. Ha megnézzük a ( $\Rightarrow$ ) összekötőjel igazságátláját (7.8. ábra), láthatjuk, hogy az implikáció igaz, amikor a premisszája hamis – *tekintet nélkül* a konklúzió igazságártalmára. Így tehát az univerzális kvantorral ellátott mondat kijelentése egyenértékű azzal, hogy egyedi implikációk teljes listáját definiáljuk. Végül is egy szabály konklúzióját adjuk még azokra az objektumokra, amelyek premisszája igaz, de nem mondunk egyáltalán semmit azokról az egyedekről, akirek nézve a premissza hamis. Így az implikáció igazságátlájának sorai tökéletesnek bizonyulnak univerzális kvantorokat tartalmazó általános szabályok megírásához.

Gyakori hiba, amelyet még figyelmes olvasók is elkövetnek, akik pedig már többször is elolvasták ezt a bekezdést, hogy konjunkciót használnak implikáció helyett. A

$$\forall x \text{ Király}(x) \wedge \text{Személy}(x)$$

mondat egyenértékű lenne azzal az állítással, hogy

$$\text{Oroszlánszívű Richárd egy király} \wedge \text{Oroszlánszívű Richárd egy személy}$$

$$\text{János király egy király} \wedge \text{János király egy személy}$$

$$\text{Richárd bal lába egy király} \wedge \text{Richárd bal lába egy személy}$$

és így tovább. Nyilvánvaló, hogy ez nem azt jelenti, mint amit szeretnénk elérni.

## Egzisztenciális kvantorok ( $\exists$ )

Az univerzális kvantor állításokat fogalmaz meg minden objektumról. Hasonlóan, az egzisztenciális kvantor felhasználásához állításokat tehetünk az univerzum *néhány* objektumáról anélkül, hogy megneveznénk azokat. Azt az állítást tehát, hogy János királynak koronája van, így is leírhatjuk:

$$\exists x \text{ Korona}(x) \wedge \text{Fején}(x, \text{János})$$

Az  $\exists x$  kiolvasása „Létezik egy olyan  $x$ ...” vagy „Néhány  $x$ -re...” .

Érthetően fogalmazva, az  $\exists x P$  mondat azt jelenti, hogy a  $P$  legalább egy  $x$  objektusra igaz. Pontosabban az  $\exists x P$  igaz egy adott modellben, egy adott interpretáció belül, ha a  $P$  igaz *legalább egy* kiterjesztett interpretációban, amely az  $x$ -et a tartomány egy eleméhez rendeli. A mi példánkban ez azt jelenti, hogy a következő állítások közül legalább egy igaz:

$$\text{Oroszlánszívű Richárd egy korona} \wedge \text{Oroszlánszívű Richárd János fején van}$$

$$\text{János király egy korona} \wedge \text{János király János fején van}$$

$$\text{Richárd bal lába egy korona} \wedge \text{Richárd bal lába János fején van}$$

$$\text{János bal lába egy korona} \wedge \text{János bal lába János fején van}$$

$$\text{A korona egy korona} \wedge \text{a korona János fején van}$$

Az ötödik állítás igaz a modellben, így az eredeti egzisztenciális kvantorral ellátott mondat igaz a modellben. Vegyük észre, hogy definíciónk szerint a mondat igaz lenne egy olyan modellben is, ahol János király két koronát visel. Ez teljes mértékben konzisztens az eredeti mondattal, miszerint „János királynak korona van a fején.”<sup>6</sup>

Mint ahogyan természetesnek tűnik az  $\Rightarrow$  használata összekötőjelnek a  $\forall$  használatakor, az  $\wedge$  az a természetes összekötőjel, amit az  $\exists$ -vel használunk. Az  $\wedge$  használata fő összekötőjelként az  $\forall$ -val túlságosan is erős állításokhoz vezetett az előző alfejezetben tárgyalt példában; a  $\Rightarrow$  használata az  $\exists$ -vel igazából nagyon gyenge állításhoz vezet:

$$\exists x \text{ Korona}(x) \Rightarrow \text{Fején}(x, \text{János})$$

A felszínén ez a mondatunk elfogadható átírásának tűnik. A szemantikát alkalmazva látjuk, hogy a mondat azt jelenti ki, hogy a következő állítások legalább egyike igaz:

Oroszlánszívű Richárd egy korona  $\Rightarrow$  Oroszlánszívű Richárd János fején van

János király egy korona  $\Rightarrow$  János király János fején van

Richárd bal lába egy korona  $\Rightarrow$  Richárd bal lába János fején van

És így tovább. Mármost, egy implikáció akkor igaz, ha a premissza és a konklúzió egyaránt igaz, vagy ha a premisszája hamis. Tehát ha Oroszlánszívű Richárd nem egy korona, akkor az első állítás igaz, és az egzisztenciális kvantort kielégítettük. Így tehát egy egzisztenciális kvantort tartalmazó implikációs mondat igaz bármely olyan modellben, amely tartalmaz egy olyan objektumot, amelyre az implikáció premisszája hamis; következésképpen az ilyen mondatok valójában nem túl sokat mondanak.

## Egymásba ágyazott kvantorok

Gyakran szükségünk van még összetettebb mondatok kifejezésére, amelyek több kvantort is használnak. A legegyszerűbb eset az, amikor a kvantorok azonos típusúak. Például „A fivérek testvérek” mondat leírható úgy, hogy:

$$\forall x \forall y \text{ Fivér}(x, y) \Rightarrow \text{Testvér}(x, y)$$

Az egymást követő azonos típusú kvantorokat írhatjuk úgy is, hogy egy kvantort alkalmazunk az összes változóra. Például hogy azt állítsuk, a testvérség egy szimmetrikus kapcsolat, ezt írhatjuk:

$$\forall x, y \text{ Testvér}(x, y) \Leftrightarrow \text{Testvér}(y, x)$$

Más esetekben kevert jelöléssel van dolgunk. A „ mindenki szeret valakit” azt jelenti, hogy minden személyhez tartozik valaki, akit a személy szeret:

$$\forall x \exists y \text{ Szeret}(x, y)$$

Másrészt, hogy azt állítsuk: „Van valaki, akit mindenki szeret”, ezt írjuk:

$$\exists y \forall x \text{ Szeret}(x, y)$$

<sup>6</sup> Létezik az egzisztenciális kvantornak egy variánsa, amelyet  $\exists^1$ -gyel vagy  $\exists!$ -lel jelölünk, és ami azt jelenti, hogy „Létezik pontosan egy...”. Ugyanez a jelentés kifejezhető egyenlőségállításokkal is, mint azt a 8.2. alfejezetben bemutattuk.

A kvantorok sorrendje tehát nagyon fontos. Még világosabbá válik, ha zárójeleket is alkalmazunk. A  $\forall x (\exists y \text{ Szeret}(x, y))$  azt jelenti, hogy mindenki van egy bizonyos tulajdonsága, nevezetesen, az a tulajdonság, hogy mindenki szeret valakit. Másrészt, a  $\exists x (\forall y \text{ Szeret}(x, y))$  azt jelenti, hogy a világban valakinek van egy bizonyos tulajdonsága, méghozzá az, hogy mindenki szereti őt.

Zavart okozhat, ha két kvantort használunk ugyanazzal a változó névvel. Tekintsük a következő mondatot:

$$\forall x [\text{Korona}(x) \vee (\exists x \text{ Fivér}(Richárd, x))]$$

Itt az  $x$  a *Fivér(Richárd, x)*-ben *egzisztenciális* kvantorral van ellátva. Az a szabály, hogy a változó azzal a legbelső kvantorhoz tartozik, ami a változót megemlíti, és utána a változó nem tárgya semmilyen kvantor alárendelésnek.<sup>7</sup>

A másik módja a megoldásnak:  $\exists x \text{ Fivér}(Richárd, x)$ , ahol ez egy Richádról szóló mondat (hogy neki van egy fivére), és nem pedig az  $x$ -ről. Így tehát annak nem lesz hatása, ha kívül elő teszünk egy  $\forall x$ -et. Éppen így írhattuk volna azt is, hogy  $\exists z \text{ Fivér}(Richárd, z)$ . Mivel ez tévesztés forrása lehet, ezért minden különböző változókat fogunk használni.

## Az $\forall$ és az $\exists$ kapcsolata

A két kvantor valójában szoros kapcsolatban áll egymással a negáción keresztül. Ha valaki azt mondja, hogy mindenki utálja a paszternákat, akkor egyúttal azt is mondja, hogy nem létezik olyan valaki, aki szereti azt, és fordítva:

$$\forall x \neg \text{Szereti}(x, \text{Paszternák}) \text{ egyenértékű azzal, hogy } \neg \exists x \text{ Szereti}(x, \text{Paszternák})$$

Egy lépéssel tovább is mehetünk. A „Mindenki szereti a fagylaltot” mondat azt jelenti, hogy nincs olyan, aki ne szeretné a fagylaltot:

$$\forall x \text{ Szereti}(x, \text{Fagylalt}) \text{ egyenértékű azzal, hogy } \neg \exists x \text{ Szereti}(x, \text{Fagylalt})$$

Mivel az  $\forall$  igazából egy konjunkció minden univerzumbeli objektum felett, az  $\exists$  pedig egy diszjunkció ugyanezen objektumok felett, ezért nem meglepő, hogy eleget tesznek a De Morgan-szabályoknak. A kvantorral és a kvantor nélküli álló mondatokra vonatkozó De Morgan-szabályok a következők:

$$\begin{array}{ll} \forall x \neg P \equiv \neg \exists x P & \neg P \wedge \neg Q \equiv \neg(P \vee Q) \\ \neg \forall x P \equiv \exists x \neg P & \neg(P \wedge Q) \equiv \neg P \vee \neg Q \\ \forall x P \equiv \neg \exists x \neg P & P \wedge Q \equiv \neg(\neg P \vee \neg Q) \\ \exists x P \equiv \neg \forall x \neg P & P \vee Q \equiv \neg(\neg P \wedge \neg Q) \end{array}$$

<sup>7</sup> Az azonos változónak használó kvantorok interferenciára való képessége teszi lehetővé a kiterjesztett interpretációk kissé barokkos mechanizmusát a kvantorral ellátott mondatok szemantikájában. Az intuitív legegyértelműbb megközelítése az objektumok helyettesítésének az  $x$  minden előfordulásánál nem működik a példánkban, mert az  $x$ -et a *Fivér(Richárd, x)*-ben „foglyul ejtené” a behelyettesítés. A kiterjesztett interpretációk ezt pontosabban oldják meg, mert a belső kvantor hozzákapcsolása az  $x$  termhez kioltja a külső kvantor hatását.

Így valójában nincs szükségünk egyszerre mind az  $\forall$ -ra, mind az  $\exists$ -re, mint ahogy nincs szükség egyszerre mind az  $\wedge$ -ra és az  $\vee$ -ra sem. Mégis, mivel az olvashatóság fontosabb, mint a takarékokosság, használni fogjuk minden két kvantort.

## Egyenlőség

Az elsőrendű logika a predikátumok és termek korábban leírt használatán kívül még egy lehetőséget nyújt atomi mondatok készítésére. Alkalmazhatjuk az **egyenlőségszimbólumot** (*equality symbol*) állítások készítésére is annak kifejezésére, hogy két term ugyanarra az objektumra vonatkozik. Például az

$$Apja(János) = Henrik$$

azt mondja ki, hogy az *Apja(János)* által hivatkozott objektum és a *Henrik* által hivatkozott objektum azonos. Mivel az interpretáció rögzíti bármely term referenciáit, egy egyenlőségi állítás igazságának megállapításához elegendő azt látni, hogy a két term referenciái ugyanazok az objektumok.

Az egyenlőségszimbólumot használhatjuk arra, hogy tényeket állapítsunk meg egy adott függvényről, ahogy ezt megtettük az imént az *Apja* szimbólummal. Alkalmazhatjuk az egyenlőségszimbólumot negációval is, annak a kifejezésére, hogy két term nem ugyanaz az objektum. Ha azt akarjuk állítani, hogy Richárdnak legalább két fivére van, ezt írjuk:

$$\exists x, y \text{ } Fivér(x, \text{Richárd}) \wedge Fivér(y, \text{Richárd}) \wedge \neg(x = y)$$

Annak a mondatnak, hogy

$$\exists x, y \text{ } Fivér(x, \text{Richárd}) \wedge Fivér(y, \text{Richárd})$$

nem ugyanaz a szándékolt jelentése. Ez a mondat például igaz a 8.2. ábrán látható modellre, ahol Richárdnak csak egy fivére van. Hogy ezt megértsük, figyeljük meg azt a kiterjesztett interpretációt, ahol mind az  $x$ , mind az  $y$  János királyhoz van rendelve. A  $\neg(x = y)$  viszont már nem eleme a modellnek. Az  $x \neq y$ -t néha rövidítésként használjuk a  $\neg(x = y)$  kifejezésére.

## 8.3. AZ ELSŐRENDŰ LOGIKA HASZNÁLATA

Most, hogy definiáltunk egy kifejező logikai nyelvet, itt az ideje, hogy megtanuljuk használni is. A legjobb módja ennek az, ha példákon keresztül tessziük. A logikai szintaxis többféle aspektusának illusztrálására már láttunk néhány egyszerű mondatot; ebben az alfejezetben az egyszerű tárgyterületeknek (*domain*) több szisztematikus reprezentációját fogjuk bemutatni. A tudásreprezentációban egy tárgyterület a világnak az a része, amelyről valamilyen tudást akarunk kifejezni.

Az elsőrendű tudásbázisokra definiálható KJELENT/KÉRDEZ eljárások rövid leírásával fogunk kezdeni. Ezután megvizsgáljuk a családi kapcsolatok, a számok, a halmazok és a listák tárgyterületét, valamint a wumpus világot. A következő alfejezet egy még alapvetőbb példát (elektronikus áramkörök) tartalmaz, míg a 10. fejezet minden áttekint az univerzumban.

## Kijelentések és lekérdezések az elsőrendű logikában

A mondatokat a KIELENT használatával adjuk hozzá a tudásbázishoz, pontosan úgy, mint az ítéletlogikában. Ezeket a mondatokat **kijelentéseknek (assertions)** nevezzük. Például, azt jelenthetjük ki, hogy János egy király, és hogy a királyok személyek:

$\text{KIELENT}(\text{TB}, \text{Király(János)})$

$\text{KIELENT}(\text{TB}, \forall x \text{ Király}(x) \Rightarrow \text{Személy}(x))$

Kérdéseket tehetünk fel a tudásbázisról a KÉRDEZ használatával. Például a:

$\text{KERDEZ}(\text{TB}, \text{Király(János)})$

*igazat* ad vissza. A KÉRDEZ-zel feltett kérdéseket **lekérdezéseknek (queries)** vagy **célokknak (goals)** nevezzük (ne tévesszük össze azokkal a célokkal, amelyek az ágensek kívánt állapotát írják le). Általánosságban, bármely lekérdezésre, ami logikusan következik a tudásbázisból, igenlő választ kell adni. Például ha adott a két kijelentés az előző bekezdésben, akkor a

$\text{KÉRDEZ}(\text{TB}, \text{Személy(János)})$

lekérdezésnek szintén azt kell visszaadnia, hogy *igaz*. Kvantorokat alkalmazó kérdéseket is feltehetünk, úgy mint:

$\text{KÉRDEZ}(\text{TB}, \exists x \text{ Személy}(x))$

Erre a lekérdezésre a válasz lehet *igaz*, de ez nem segít, és nem is szórakoztató. (Ez olyasmi, mintha a „Meg tudná mondani mennyi az idő?” kérdésre azt válaszolnánk, hogy „Igen”.) Egy egzisztenciális változókkal ellátott lekérdezés valójában azt a kérdést teszi fel, hogy „Van-e olyan  $x$ , amely...”, és úgy oldjuk meg, hogy megadunk egy ilyen  $x$ -et. Egy ilyen típusú válasz szokásos formája egy **helyettesítési lista (substitution list)** vagy **lekötési lista (binding list)**, amely változó/term párok halmaza. Ebben az esetben, amikor csak két kijelentés szerepel, a válasz az  $\{x/János\}$  lenne. Ha több lehetőséges válasz van, akkor a helyettesítések listáját kapjuk vissza.

## A rokonsági tárgyterület

Az első példában a családi kapcsolatokat, vagyis a rokonsági tárgyterületet vizsgáljuk meg. Ez a tárgyterület tartalmaz olyan tényeket, mint „Erzsébet Károly anyja” és „Károly Vilmos apja”, valamint olyan szabályokat, mint „Valakinek a nagyanyja az ő szülőjének az anyja”.

Egyértelmű, hogy ennek a tárgyterületnek az objektumai személyek. Két unáris predikátumunk lesz a *Férfi* és a *Nő*. A rokonsági kapcsolatokat – szülőség, testvérség, házasság és így tovább – bináris predikátumokkal fogjuk ábrázolni: *Szülője*, *Testvére*, *Fivére*, *Nővére*, *Gyereke*, *Lánya*, *Fia*, *Házastársa*, *Felesége*, *Férje*, *Nagyszülője*, *Unokatestvére*, *Nagynénje* és *Nagybátyja*. Az *Anyja* és *Apja* reprezentálására függvényeket fogunk alkalmazni, mivel minden személynak pontosan egy ilyen rokona van (legalább is a természet rendje szerint).

Leírhatjuk a predikátumokat és a függvényeket úgy, hogy más szimbólumokkal fejezzük ki, amit róluk tudunk. Például valakinek az anyja, az ő nőnemű szülője:

$$\forall m, c \text{ Anyja}(c) = m \Leftrightarrow \text{Nő}(m) \wedge \text{Szülője}(m, c)$$

Valakinek a férje az ő férfi házastársa:

$$\forall w, h \text{ Férfi}(h, w) \Leftrightarrow \text{Férfi}(h) \wedge \text{Házastársa}(h, w)$$

A nő és a férfi diszjunkt kategóriák:

$$\forall x \text{ Férfi}(x) \Leftrightarrow \neg \text{Nő}(x)$$

A szülője és a gyereke inverz relációk:

$$\forall p, c \text{ Szülője}(p, c) \Leftrightarrow \text{Gyereke}(c, p)$$

Egy nagyszülő a szülője valaki szülőjének:

$$\forall g, c \text{ Nagyszülője}(g, c) \Leftrightarrow \exists p \text{ Szülője}(g, p) \wedge \text{Szülője}(p, c)$$

Egy testvér a másik gyereke ugyanannak a szülőnek:

$$\forall x, y \text{ Testvére}(x, y) \Leftrightarrow x \neq y \wedge \exists p \text{ Szülője}(p, x) \wedge \text{Szülője}(p, y)$$

Folytathatnánk így tovább, és a 8.11. feladatban kérni is fogjuk, hogy tegye ezt meg.

Minden egyes ilyen mondat tekinthető a rokonsági tárgyterület egy **axiómájának (axiom)**. Az axiómákat általában a tisztán matematikai tárgyterületekhez kötjük – röviden meg fogunk vizsgálni néhány axiómát számokra – de valamennyi tárgyterületben szükség van rájuk. Az axiómák ellátnak bennünket alapvető ténybeli információkkal, amelyekből hasznos következtetéseket vonhatunk le. Rokonsági axiómáink egyúttal **definíciók (definitions)** is; formájuk pedig  $\forall x, y P(x, y) \Leftrightarrow \dots$ . Az axiómák meghatározzák az *Anyja* függvényt és a *Férje*, *Férfi*, *Szülője*, *Nagyszülője* és *Testvére* predikátumokat más predikátumok felhasználásával. A predikátumok alapvető halmazainál (*Gyereke*, *Házastársa* és *Nő*) a definícióinkat már nem tudjuk más predikátumokkal leírni. Ez természetes módja annak, ahogyan felépítjük egy tárgyterület reprezentációját, és analóg azzal, ahogyan a szoftvereket felépítik primitív könyvtárfüggvények szubrutinjainak az egymást követő definícióival. Vegyük észre, hogy a predikátumprimitíveknek nincs feltétlenül egyetlen kizárolagos halmaza; éppen ilyen jól használhattuk volna a *Szülője*, *Házastársa* és *Férfi* predikátumokat. Néhány tárgyterületben, mint azt látni fogjuk, nincs egyértelműen meghatározható alaphalmaz.

Nem minden a tárgyterületről szóló logikus mondat axióma. Néhány közülük **tétel (theorem)** – amelyeket az axiómák vonnak maguk után. Például vizsgáljuk meg a kitelentést, hogy a testvérseg szimmetrikus:

$$\forall x, y \text{ Testvére}(x, y) \Leftrightarrow \text{Testvére}(y, x)$$

Ez vajon egy axióma vagy egy téTEL? Valójában ez egy téTEL, amely logikusan következik a testvérseget meghatározó axiómából. Ha ezt a mondatot KÉRDEZ-zük a tudásbázistól, akkor az eljárásnak az *igaz* eredményt kell visszaadnia.

Tisztán logikai nézőpontból egy tudásbázis csak axiómákat, és nem tételeket tartalmaz, mert a tételek nem növelik meg a konklúziók halmazát, amelyeket a tudásbázis maga után vonz. Gyakorlati szempontból a tételek elengedhetetlenek az új mondatok

előállításánál a számítási költségek csökkentéséhez. Nélküük egy bizonyítási rendszernek minden egyes alkalommal az első alapelvktől kellene indulnia, mintha egy fizikusnak mindig újra le kellene vezetnie a számítások szabályait az összes új problémánál.

Nem minden axióma definíció. Néhány axióma általánosabb információkat fr le egyes predikátumokról, anélkül hogy létrehozna egy definíciót. Valójában egyes predikátumoknak nincs teljes definíciójuk, mert nem tudunk eleget róluk ahhoz, hogy teljes mértékben meghatározzuk őket. Például nem létezik egyértelmű befejezése a következő mondatnak:

$$\forall x \text{ Személy}(x) \Leftrightarrow \dots$$

Szerencsére az elsőrendű logika lehetővé teszi számunkra, hogy használjuk a *Személy* predikátumot, anélkül hogy teljes mértékben definiálnánk. Ehelyett részleges pontosításokat írhatunk tulajdonságokról, amelyekkel minden személy rendelkezik, és olyan tulajdonságokról, amelyek valamit személylé tesznek:

$$\forall x \text{ Személy}(x) \Rightarrow \dots$$

$$\forall x \dots \Rightarrow \text{Személy}(x)$$

Az axiómák lehetnek „csak pusztta tények”, mint amilyenek a *Férfi(Jim)* és a *Házastársa(Jim, Laura)*. Az ilyen tények alkotják a speciális probléma példányok leírásait, lehetővé téve speciális kérdések megválaszolását. Ilyenkor az ezekre a kérdésekre adott válaszok tételek lesznek, amelyek az axiómákból következnek. Gyakran előfordul, hogy az elvárt válasz nem kapható meg – például a *Férfi(George)* és a *Házastársa(George, Laura)*-ból azt várnánk, hogy kikövetkezhetőnek kellene lennie annak, hogy *Nő(Laura)*; de ez nem következik a korábban megadott axiómákból. Ez annak a jele, hogy egy axióma hiányzik. A 8.8. feladat ennek a megadását kéri.

## Számok, halmazok és listák

A számok talán a leglátványosabb példái annak, hogy hogyan építhetünk fel egy hatalmas elméletet axiómák kicsi darabjait használva. Le fogjuk írni a **természetes számok (natural numbers)**, vagyis a nem negatív egész számok elméletét. Szükségünk van egy predikátumra *TermSzám*, amely igaz lesz minden természetes számra; szükségünk van továbbá egy konstansszimbólumra, ez lesz a 0; valamint egy függvényszimbólumra, az *S* rekurziót leíró függvényre. A **Peano-axiómák (Peano axioms)** meghatározzák a természetes számokat és az összeadást.<sup>8</sup> A természetes számoknak egy rekurzív definícióját adjuk meg:

$$\text{TermSzám}(0)$$

$$\forall n \text{ TermSzám}(n) \Rightarrow \text{TermSzám}(S(n))$$

Ez azt jelenti, hogy a 0 egy természetes szám, és hogy minden *n* objektumra, ha az *n* egy természetes szám, akkor *S(n)* is egy természetes szám. Tehát, a természetes számok a 0, az *S(0)* és az *S(S(0))* és így tovább. Szükségünk van axiómáakra ahhoz is, hogy lezárjuk a rekurziót leíró függvényt:

<sup>8</sup> A Peano-axiómák még tartalmazzák az indukció elvét is, amely inkább a másodrendű logikába tarozó állítás, mintsem az elsőrendű logikába. Ennek a megkülönböztetésnek a leírását lásd a 9. fejezetben.

$$\forall n \ 0 \neq S(n)$$

$$\forall m, n \ m \neq n \Rightarrow S(m) \neq S(n)$$

Most meg tudjuk határozni az összeadást az utódfüggvény szempontjából:

$$\forall m \ TermSzám(m) \Rightarrow + (0, m) = m$$

$$\forall m, n \ TermSzám(m) \wedge TermSzám(n) \Rightarrow + (S(m), n) = S(+ (m, n))$$

Az első axióma azt mondja ki, hogy ha hozzáadunk egy 0-t bármely  $m$  természetes számhoz, akkor  $m$ -et kapunk. Figyeljük meg a „+” bináris függvényszimbólum használatát a  $+ (m, 0)$  termben. A matematika jelölése szerint a termet a következőképpen írnánk:  $m + 0$ , infix (infix) jelölés használatával. (Az eddigiekben az elsőrendű logikához használt jelöléseket prefixnek (prefix) nevezzük.) Azért, hogy könnyebben olvashatóvá tegyük a számokról szóló mondatainkat, lehetővé tesszük az infix jelölések használatát. Így az  $S(n)$ -t leírhatjuk úgy is, hogy  $n + 1$ , ami által a második axióma a következő lesz:

$$\forall m, n \ TermSzám(m) \wedge TermSzám(n) \Rightarrow (m + 1) + n = (m + n) + 1$$

Ez az axióma az utódfüggvény ismételt alkalmazásává egyszerűsíti az összeadást.

Az infix jelölések használata példa a **szintaktikus édességre** (syntactic sugar), ami nem más, mint az alapszintaxis kiterjesztése vagy rövidítése, a szemantika megváltoztatása nélkül. Bármely mondat, amely ezt a szintaktikai édességet használja visszaalkútható, egyenértékű mondatot előállítva a hagyományos elsőrendű logikában.

Mihelyt van egy összeadás-definícióink, magától értetődik a szorzást ismételt hozzáadásokként definiálni, a hatványozást ismételt szorzásokként, meghatározni az egész számok osztását és maradékait, a prímszámokat és így tovább. Így tehát a teljes számelmélet (beleértve a titkosírást) felépíthető egy konstansból, egy függvényből, egy predikátumból és négy axiómából.

A **halmazok** (sets) tárgyterületének szintén alapvető szerepe van a matematikában, ugyanúgy mint a hétköznapi gondolkodásban is. (Valójában a származékok levezethető a halmazelméletből kiindulva.) Halmazokat szeretnénk reprezentálni, beleérte az üres halmazt is. Szükségünk van a halmazok felépítésének egy módjára, amelyben vagy hozzá tudunk adni egy elemet a halmazhoz, vagy két halmaz metszetét vagy unióját tudjuk képezni. Meg kell tudnunk állapítani, hogy egy elem része-e egy halmaznak, és szeretnénk megkülönböztetni a halmazokat az olyan objektumoktól, amelyek nem halmazok.

A halmazelmélet hagyományos szókincsét fogjuk használni szintaktikai édessékként. Az üres halmaz egy konstans, amelynek a jele:  $\{ \}$ . Definiálunk egy unáris predikátumot, a *Halmaz*-t, amely minden halmaz esetében igaz. A bináris predikátumok az  $x \in s$  (az  $x$  az  $s$  halmaz része) és az  $s_1 \subseteq s_2$  (az  $s_1$  halmaz egy részhalmaza, nem feltétlenül valódi részhalmaza az  $s_2$ -nek). A bináris függvények az  $s_1 \cap s_2$  (két halmaz metszete), az  $s_1 \cup s_2$  (két halmaz uniója) és az  $\{x|s\}$  (a halmaz az  $x$  elem  $s$  halmazhoz való csatlakozásának eredménye). Az axiómák egy lehetséges sorozata a következő:

1. Egy halmaz vagy az üres halmaz, vagy azok, amelyeket egy halmaz bővítésével hoztunk létre:

$$\forall s \ Halmaz(s) \Leftrightarrow (s = \{ \}) \vee (\exists x, s_2 \ Halmaz(s_2) \wedge s = \{x|s_2\})$$

2. Az üres halmaznak nincs hozzáadott eleme, más szóval nincs lehetőség arra, hogy az *ÜresHalmaz*-t szétválasszunk egy kisebb halmazra és egy elemre:

$$\neg \exists x, s \setminus \{x|s\} = \{\}$$

3. Egy halmazbeli elem újból hozzáadása a halmazhoz nem változtat a halmazon.

$$\forall x, s \setminus x \in s \Leftrightarrow s = \{x|s\}$$

4. Egy halmaznak csak az az eleme, amit hozzádunk. Ezt rekurzívan fejezzük ki, kimondva, hogy  $x$  akkor és csak akkor eleme az  $s$  halmaznak, ha  $s$  azonos valamilyen  $s_2$  halmaz  $\setminus$  elemmel való bővítésével, ahol vagy azonos  $y$   $x$ -szel, vagy  $x$  eleme  $s_2$ -nek:

$$\forall x, s \setminus x \in s \Leftrightarrow [\exists y, s_2 \setminus (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$$

5. Egy halmaz egy másiknak a részhalmaza, ha az összes első halmazbeli elem eleme a második halmaznak:

$$\forall s_1, s_2 \setminus s_1 \subseteq s_2 \Leftrightarrow (\forall x \setminus x \in s_1 \Rightarrow x \in s_2)$$

6. Két halmaz akkor és csak akkor egyenlő, ha egymásnak részhalmazai:

$$\forall x, s_1, s_2 \setminus (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

7. Egy objektum akkor és csak akkor eleme két halmaz metszetének, ha eleme minden két halmaznak:

$$\forall x, s_1, s_2 \setminus x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

8. Egy objektum akkor és csak akkor eleme két halmaz uniójának, ha a két halmaz bármelyikének az eleme:

$$\forall x, s_1, s_2 \setminus x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

A **listák** (*lists*) tárgyköre nagyon hasonló a halmazok tárgyköréhez. A különbség az, hogy a listák rendezettek, és ugyanaz az elem több helyen is előfordulhat egy listában. Használhatjuk a Lisp szótárát a listákra: a *Nil* az elem nélküli listát jelölő konstans, a *Cons*, az *Append*, a *First* és a *Rest* függvények. A *Find* egy predikátum, amely azt hajtja végre a listáknál, mint az *Eleme* a halmazoknál. A *List?* egy predikátum, amely csak a listákra igaz. Úgy, mint a halmazoknál, a listáról szóló logikai mondatokban is szokás használni a szintaktikai édességet. Az üres lista a: [ ]. A *Cons(x, y)* termet, ahol az  $y$  nem üres lista, úgy írjuk, hogy:  $[x|y]$ . A *Cons(x, Nil)* termet (egy lista, amely tartalmazza az  $x$  elemet) úgy írjuk, hogy:  $[x]$ . Több elem listája, mint az  $[A, B, C]$ , ugyanazt jelenti, mint az egymásba ágyazott term: *Cons(A, Cons(B, Cons(C, Nil)))*). A 8.14. feladat tárgya az axiómák megírása a listák tárgyköréhez.

## A wumpus világ

A 7. fejezetben megadtunk néhány ítéletlogikai axiómát a wumpus világra. Az ebben az alfejezetben szereplő elsőrendű logika axiómái sokkal tömörebbek lesznek, természetes módon ábrázolva azt, amit pontosan mondani szeretnénk.

Emlékezzünk arra, hogy egy wumpus ágens egy öt elemből álló érzetvektort kap. Az ezt leíró elsőrendű logikai mondat, amelyet a tudásbázisban tárolunk, tartalmazza mind az érzetet, mind az időpontot amikor ez bekövetkezett, különben az ágens összekerverné, hogy mikor láttá azt, amit látott. Az időpontokat jelölő lépésekre egész számokat használunk. Egy tipikus érzetmondat ilyen lesz:

*Érzet([Bűz, Szellő, Ragyogás, Nincs, Nincs], 5)*

Itt az *Érzet* egy bináris predikátum, míg a *Bűz* és a többi érzet listába rendezett konstansok. A wumpus világban lejátszódó cselekvéset a következő logikai termekkel reprezentálhatjuk:

*Fordul(Jobbra), Fordul(Balra), Előre, Lő, Megragad, Eleresz, Mászik*

Hogy meghatározzuk, melyik cselekvés a legjobb az adott helyzetben, az ágens program létrehoz egy lekérdezést, mint például:

*Ǝa LegjobbCselekvés(a, 5)*

A KÉRDEZ-zel végrehajtjuk ezt a lekérdezést, és vissza fogunk kapni egy lekötési listát, mint például: *{alMegragad}*. Az ágensprogram ezután visszaadhatja a *Megragad*-ot, mint elvégzendő cselekvést, de először a KUELEMENT felhasználásával közölnie kell a saját tudásbázisával, hogy egy *Megragad*-ot hajt végre.

A nyers érzetadatok alapján következtethetünk az aktuális állapot bizonyos tényiről. Például:

$\forall t, s, g, m, c \quad \text{Érzet}([s, \text{Szellő}, g, m, c], t) \Rightarrow \text{Szellő}(t)$   
 $\forall t, s, g, m, c \quad \text{Érzet}([s, \text{Ragyogás}, m, c], t) \Rightarrow \text{Ragyogás}(t)$

és így tovább. A szabályok a következtetési folyamat egy triviális formáját mutatják, amit **érzékelésnek (perception)** nevezünk, és amelyet a 24. fejezetben fogunk részletesen tanulmányozni. Vegyük észre az univerzális kvantort az idő (*t*) felett. Az ítélet-logikában minden egyes időlépéshez szükségünk volt a mondatok másolatára.

Az egyszerű „reflex” viselkedés is leírható kvantorokat alkalmazó implikációs mondatokkal. Például lehet egy ilyen mondatunk:

$\forall t \quad \text{Ragyogás}(t) \Rightarrow \text{LegjobbCselekvés}(\text{Megragad}, t)$

Mivel adottak az érzet és az előző bekezdésekben megismert szabályok, ez a mondat a kívánt konklúziót eredményezné: *LegjobbCselekvés(Megragad, 5)* – ami azt jelenti, hogy a *Megragad* a legjobb cselekvés, amit végrehajthatunk. Vegyük észre az egybeesést a 7.20. ábrában az áramkörön alapuló ágensnél e szabály és a közvetlen érzet-cselekvés kapcsolat között. Az áramkör jellegéből következően *impliciten* látja el kvantorral az időt.

Eddig ebben az alfejezetben az idővel operáló mondatok minden szinkrónok (*synchronic*), azaz egyidejűek voltak, ami azt jelenti, hogy a világ állapotának tulajdonságait vonatkoztatják ugyanannak a világállapotnak más tulajdonságaihoz. Azokat a mondatokat, amelyek lehetővé teszik az időbeliséget figyelembe vevő következtetést **diakrónnak (diachronic)** nevezzük. Például az ágensnek tudnia kell, hogy hogyan kombinálja a megelőző helyzetéről lévő információt azzal az információval, amely helyzetének meghatározásához szükséges. A diakrón mondatok tárgyalását elhalasztjuk a 10. fejezetig, most csak tételezzük fel, hogy a helyzetre és más időfüggő predikátumokra vonatkozó következtetések megtörténtek.

Bemutattuk az érzeteket és a cselekvéseket, ezután itt az ideje, hogy bemutassuk magát a környezetet. Kezdjük az objektumokkal. A négyzetek, a csapdák és a wumpus nyilvánvaló szereplők. Elnevezhetnénk minden egyes négyzetet –  $Négyzet_{1,2}$  és így tovább – de akkor az a tény, hogy a  $Négyzet_{1,2}$  szomszédos a  $Négyzet_{1,3}$ -mal, egy extra tényt igényelne, és minden egyes négyzetpárnál szükségünk lenne egy ilyen tényre. Jobb egy komplex termet használni, amelyben a sor és az oszlop egész számokként jelenik meg; például egyszerűen használhatjuk a lista termet: [1, 2]. Bármely két négyzet szomszédossága meghatározható így:

$$\forall x, y, a, b \ Szomszédos([x, y], [a, b]) \Leftrightarrow \\ [a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}$$

Minden egyes csapdát elnevezhetnénk, de ez sem lenne megfelelő, méghozzá más okból: nincs ok arra, hogy megkülönböztetést tegyünk a csapdák között.<sup>9</sup> Sokkal egyszerűbb használni egy *Csapda* unáris predikátumot, amely igaz a csapdát tartalmazó négyzetekre. Végül, mivel pontosan egy wumpus van csak, ezért egy *Wumpus* konstans éppen úgy megfelelő, mintha egy unáris predikátummal jelölnék (és a wumpus szempontjából talán tiszteletre méltóbb is). A wumpus pontosan egy négyzetben lakik, tehát jó gondolat egy függvényt használni ennek a négyzetnek a megnevezésére, ami az *Otthon(Wumpus)*. Ez kiküszöböli azt a számos mondatot, amelyre az ítéletlogikában szükségünk van ahhoz, hogy megmondjuk, pontosan egyetlen négyzet tartalmaz wumpust. (Még nehezebb helyzetet teremtene az ítéletlogika számára, ha két wumpus volna.)

Az ágens helyzete változik az idő haladtával, amit úgy jelölünk, hogy *Akkor*(Ágens, s, t), ami azt jelenti, hogy az ágens az s négyzetről van t időben. Mihelyt megadtuk az aktuális helyzetét, az ágens következtetni tud a négyzet tulajdonságaira a jelenlegi érzete tulajdonsáiból. Például, ha az ágens egy négyzetről van, és szellőt érzékel, akkor az a négyzet szellős:

$$\forall s, t \ Akkor(\text{Ágens}, s, t) \wedge \text{Szellő}(t) \Rightarrow \text{Szellős}(s)$$

Hasznos tudni, hogy egy négyzet szellős, mert tudjuk, hogy a csapdák nem tudnak mozogni. Vegyük észre, hogy a *Szellős*-nek nincsen időargumentuma.

Miután felfedeztük, mely helyek szellősek (vagy büdősek) vagy – és ez nagyon fontos – nem szellősek (vagy nem büdősek), az ágens ki tudja következtetni, hogy hol vannak a csapdák (és hogy hol van a wumpus). Kétfajta szinkrón szabály van, amely lehetővé teszi az ilyen következtetéseket:

- **Diagnosztikus szabályok**

A diagnosztikus szabályok a megfigyelt hatásuktól a rejтett okokhoz vezetnek mincket. A csapdák megtalálásához a legkézenfekvőbb diagnosztikai szabályok azt mondják ki, hogyha a négyzet szellős, akkor valamelyik szomszédos négyzet biztosan tartalmaz csapdát:

$$\forall s \ Szellős(s) \Rightarrow \exists r \ Szomszédos(r, s) \wedge Csapda(r)$$

<sup>9</sup> Hasonlóan, legtöbbünk nem nevezi meg a madarat, amint elszáll fejünk felett, hogy melegebb vidékre költözön télen. Egy ornitológus viszont, aki a vonulási mintákat és a túlélési arányokat szeretné tanulmányozni, megnevez minden egyes madarat azáltal, hogy megyűrűzi őket, mivel a madarakat egyenként kell nyomon követnie.

és hogyha egy négyzet nem szellős, akkor egyetlen szomszédos négyzet sem tartalmaz csapdát:<sup>10</sup>

$$\forall s \ \neg Szellős(s) \Rightarrow \neg \exists r \ Szomszédos(r, s) \wedge Csapda(r)$$

E kettő kombinálásával, ezt a bikondicionális mondatot kapjuk:

$$\forall s \ Szellős(s) \Leftrightarrow \exists r \ Szomszédos(r, s) \wedge Csapda(r) \quad (8.3)$$

- Ok-okozati szabályok**

Az ok-okozati szabályok a világban lévő okozatiság feltételezett irányát tükrözik: a világ néhány rejtett tulajdonsága bizonyos érzetek generálását eredményezi. Például egy csapda azt eredményezi, hogy az összes szomszédos négyzet szellős lesz:

$$\forall r \ Csapda(r) \Rightarrow [\forall s \ Szomszédos(r, s) \Rightarrow Szellős(s)]$$

és ha az adott négyzettel szomszédos négyzetek egyike sem tartalmaz csapdát, a négyzet nem lesz szellős:

$$\forall s \ [\forall r \ Szomszédos(r, s) \Rightarrow \neg Csapda(r)] \Rightarrow \neg Szellős(s)$$

Némi munkával meg lehet mutatni, hogy ezek a mondatok együtt logikusan egyenértékűek a (8.3) egyenletben leírt ekvivalenciamondatokkal. Maga az ekvivalencia is tekinthető ok-okozatnak, mert megmutatja, hogy a *Szellős* igazságérteke a világ állapota alapján határozható meg.

Azokat a rendszereket, amelyek ok-okozati szabályokat alkalmaznak **modellalapú következtető** (*modell-based reasoning*) rendszereknek nevezzük, mivel az ok-okozati szabályok a környezet működésének modelljét alkotják. A modellalapú és a diagnosztikus következtetés közötti megkülönböztetés a mesterséges intelligencia számos területén fontos. Az orvosi diagnosztika egy különösen aktív területe a kutatásnak, amelyben a tünetek és a betegségek közötti közvetlen asszociációkon alapuló (diagnosztikai megközelítések) megközelítéseket fokozatosan felváltották az olyan módszerek, amelyek a betegség kialakulásának és a betegségtünetek manifesztálódásának egy explicit modelljét használják. Ez a téma a 13. fejezetben újra előkerül.

Bármelyik reprezentációs formát is használja az ágens, *ha az axiómák pontosan és teljesen leírják a világ működésének a módját és azt a módot, ahogyan az érzetek létrejönnek, akkor bármely teljes logikai következtetési eljárás az elérhető érzetek megadása után ki fogja következtetni a világ állapotának legvalószínűbb lehetséges leírását*. Így tehát az ágenstervező arra koncentrálhat, hogy a tudást helyesen adja meg, anélkül hogy sokat kellene töprengenie a következtetések folyamatain. Láttuk továbbá azt is, hogy az elsőrendű logika hasonlóan tömören tudja reprezentálni a wumpus világot, mint a 7. fejezetben megadott eredeti magyar nyelvű leírás.

<sup>10</sup> Az emberben van hajlandóság arra, hogy elfelejtse leírni az ehhez hasonló negatív információkat. Egy beszélgetés során ez a tendencia teljesen normális – furcsa lenne, ha azt mondanánk: „Van két csésze az asztalon, és nincs ott három vagy több.” Még akkor is, ha a „Van két csésze az asztalon” szigorúan szólva akkor is igaz, ha három van ott. A 10. fejezetben visszatérünk erre a problémára.

## 8.4. TUDÁSTERVEZÉS AZ ELSŐRENĐÚ LOGIKÁBAN

A megelőző alfejezet bemutatta az elsőrendű logika használatát a tudás reprezentálására három egyszerű tárgyterületben. Ez az alfejezet a tudásbázis felépítésének általános folyamatát írja le – egy folyamatot, amit **tudástervezésnek** (*knowledge engineering*) nevezünk. A tudásmérnök egy olyan személy, aki egy bizonyos tárgyterületet vizsgál, megismeri, hogy mely koncepciók fontosak abban a tárgyterületben, és megalkot egy formális reprezentációt a tárgyterületben található objektumokra és relációkra. A tudástervezés folyamatát az elektronikus áramkör tárgyterületében fogjuk illusztrálni, amely már valószínűleg ismerős, így koncentrálhunk az ezzel járó reprezentációs problémákra. Az általunk alkalmazott megközelítés *speciális célú* tudásbázisok kialakításához megfelelő, amelyek tárgyterülete alaposan körülhatárolt, és amelyek lekérdezéseinek az egész sorozatát előre ismerjük. Általános *célú* tudásbázisokat, amelyeket arra használunk, hogy lekérdezésekkel tegyenek lehetővé az emberi tudás teljes területére vonatkozóan, a 10. fejezetben tárgyaljuk majd meg.

### A tudástervezés folyamata

A tudástervezési projektek különbözők tárgyukat, tárgykörüket és nehézségüket tekintve, de minden ilyen projekt tartalmazza a következő lépéseket:

1. *A feladat beazonosítása.* A tudásmérnöknek fel kell vázolnia a kérdések sorát, amelyekkel a tudásbázis foglalkozni fog, és a tényeknek azokat a csoportjait, amelyek minden egyes problémaspecifikus példányban megtalálhatók lesznek. El kell döntenie például, hogy a wumpus tudásbázisnak képesnek kell-e lennie a cselekvések kiválasztására, vagy hogy csak az várható el, hogy a környezet elemeivel kapcsolatos kérdéseket válaszolja meg. Tudnia kell, hogy az érzékelőktől származó tények leírják-e a jelenlegi helyzetet. A feladat határozza meg, hogy mely tudást kell tárolni, hogy a problémapéldányokban a válaszokat megadhassuk az adott esetre vonatkozóan. Ez a lépés analóg az ágensek tervezésénél látott TKBÉ-folyamattal, amelyről a 2. fejezetben írtunk.
2. *A releváns tudás összegyűjtése.* A tudásmérnök vagy már szakértője a tárgyterületnek, vagy együtt kell működni igazi szakértőkkel, hogy megismerje az ő tudásukat – ezt a folyamatot **tudásmegszerzésnek** (*knowledge acquisition*) nevezzük. Ezen a szinten a tudást formálisan nem reprezentáljuk. A cél az, hogy megértsük a tudásbázis tárgykörét, amit a feladat határol be, és meg kell érteni azt is, hogy a tárgyterület hogyan működik a gyakorlatban.

A wumpus világban, amelyet mesterségesen létrehozott szabályrendszer határoz meg, könnyű az idevonatkozó tudásbázist azonosítani. (Vegyük észre azonban, hogy a szomszédosság definíciója nem volt explicit megadva a wumpus világ szabályai-ban.) A valódi tárgyterületekben a relevancia problémája meglehetősen bonyolult lehet – például egy VLSI tervező szimulációs rendszernek figyelembe kell vennie a szórt kapacitásokat és a felületi hatásokat.

3. *Meg kell határozni a predikátumok, függvények és konstansok szótárát.* Ez azt jelenti, hogy a fontos tárgyterület szintű koncepciókat le kell fordítani logikai szintű

nevekre. Ez számos, a megközelítés jellegét tárgyaló kérdést érint. Hasonlóan, mint a programozási stílusnak, ennek is jelentős hatása lehet a projekt végső sikérére. Például ilyen kérdés, hogy a csapdákat vajon objektumok jelenítések-e meg vagy egy a négyzetekre vonatkozó unáris predikátum? Az ágens irányá függvény legyen-e vagy predikátum? A wumpus helyzete az időtől függjön-e? Mihelyt a választás meg-történt, az eredmény egy szótár, amit a tárgyterület ontológiájának (**ontology**) nevez-zünk. Az **ontológia** fogalma egy olyan elméletet takar, ami a létezés természetét írja le. Meghatározza, hogy milyen dolgok léteznek, de nem határozza meg a rájuk jellem-ző tulajdonságokat, sem a köztük fennálló kapcsolatokat.

4. A tárgyterületről szóló általános tudás kódolása. A tudásmérnök leírja a szótár össz-szes termjéhez tartozó axiómákat. Ez lerögzíti (amennyire lehetséges) a termek je-lentését, és lehetővé teszi a szakértő számára a tartalom ellenőrzését. Ez a lépés gyakran feltárja a félreértelemezéseket vagy a hiányosságokat a szótárban, amelyeket a 3. lépéshöz újra és újra visszatérve, iteratív eljárással javíthatunk.
5. Az adott probléma példány leírásának kódolása. Ha az ontológia jól átgondolt, ez a lépés már könnyű lesz. Egyszerű atomi mondatokat kell az ontológiában már leírt fo-galmak példányaira megfogalmazni. Egy logikai ágens számára a problémápéldányo-kat az érzékelők biztosítják, amikor a „különálló” tudásbázist kiegészítik mondatokkal ugyanúgy, ahogy a hagyományos programoknak bemeneti adatokat adunk meg.
6. Lekérdezéseket fogalmazunk meg a következtetési folyamat számára és válaszokat vezetünk le. Itt kapjuk meg az eddigi munkánk jutalmát: működtethetjük a követke-ztetési folyamatot az axiómákon és a problémáspecifikus tényeken, hogy megkapjuk a minket érdeklő tényeket.
7. Szűrjük ki a hibákat a tudásbázisból. Sajnos az első próbálkozásra a kérdésekre kapott válaszok nagyon ritkán lesznek helyesek. Pontosabban, a válaszok helyesek lesznek a megadott tudásbázis szempontjából, feltételezve, hogy a következtetési folyamat meg-felelő, de a válaszok nem azok lesznek, amiket a felhasználó vár. Például ha hiányzik egy axióma, akkor bizonyos kérdések megválaszolhatatlannak lesznek a tudásbázis alapján. Ilyenkor egy hibajavítási folyamatra van szükség. A hiányzó vagy túl gyenge axiómák könnyen megtalálhatók úgy, hogy felfedezzük azokat a helyeket, ahol a kö-vetkeztetés láncá várhatlanul megszakad. Például ha a tudásbázis tartalmazza az egyik, csapdákra vonatkozó diagnosztikus axiómát,

$$\forall s \text{ } Szellős(s) \Rightarrow \exists r \text{ } Szomszédos(r, s) \wedge Csapda(r)$$

de nem tartalmazza a másikat, akkor az ágens soha nem lesz képes bizonyítani a csapdák hiányát. A helytelen axiómák azonosíthatók, mivel ezek hamis állítások a világról. Például az a mondat, hogy:

$$\forall x \text{ } LábakSzáma}(x, 4) \Rightarrow Emlős(x)$$

hamis a hüllőkre, kétéltűekre vagy ami még fontosabb, az asztalokra nézve. Ennek a mondatnak a hamissága a tudásbázis többi részétől függetlenül meghatározható. Ezzel szemben egy tipikus programhiba ilyen:

$$\text{eltolás} = \text{pozíció} + 1$$

Nem lehet ez alapján megmondani, hogy a mondat helyes-e anélkül, hogy megnéz-nénk a program többi részét. Például azt, hogy az eltolás-t a jelenlegi pozícióra



vonatkozóan használjuk-e, vagy arra, amely egygel a jelenlegi pozíció mögött van, vagy arra, hogy a pozíció értéke megváltozott egy másik állítás által, és így az eltolás-t is meg kell változtatni.

Hogy jobban megértsük ezt a hétlépéses folyamatot, alkalmazzuk most egy kiterjesztett példára – az elektronikus áramkörök tárgyterületére.

## Az elektronikus áramkörök tárgyterülete<sup>11</sup>

Létrehozunk egy ontológiát és egy tudásbázist, amelynek felhasználásával képesek leszünk következtetéseket végezni olyan típusú digitális áramkörökről, mint amilyet a 8.4. ábra mutat. A tudástervezés hétlépéses folyamatát fogjuk követni.

### A feladat meghatározása

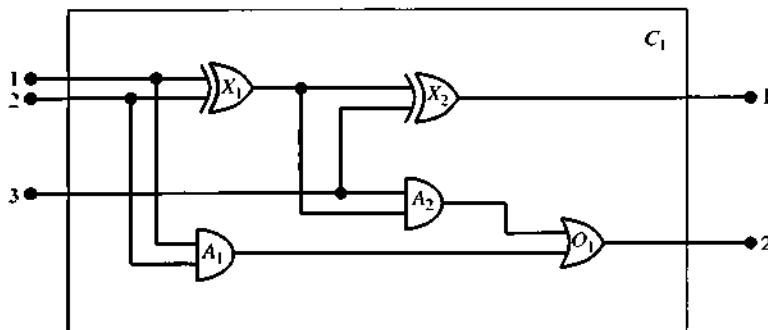
Számos következtetési feladatot lehet a digitális áramkörökkel kapcsolatban elvégezni. A legmagasabb szinten az áramkör funkcionalitását vizsgálhatjuk. Például szabályosan ad-e össze a 8.4. ábrán látható áramkör? Ha minden bemenet magas, akkor mi az  $A_2$  kapu kimenetének állapota? Az áramkör szerkezetéről feltett kérdések is érdekesek. Például melyek azok a kapuk, amelyek az első bemeneti ponthoz vannak kapcsolva? Tartalmaz-e az áramkör visszacsatolásokat? Ebben az alfejezetben ezeket a feladatokat vizsgáljuk meg. Léteznek az elemzésnek részletesebb szintjei is, például amelyek az időzítés késleltetésével, az áramkör területével, áramfogyasztásával, működtetési költségeivel és így tovább kapcsolatosak. minden ilyen szint vizsgálata további ismereteket igényel.

### A releváns tudás összegyűjtése

Mit tudunk a digitális áramkörökről? A céljainknak megfelelően azt, hogy vezetékek-ből és kapukból állnak. A jelek a vezetékeken keresztül áramlanak a kapuk bemeneti termináljához, és minden kapu egy jelet hoz létre a kimeneti terminálon, ami aztán egy másik vezetéken áramlik. Ahhoz, hogy meghatározzuk, melyek is ezek a jelek, ismernünk kell, hogy a kapuk hogyan alakítják át bemeneti jeleket. Négyféle kaput használunk: az AND, az OR és a XOR kapuknak két bemenetük van, míg a NOT kapuknak csak egy. minden kapunak egy kimenete van. Az áramköröknek, hasonlóan a kapukhoz, bemeneteik és kimeneteik is vannak.

Ahhoz, hogy következtetéseket végezhessünk a funkcionalitásról és az összeköttetések ről, nem szükséges magukról a vezetékekről vagy ezek útvonalairól, két vezeték

<sup>11</sup> Ez az alfejezet a digitális áramkörök meglehetősen sajátos bemutatására vállalkozik. Miközben nem javasoljuk, hogy valaki ez alapján ismerkedjen meg a digitális áramkörökkel, és esetleg az itt bemutatottak alapján próbáljon digitális áramköröt tervezni, az itt leírtak mindenképpen érdekes nézőpontot tükröznek, és érdekes kísérletet jelentenek arra, hogy az elsőrendű logika fogalomkészletével mutassák be a digitális áramköröket. (A szerk.)



**8.4. ábra.** Egy  $C_1$ -es digitális áramkör, amelynek az a célja, hogy egy egy bites teljes összeadást végezzen. Az első két bemenet az a két bit, amit össze kell adni, míg a harmadik bemenet az átvitel. Az első kimenet az összeg, míg a második kimenet az átvitel a következő összeadó felé. Az áramkör két XOR, két AND és egy OR kaput tartalmaz.

találkozásánál levő kereszteződéseiről tudást megfogalmazni. Csak a be- és kimenetek közötti összeköttetések számítanak – tehát csak azt kell kimondani, hogy egy kimenet össze van-e kapcsolva egy másik bemenettel, anélkül hogy meg kellene említeni a vezetékeket, amik valójában összekötik őket. A tárgyterületnek sok más tényezője van, ami a mi vizsgálatunkban nem releváns. Ilyen például a különböző elemek mérete, formája, színe vagy ára.

Ha a célunk valami más lenne, és nem a kapuszintű tervezés helyességének ellenőrzése, akkor az ontológiánk is más lenne. Például ha az érdekelne minket, hogyan lehetne a hibákat a hibás áramkörökben megtalálni, akkor valószínűleg jó ötlet lenne a vezetékekkel is foglalkozni az ontológiában, mivel egy hibás vezeték meghamisíthatja a rajta keresztülhaladó jelet. Az időzítési hibák megtalálásához a kapuk késleltetésével kapcsolatos fogalmakat kellene leírni. Ha az érdekelne minket, hogy hogyan lehet egy nyereséges terméket tervezni, akkor az áramkörök költségének és sebességének a piacon jelen lévő egyéb termékekkel történő összehasonlíthatósága lenne fontos.

## A szótár meghatározása

Tudjuk, hogy áramkörök ról, be- és kimenetekről, jelek ról és kapukról akarunk beszélni. A következő lépés az ezeket reprezentáló függvények, predikátumok és konstansok ki-választása. Az egyes kaputípusuktól fogunk indulni, és végül eljutunk az áramkörökig.

Először is meg kell tudnunk különböztetni egy kaput a többi kaputól. Ezt úgy érjük el, hogy konstansokat használunk a kapuk megnevezésére:  $X_1$ ,  $X_2$  és így tovább. Habár minden kapu a maga egyedi módján kapcsolódik az áramkörhöz, a viselkedése – vagyis az a mód, ahogyan átalakítja a bemeneti jeleket kimeneti jeleké – csak a típusától függ. Egy kapu típusának jelölésére<sup>12</sup> egy függvényt használhatunk. Például írhatjuk:

<sup>12</sup> Vegyük észre, hogy megfelelő betűkkel kezdődő neveket használtunk –  $A_1$ ,  $X_1$  és így tovább – pusztán azért, hogy könnyebben olvashatóvá tegyük a példát. A tudásbázisnak még így is tartalmaznia kell a kapuk típusára vonatkozó információkat.

hogy  $\text{Tipus}(X_1) = \text{XOR}$ . Ez hozzárendeli az  $\text{XOR}$  konstanst egy bizonyos kaputípushoz. A többi konstans nevei:  $\text{OR}$ ,  $\text{AND}$  és  $\text{NOT}$  lesznek. A  $\text{Tipus}$  függvény nem az egyetlen lehetséges módja annak, hogy kódoljuk az ontológiai megkülönböztetést. Használhatunk volna egy bináris predikátumot is, mint a  $\text{Tipus}(X_1, \text{XOR})$ , vagy több egyargumentumú predikátumot, mint például az  $\text{XOR}(X_1)$ . Ezeknek a megoldásoknak bármelyike jól működne, de a  $\text{Tipus}$  függvény választásával kiküszöböltük, hogy szükség legyen egy olyan axiómára, amely azt mondja ki, hogy minden egyes kapunak csak egyetlen típusa lehet.

Ezután megvizsgáljuk a végpontokat (be- és kimeneteket). Egy kapunak vagy áramkörnek egy vagy több bemenete, és egy vagy több kimenete lehet. Mindegyiket elnevezhetünk egyszerűen egy konstanssal, mint ahogyan azt a kapukkal tettük. Így az  $X_1$  kapunak olyan végpontjai lennének, mint az  $X_1Be_1$ , az  $X_1Be_2$  és az  $X_1Ki_1$ . A hosszú, összetett elnevezéseket azonban célszerű kerülni. Az, hogy valamit  $X_1Be_1$ -nek nevezünk, nem jelenti azt, hogy ez az  $X_1$  első bemenete; még ekkor is hozzá kell tennünk valamit egy explicit állítást használva. Valószínűleg szerencsésebb egy függvénnyel leírni egy kaput, hasonlóan mint, ahogy János király bal lábát elnevezük *BalLáb(János)*-nak. Így tehát jelöljük az  $X_1$  kapu első bemenetét úgy, hogy:  $Be(1, X_1)$ . Egy hasonló  $Ki$  függvényt használunk a kimenetekre.

A kapuk közötti összeköttetést reprezentálhatjuk az *Összekapcsolt* predikátummal, ami két végpontot vesz argumentumként, például így:  $\text{Összekapcsolt}(Ki(1, X_1), Be(1, X_2))$ .

Végül, ismernünk kell, hogy egy jel magas vagy alacsony állapotban van-e. Erre egy lehetőség egy *On* bináris predikátum használata, és akkor igaz, ha a jel egy végponton magas értékű. Ez azonban egy kissé megnehezíti az olyan kérdések feltevését, mint például: „Mik a lehetséges értékei a  $C_1$  áramkör kimenetein lévő jeleknek?” Ezért be fogunk vezetni objektumokként két „jelértéket”, az 1-et és a 0-t, valamint egy *Jel* függvényt, aminek egy végpont az argumentuma, és ami kijelöli ennek a végpontnak a jelértékét.

## A tárgyterülettel kapcsolatos általános tudás kódolása

Az egyik jele annak, hogy megfelelő ontológiát használunk az, hogy kevés olyan általános szabály van, amit később a példányokra specifikussá kellene tennünk. A helyes szótár jellemzője az, hogy minden egyes szabályt világosan és tömören meg tudunk fogalmazni. A mi példánkban csak hétféle egyszerű szabályra van szükségünk, hogy leírunk minden, amit tudunk kell az áramkörök ról.

1. Ha két végpont össze van kapcsolva, akkor ugyanaz lesz a jelértékük:

$$\forall t_1, t_2 \text{ } \text{Összekapcsolt}(t_1, t_2) \Rightarrow \text{Jel}(t_1) = \text{Jel}(t_2)$$

2. A jel minden végpontnál vagy 1, vagy 0 (de soha nem mindkettő):

$$\begin{aligned} \forall t_1 \text{ } \text{Jel}(t) = 1 \vee \text{Jel}(t) = 0 \\ 1 \neq 0 \end{aligned}$$

3. Az Összekapcsolt egy felcserélhető (kommutatív) predikátum:

$$\forall t_1, t_2 \text{ } \text{Összekapcsolt}(t_1, t_2) \Leftrightarrow \text{Összekapcsolt}(t_1, t_2)$$

4. Egy OR kapu kimenete akkor és csak akkor 1, ha bármelyik bemenete 1:

$$\begin{aligned} \forall g \quad & \text{Típus}(g) = OR \Rightarrow \\ & Jel(Ki(1, g)) = 1 \Leftrightarrow \exists n \quad Jel(Be(n, g)) = 1 \end{aligned}$$

5. Egy AND kapu kimenete akkor és csak akkor 0, ha bármelyik bemenete 0:

$$\begin{aligned} \forall g \quad & \text{Típus}(g) = AND \Rightarrow \\ & Jel(Ki(1, g)) = 0 \Leftrightarrow \exists n \quad Jel(Be(n, g)) = 0 \end{aligned}$$

6. Egy XOR kapu kimenete akkor és csak akkor 1, ha a bemenetei különbözök:

$$\begin{aligned} \forall g \quad & \text{Típus}(g) = XOR \Rightarrow \\ & Jel(Ki(1, g)) = 1 \Leftrightarrow Jel(Be(1, g)) \neq Jel(Be(2, g)) \end{aligned}$$

7. Egy NOT kapu kimenete különbözik a bemenetétől:

$$\forall g \quad (\text{Típus}(g) = NOT \Rightarrow Jel(Ki(1, g)) \neq Jel(Be(1, g)))$$

## A problémaspecifikus példányok kódolása

A 8.4. ábrán bemutatott áramkör neve  $C_1$  és az itt következő leírással adjuk meg. Először kategorizáljuk a kapukat:

$$\begin{array}{ll} \text{Típus}(X_1) = XOR & \text{Típus}(X_2) = XOR \\ \text{Típus}(A_1) = AND & \text{Típus}(A_2) = AND \\ \text{Típus}(O_1) = OR & \end{array}$$

Ezután leírjuk a köztük fennálló kapcsolatokat:

$$\begin{array}{ll} \text{Összekapcsolt}(Ki(1, X_1), Be(1, X_2)) & \text{Összekapcsolt}(Be(1, C_1), Be(1, X_1)) \\ \text{Összekapcsolt}(Ki(1, X_1), Be(2, A_2)) & \text{Összekapcsolt}(Be(1, C_1), Be(1, A_1)) \\ \text{Összekapcsolt}(Ki(1, A_2), Be(1, O_1)) & \text{Összekapcsolt}(Be(2, C_1), Be(2, X_1)) \\ \text{Összekapcsolt}(Ki(1, A_1), Be(2, O_1)) & \text{Összekapcsolt}(Be(2, C_1), Be(2, A_1)) \\ \text{Összekapcsolt}(Ki(1, X_2), Ki(1, C_1)) & \text{Összekapcsolt}(Be(3, C_1), Be(2, X_2)) \\ \text{Összekapcsolt}(Ki(1, O_1), Ki(2, C_1)) & \text{Összekapcsolt}(Be(3, C_1), Be(1, A_2)) \end{array}$$

## Lekérdezések megfogalmazása a következtetési eljárás felé

Milyen bemeneti kombinációk esetében lenne a  $C_1$  első kimenete (az összegelem) 0 és a  $C_1$  második kimenete (a maradék elem) 1?

$$\begin{aligned} \exists i_1, i_2, i_3 \quad & Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2 \wedge Jel(Be(3, C_1)) = i_3 \\ & \wedge Jel(Ki(1, C_1)) = 0 \wedge Jel(Ki(2, C_1)) = 1 \end{aligned}$$

A válaszok az  $i_1$ ,  $i_2$  és  $i_3$  változók behelyettesítései úgy, hogy a keletkezett mondat következzen a tudásbázisból. Hárrom ilyen behelyettesítés létezik:

$$\{i_1/1, i_2/1, i_3/0\} \quad \{i_1/1, i_2/0, i_3/1\} \quad \{i_1/0, i_2/1, i_3/1\}$$

Melyek a az összeadó áramkör összes végpontjának lehetséges értékhalmazai?

$$\begin{aligned} \exists i_1, i_2, i_3, o_1, o_2 \quad & Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2 \\ & \wedge Jel(Be(3, C_1)) = i_3 \wedge Jel(Ki(1, C_1)) = o_1 \wedge Jel(Ki(2, C_1)) = o_2 \end{aligned}$$

Ez az utolsó lekérdezés egy teljes bemenet-kimenet táblázatot ad meg az eszközre, amelyet aztán ellenőrizhetünk, hogy valóban helyesen adja-e össze a bemeneteket. Ez egy egyszerű példa az **áramkör ellenőrzésére** (*circuit verification*). Az áramkör bemutatott definícióját nagyobb digitális rendszerek építésére is felhasználhatjuk, amelyekre aztán ugyanez a fajta ellenőrzési folyamat alkalmazható (lásd 8.17. feladat). Sok tárgyterület kezelhető egy ehhez hasonló strukturált tudásbázis-fejlesztéssel, amelyekben összetettebb koncepciókat határozunk meg egyszerűbb koncepciókra építve.

## Hibák kiszűrése a tudásbázisból

Sokféleképpen perturbálhatjuk a tudásbázist, hogy meglássuk, milyen fajta hibás viselkedések fordulhatnak elő. Például tételezzük fel, hogy kihagyjuk az  $1 \neq 0^{13}$  állítást. A rendszer ebben a pillanatban már nem lesz képes semmilyen kimenetet sem produkálni az áramkörben, kivéve, ha a bemenet a 000 vagy az 110. Rábukkanhatunk a problémára, ha minden egyes kapu kimenetére ráraknánk. Például, megkérdezhetjük:

$$\exists i_1, i_2, o \quad Jel(Be(1, C_1)) = i_1 \wedge Jel(Be(2, C_1)) = i_2 \wedge Jel(Ki(1, X_1))$$

ami felfedi, hogy az  $X_1$ -nél nem ismerjük a kimeneteit az 10 és a 01 bemenetek esetén. Ezután megnézzük az XOR kapu axiómát az  $X_1$ -re alkalmazva:

$$Jel(Ki(1, X_1)) = 1 \Leftrightarrow Jel(Be(1, X_1)) \neq Jel(Be(2, X_1))$$

Ha tudjuk, hogy a bemenetek, mondjuk, 1 és 0 voltak, akkor ezt lerövidíthetjük így:

$$Jel(Ki(1, X_1)) = 1 \Leftrightarrow 1 \neq 0$$

Most már látható, hogy mi a probléma oka: a rendszer nem képes kikövetkeztetni azt, hogy  $Jel(Ki(1, X_1)) = 1$ , így meg kell neki mondaniuk, hogy  $1 \neq 0$ .

## 8.5. ÖSSZEFOGLALÁS

Ebben a fejezetben megmutattuk, hogy hogyan használható az **elsőrendű logika** (*first-order logic*) egy tudásalapú ágens reprezentációs nyelveként. A legfontosabb megállapítások a következők:

- A tudásreprezentációs nyelveknek deklaratívnak, kompozíciósnak, kifejezőnek, a szövegkörnyezettől függetlennek és egyértelműnek kell lenniük.

<sup>13</sup> Ez a fajta kihagyás eléggy gyakori, mert az emberek általában feltételezik, hogy a különböző nevek különböző dolgokat takarnak. A logikai programozási rendszerek, amelyeket többében a 9. fejezetben mutatunk be, szintén megteszik ezt a feltételezést.

- A logikák különböznek **ontológiai** és **episztemológiai megállapításaikban**. Amíg az ítéletlogika megállapításai csak tények létezésére vonatkoznak, addig az elsőrendű logika az objektumok és a relációk létezését is felhasználhatja, így nagyobb kifejezőerővel rendelkezik.
- Egy **lehetséges világot** vagy **modellt** az elsőrendű logikában objektumok egy hal-maza definiál, a köztük lévő relációk és a rájuk alkalmazható függvények által.
- A **konstansszimbólumok** (*constant symbols*) objektumokat neveznek meg, a **predikátumszimbólumok** (*predicate symbols*) relációkat, míg a **függényszimbólumok** (*function symbols*) függvényeket. Egy interpretáció megadja a leképezést a szimbólumok és a modell között. Az **összetett termek** (*complex terms*) függvény-szimbólumokat rendelnek hozzá a termekhez, hogy nevet adjanak egy objektumnak. Ha megadunk egy interpretációt és egy modellt, a mondat igazságtartalmát meghatároztuk.
- Egy **atomi mondat** (*atomic sentence*) egy vagy több termre alkalmazott prediká-tumból áll. A mondat csak akkor igaz, ha a predikátum által megnevezett reláció fennáll a termek által megnevezett objektumok között. Az **összetett mondatok** (*complex sentences*) – az ítéletlogikához hasonló módon – összekötőjeleket használnak, a **kvantorok** (*quantifiers*) használata pedig lehetővé teszi általános szabályok megfogalmazását is.
- Egy tudásbázis építése az elsőrendű logikában a tárgyterület alapos elemzését igényli, valamint egy szótár megválasztását és a kívánt következetisések eléréséhez szükséges axiómák kódolását.

## Irodalmi és történeti megjegyzések

Habár már Arisztotelész logikája képes volt objektumok feletti általánosítások kezelésére, az igazi elsőrendű logika keletkezését Gottlob Frege *Begriffschrift* (Fogalmak írása vagy Fogalmi jelölés; Frege, 1879) c. művének megjelenésétől, a kvantorok bevezetésétől számíthatjuk. A Frege-féle képesség a kvantorok egymásba ágyazása jelentős előrelépést jelentett, de az általa alkalmazott jelölésrendszer igen körülmenyes volt. Az elsőrendű logika jelenlegi jelölésrendszere alapvetően Giuseppe Peanótól származik (Peano, 1889), a szemantika azonban megegyezik a Frege által bemutatottal. Elégé különös, hogy Peano axiómái nagyrészt Grassmann-nak (Grassmann, 1861) és Dedekindnek (Dedekind, 1888) voltak köszönhetők.

Az elsőrendű logika fejlődésének fontos akadálya volt az egy változós predikátumok előtérbe helyezése, illetve a több változós predikátumok kizárasa. Az egy változós predikátumokhoz való ragaszkodás általánosan jellemző volt a logikai rendszerekre Arisztotelészről Boole-ig. A logikai relációk első rendszerezett leírását Augustus De Morgan adta (De Morgan, 1864). De Morgan a következő példával mutatta be az arisztotelészi logikával nem kezelhető következetéseket: „Minden ló állat; ezért a ló feje egy állat feje”. Ez a következetés nem valósítható meg az arisztotelészi rendszerrel, mert bármely, a következetésben alkalmazható szabálynak először az „*x* a feje *y*-nak” két predikátumot tartalmazó mondatot kell elemeznie. A relációk logikáját alaposan tanulmányozta Charles Sanders Peirce (Peirce, 1870), aki Frege-től függetlenül, néhány évvel később szintén kifejlesztette az elsőrendű logikát (Peirce, 1883).

Leopold Löwenheim adta meg a modellelmélet rendszerezett leírását az elsőrendű logika számára (Löwenheim, 1915). Cikke az egyenlőségszimbólumot már a logika szerves részének tekinti. Löwenheim eredményeit Thoralf Skolem fejlesztette tovább (Skolem, 1920). Alfred Tarski a halmazelméletet felhasználva megadta az igazság és a modellelméleti kielégíthetőség explicit definícióját (Tarski, 1935, 1956).

Elsőszorban McCarthy érdeme az elsőrendű logika alkalmazása az MI-rendszerek eszközeként (McCarthy, 1958). A logikán alapuló MI-rendszerek fejlődésében jelentős előrelépést jelentett Robinson rezolúciós algoritmusá (Robinson, 1965), ami egy komplex elsőrendű logikai következtetési folyamat. A rezolúciót a 9. fejezetben fogjuk tárgyalni. A logikai megközelítés alapjaival a Stanfordon sok eredményt értek el. Cordell Green kifejlesztett egy elsőrendű logikai következtetési rendszert, a QA3-at (Green, 1969a, 1969b), amely ahhoz vezetett, hogy először kísérletek meg létrehozni egy logikai robotot a Stanford Research Institute-ban (Fikes és Nilsson, 1971). Az elsőrendű logikát Zohar Manna és Richard Waldinger alkalmazta a programokban való következtetésre (Manna és Waldinger, 1971), majd később Michael Genesereth az áramkörökhez (Genesereth, 1984). Európában a logikai programozást – az elsőrendű következtetés korlátozott változatát – fejlesztették ki a nyelvészeti elemzésekhez (Colmerauer és társai, 1973) és általános deklaratív rendszerekhez (Kowalski, 1974). A számítógépes logikával Edinburgh-ban sikeresen foglalkoztak, az LCF (Számítási Funkciók Logikája) projekten keresztül (Gordon és társai, 1979). Ezeket a fejlesztéseket a 9. és 10. fejezetben követjük nyomon.

Számos színvonalas, az elsőrendű logika korszerű bevezetését adó írás ismert. Ezek közül Quine műve az egyik legjobban áttekinthető (Quine, 1982). Enderton írása inkább matematikai megközelítést követ (Enderton, 1972). Bell és Machover erősen formális leírást készített az elsőrendű logikáról, és ezzel együtt számos bonyolult, a logika tárgyterületébe tartozó témáról (Bell és Machover, 1977). Manna és Waldinger a számítástechnika oldaláról közelítő, jól olvasható bevezetést készített (Manna és Waldinger, 1985). Gallier az elsőrendű logika rendkívül precíz matematikai bemutatását adta meg, együtt a logika automatikus következtetésre történő alkalmazásának részletes tárgyalásával (Gallier, 1986). A *Logical Foundations of Artificial Intelligence* (A mesterséges intelligencia logikai alapjai) (Genesereth és Nilsson, 1987) c. könyv a logika alapos bemutatásán kívül az érzeteket és cselekvéseket kezelő logikai ágensek első rendszerezett bevezetését nyújtja.

## Feladatok

- 8.1.** Egy logikai tudásbázis a világot mondatokkal reprezentálja határozott struktúra nélkül. Egy analóg (*analog*) reprezentáció viszont strukturált, ahol a leírás struktúrája közvetlenül megfelel a reprezentált doleg struktúrájának. Tekintsük egy ország autótérképét mint az országról ismert tények egy részének analóg reprezentációját. A térkép kétdimenziós felépítése megfelel a terület kétdimenziós felszínének.
- Adjон öt példát a térképnyelv szimbólumaira.
  - Explicit mondatnak nevezzük az olyan mondatot, amelyet a reprezentáció létrehozója közvetlenül leír. Az implicit mondatok az explicit mondatokból

keletkeznek az analóg reprezentáció tulajdonságai szerint. Adjon három példát a térképnyelv *implicit* és *explicit* monodataira.

- (c) Adjon az ország fizikai struktúráját leíró tényekre három olyan példát, amely nem reprezentálható a térképnyelvvel.
- (d) Adjon két példát olyan tényekre, amelyek egyszerűbben kifejezhetők a térképnyelvvel, mint az elsőrendű logikával.
- (d) Adjon még két példát analóg reprezentációkra. Melyek az előnyei és a hátrányai ezeknek a nyelveknek?

**8.2.** Tekintsünk egy tudásbázist, amely csak két kijelentést tartalmaz:  $P(a)$  és  $P(b)$ . Ebből a tudásbázisból következik-e az  $\forall x P(x)$ ? Magyarázza meg válaszát modellek segítségével.

**8.3.** Érvényes-e a következő mondat:  $\exists x, y \ x = y$ ? Magyarázza meg.

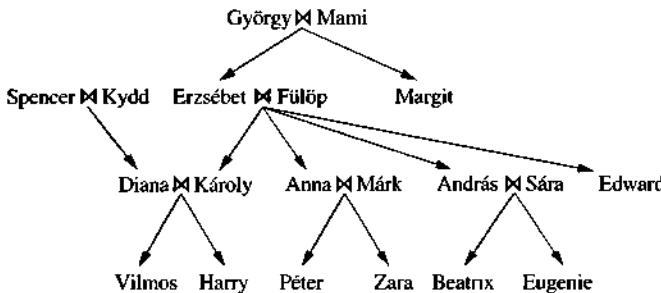
**8.4.** Írjon le egy olyan logikai mondatot, hogy minden olyan világban, ahol ez a mondat igaz, pontosan csak egy objektum legyen.

**8.5.** Tekintsünk egy szimbólumszótárat, amely tartalmaz egy  $c$  konstansszimbólumot,  $p_k$  predikátumszimbólumokat minden egyes  $k$  értékre és függvényszimbólumokat minden egyes  $k$  értékhez, ahol  $1 \leq k \leq A$ . A tárgyterület méretét rögzítük  $D$ -ben. Bármely adott interpretációmodell-kombinációban minden egyes predikátum vagy függvény hozzá van rendelve az ugyanazon értékhez tartozó relációhoz vagy függvényhez. Feltételezzük, hogy a modellben szereplő függvények lehetővé teszik, hogy néhány bemenetnek ne legyen értéke a függvény számára (vagyis, hogy az érték a láthatatlan objektum). Vezessen le egy formulát a lehetséges interpretációmodell-kombinációk számának megállapítására a  $D$  elemet tartalmazó tárgyterületben. Ne riassza meg, ha el kell távolítania a felesleges kombinációkat.

**8.6.** Reprezentálja a következő mondatokat az elsőrendű logikában úgy, hogy egy következetes szótárt használ (amelyet előzőleg definiálnia kell):

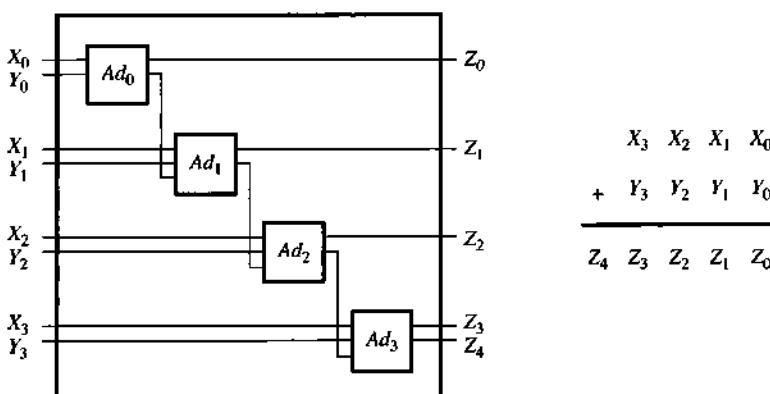
- (a) Néhány diák felvette a franciát 2001 tavaszi félévében.
- (b) minden diák, aki felveszi a franciát, átmegy a vizsgán.
- (c) Csak egy diák vette fel a görögöt 2001 tavaszi félévében.
- (d) A görögben elért legmagasabb pontszám mindig magasabb a franciaiban elérte legmagasabbnál.
- (e) minden személy okos, aki biztosítást köt.
- (f) senki nem köt drága biztosítást.
- (g) Van egy ügynök, aki csak azokkal köt biztosítást, akiknek még nincs kötvényük.
- (h) Van egy borbély, aki minden férfit megborotvál a városban, aki nem borotválkozik.
- (i) Az a személy, aki az Egyesült Királyságban születik, és akinek mindenki szülője brit állampolgár vagy ottani lakos, születésénél fogva brit állam-polgár.

- (j) Az a személy, aki az Egyesült Királyságon kívül születik, és egyik szülöje született brit állampolgár, az származása alapján brit állampolgár.
- (k) A politikusok bármikor bolonddá tehetnek néhány embert, és minden ember bolonddá tehetnek egy kis időre, de nem tehetnek bolonddá mindenkit örökre.
- 8.7.** Reprezentálja a „Minden német azonos nyelvet beszél” mondatot predikátum-kalkulusban. Használja a *Beszél(x, I)* predikátumot, amely jelentse azt, hogy *x* az *I* nyelvet beszéli.
- 8.8.** Milyen axiómára van szükségünk, hogy ezt a tényt kikövetkeztessük: *Nő(Laura)*, ha adottak a tények: *Férfi(Jim)* és *Házastársak(Jim, Laura)*?
- 8.9.** Írjon egy általános tény- és axiómahalmazt, hogy reprezentálja a következő állítást: „Wellington hallott Napóleon haláláról”; és hogy helyesen megválaszolhassuk a kérdést: „Hallott Napóleon Wellington haláláról?”.
- 8.10.** Írja át a 7.5. alfejezetben bemutatott ítéletlogikai wumpus világ tényeit elsőrendű logikába. Mennyivel tömörebb ez a változat?
- 8.11.** Készítsen axiómákat, amelyek leírják a következő predikátumokat: *Unokája, Dédnagyapja, Fivére, Húga, Lánya, Fia, Nagynénje, Nagybátyja, Sőgornője, Sógora, Unokatestvére*.
- Írja le a 8.5. ábrán látható családfa tényeit. Használjon logikai következtető rendszert, és minden leírt mondat legyen a következtető rendszer számára KUELENT-ve, és KÉRDEZ-ze meg, hogy ki Erzsébet unokája, ki Diana unokabátyja és kik Zara dédszülei.
- 8.12.** Írjon le egy mondatot, amely azt állítja, hogy a + egy kommutatív funkció. Következik ez a mondat Peano axiómáiból? Ha igen, magyarázza meg, hogy miért, ha nem, adjon meg egy modellt, amelyben az axiómák igazak, az Ön mondata pedig hamis.
- 8.13.** Magyarázza meg, mi a hibás az  $\in$  halmaz tagság predikátum következő javasolt definíciójával:
- $$\forall x, s \quad x \in \{x|s\}$$
- $$\forall x, s \quad x \in s \Rightarrow \forall y \quad x \in \{y|s\}$$
- 8.14.** A halmazaxiomákat példaként használva, készítsen axiómákat a listák tárgyterületére, amelyek tartalmazzák a korábban a fejezetben említett összes konstanst, függvényt és predikátumot.
- 8.15.** Magyarázza meg, hogy mi a hiba a következő ajánlott definícióban a szomszédos négyzetekre a wumpus világban:
- $$\forall x, s \quad \text{Szomszédos}([x, y], [x + 1, y]) \wedge \text{Szomszédos}([x, y], [x, y + 1])$$



8.5. ábra. Egy tipikus családfa. Az „=” szimbólum a házastársakat köti össze, a nyílak a gyerekekre mutatnak.

- 8.16. Írja meg a wumpus helyzetének meghatározásához szükséges axiómákat a *Wumpus konstansszimbólum* és a *Be(Wumpus, Helyzet)* bináris predikátum felhasználásával. Ne felejtse el, hogy csak egy wumpus van.
- 8.17. Bővíts ki a 8.4. alfejezetben leírt szótárt úgy, hogy definiálja az összeadást az ~~Wumpus~~ *n*-bites bináris számokra. Ezután kódolja a 8.6. ábrán látható négybites összeadó leírását, és tegye fel azokat a kérdéseket, amelyek szükségesek ahhoz, hogy igazoljuk összeadó-helyességét.
- 8.18. A fejezetben az áramkör reprezentálása részletesebb a szükségesnél, ha csak az áramkör működése érdekel minket. Egy egyszerűbb formula bármely *m* bemenetű, *n* kimenetű kaput vagy áramkört leír, egy *m + n* argumentumos predikátum használatával, úgy, hogy a predikátum pontosan akkor igaz, amikor a bemenet és a kimenet konzisztens. Például, a NOT-kapukat így írjuk le: *NOT(i, o)*, amely-nél a *NOT(0, 1)*-et és a *NOT(1, 0)*-át ismerjük. A kapuk összeállítása a kapupredikátumok összekötésével van definiálva, amelyben a közös változók di-



8.6. ábra. Egy négybites összeadó

rekt összeköttetéseket jeleznek. Például egy NAND áramkört összeállíthatunk AND-ekből és NOT-okból:

$$\forall i_1, i_2, o_a, o \quad NAND(i_1, i_2, o) \Leftarrow AND(i_1, i_2, o) \wedge NOT(o_a, o)$$

Ezt a reprezentációt felhasználva definiálja a 8.4. ábrán látható egy bites összeadót és a 8.6. ábra négy bites összeadóját, és magyarázza meg, milyen lekérdezéseket használna, hogy igazolja a megoldásokat. Milyen fajta lekérdezéseket *nem* fogad el ez a reprezentáció, amelyeket pedig a 8.4. alfejezetben lévő reprezentáció elfogadott?

- 8.19.** Kérjen útlevéligenyelő lapot az Ön saját országába, nevezze meg azokat a szabályokat, amelyek az igénylés jogosultságát meghatározzák, és fordítsa le az elsőrendű logika nyelvére a 8.4. alfejezetben felvázolt lépéseket követve.

# 9. KÖVETKEZTETÉS ELSŐRENDŰ LOGIKÁBAN

Ebben a fejezetben definiálunk egy olyan hatékony következtetési mechanizmust, amelyik képes megválaszolni az elsőrendű logikában feltett kérdéseket.

A 7. fejezetben meghatároztuk a következtetés (*inference*) fogalmát, és bemutattuk, hogy hogyan érhető el helyes és teljes következtetés az ítéletlogikában. Ebben a fejezetben kiterjesztjük ezeket az eredményeket, hogy olyan algoritmusokat kapjunk, amelyek bármely, az elsőrendű logikában feltehető kérdésre válaszolni tudnak. Ez jelentős eredmény, mivel az elsőrendű logikával többé-kevésbé minden kifejezhetünk, ha elég alaposan végezzük a feladatot.

A 9.1. alfejezet következtetési szabályokat vezet be a kvantorokra, és megmutatja, hogy hogyan lehet – jólléhet nagy erőfeszítések árán – az elsőrendű logikai következtetést ítéletlogikai következtetésre redukálni. A 9.2. alfejezet leírja az egyesítés (*unification*) ötletét, bemutatva, hogy az egyesítés felhasználásával, hogyan alkothatunk következtetési szabályokat, amelyek közvetlenül az elsőrendű logikai mondatokra alkalmazhatók. Ezután a 9.3. alfejezetben megvizsgáljuk az elsőrendű logikai algoritmusok három nagy családját: az előrefelé láncolást (*forward chaining*) és ennek alkalmazását a deduktív adatbázisokban (*deductive databases*) és a produkciós rendszerekben (*production systems*). A hátrafelé láncolást (*backward chaining*) és a logikai programozást (*logic programming*) alkalmazó rendszereket a 9.4. alfejezetben tárgyaljuk; míg a rezolúció alapú tételbizonyító (*theorem-proving*) rendszereket a 9.5. alfejezet ismerteti. Általánosságban minden a leghatékonyabb, az adott feladatban reprezentáláンド tényeket és axiomákat kezelní képes módszert próbáljuk használni. A teljesen általános, a tetszőleges elsőrendű logikai mondatokon alkalmazható rezolúciós következtetés általában kevésbé hatékony, mint a bizonyos típusú mondatokon alkalmazható előre- vagy hátrafelé láncolást használó megoldások.

## 9.1. ÍTELETLOGIKAI KÖVETKEZTETÉS KONTRA ELSŐRENDŰ LOGIKAI KÖVETKEZTETÉS

Ez és a következő alfejezet bemutatja azokat a gondolatokat, amelyeken a modern logikai következtetési rendszerek alapulnak. Néhány egyszerű következtetési szabállyal kezdjük, amelyeket kvantorral ellátott mondatokhoz alkalmazhatunk, hogy a segítségiükkel kvantorok nélküli mondatokhoz juthassunk. Ezek a szabályok természetes módon elvezetnek minket ahhoz a gondolathoz, hogy az elsőrendű következtetés megvalósítható azáltal, hogy a tudásbázist ítéletlogikává alakítjuk át, és a már általunk ismert ítéletlogikai következtetést használjuk. A következő alfejezet rámutat egy kézenfekvő

egyszerűsítésre, amellyel olyan következtetési módszerekhez juthatunk, amelyek közvetlenül képesek az elsőrendű mondatok felhasználására.

## Kvantorokra vonatkozó következtetési szabályok

Kezdjük az univerzális kvantorok vizsgálatával. Tételezzük fel, hogy a tudásbázisunk tartalmazza a standard hagyományos axiómákat, amelyek szerint minden mohó király gonosz:

$$\forall x \text{ Király}(x) \wedge \text{Mohó}(x) \Rightarrow \text{Gonosz}(x)$$

Ebből lehetségesnek tűnik kikövetkeztetni a következő mondatokat:

$$\text{Király}(\text{János}) \wedge \text{Mohó}(\text{János}) \Rightarrow \text{Gonosz}(\text{János})$$

$$\text{Király}(\text{Richárd}) \wedge \text{Mohó}(\text{Richárd}) \Rightarrow \text{Gonosz}(\text{Richárd})$$

$$\text{Király}(\text{Apja(János)}) \wedge \text{Mohó}(\text{Apja(János)}) \Rightarrow \text{Gonosz}(\text{Apja(János)})$$

⋮

**Az univerzális példányosítás (Universal Instantiation, UI)** szabálya kimondja, hogy az adott változó bármely alaptermmel (ground term) (változók nélküli termmel) való helyettesítésével elérhető mondatokat kikövetkeztethetjük.<sup>1</sup> Ahhoz, hogy formálisan le tudjuk írni a következtetési szabályokat, a 8.3. alfejezetben bevezetett **helyettesítés (substitution)** fogalmát használjuk. A HELYETTESÍT( $\theta, \alpha$ ) fogja jelölni annak az eredményét, hogy egy  $\theta$  behelyettesítést alkalmazunk az  $\alpha$  mondathoz. Ekkor a szabály így írható le:

$$\frac{\forall v \alpha}{\text{HELYETTESÍT}(\{v/g\}, \alpha)}$$

bármely  $v$  változóra és  $g$  alaptermre. Például a korábban megadott három mondatot megkaphatjuk a következő helyettesítésekkel  $\{x/\text{János}\}$ ,  $\{x/\text{Richárd}\}$  és  $\{x/\text{Apja(János)}\}$ .

Az ennek megfelelő egzisztenciális példányosítási (Existential Instantiation) szabály az egzisztenciális kvantorra egy kicsivel komplikáltabb. minden  $\alpha$  mondatra,  $v$  változóra és olyan  $k$  konstans szimbólumra, amely sehol másol nem jelenik meg a tudásbázisban:

$$\frac{\exists v \alpha}{\text{HELYETTESÍT}(\{v/k\}, \alpha)}$$

Például abból a mondatból, hogy:

$$\exists x \text{ Korona}(x) \wedge \text{Fején}(x, \text{János})$$

kikövetkeztethetjük ezt a mondatot:

$$\text{Korona}(C_1) \wedge \text{Fején}(C_1, \text{János})$$

<sup>1</sup> Ne keverjük össze ezeket a helyettesítéseket a kiterjesztett interpretációkkal, amelyeket a kvantorok szemantikájának a meghatározására használtunk. A helyettesítés egy változót cserél le egy termre (egy szintaktikai elemre), hogy új mondatot hozzon létre, ezzel szemben egy interpretáció hozzárendel egy változót egy objektumhoz a tárgyterületben.

feltéve, ha a  $C_1$  nem jelenik meg sehol máshol a tudásbázisban. Alapjában véve az egzisztenciális mondat azt mondja ki, hogy van valamely objektum, amely eleget tesz egy feltételnek, és a példányosítás folyamata csak névvel látja el ezt az objektumot. Ez a név viszont már nem tartozhat más objektumhoz. Egy szép példát találunk a matematikában: tételezzük fel, hogy tudjuk, létezik olyan szám, amely egy kicsit kisebb, mint 2,71828, és eleget tesz annak az egyenletnek, hogy:  $d(x^y)/dy = x^y$  az  $x$ -re. Adhatunk nevet ennek a számnak, mint például  $e$ , de hiba lenne olyan nevet adni, amely már egy létező objektumhoz tartozik, mint például a  $\pi$ . A logikában ezt az új nevet **Skolem-konstansnak** (**Skolem constant**) nevezzük. Az egzisztenciális példányosítás egy speciális esete egy általánosabb folyamatnak, amelyet **skolemizációknak** (**skolemization**) nevezünk, és amelyet a 9.5. alfejezetben mutatunk be.

Amellett hogy az egzisztenciális példányosítás bonyolultabb, mint az univerzális példányosítás, kissé különböző szerepet is játszik a következtetésben. Míg az univerzális példányosítást többször is alkalmazhatjuk, hogy sok különböző eredményhez jussunk, az egzisztenciális példányosítást csak egyszer végezhetjük el, majd az egzisztenciális kvantorral ellátott mondatról megszabadulhatunk. Például amint hozzáadtuk a tudásbázishoz a *Meggyiköt(Gyilkos, Áldozat)* mondatot, már nincs szükségünk a  $\exists x$  *Meggyiköt(x, Áldozat)* mondatra. Pontosabban megfogalmazva, az új tudásbázis logikailag nem ekvivalens a régivel, de tekinthetjük úgy, hogy a következtetés **szempontjából ekvivalens** (**inferentially equivalent**), azaz pontosan akkor kielégíthető, amikor az eredeti tudásbázis is kielégíthető.

## Redukálás ítéletlogikára

Ha egyszer vannak szabályaink arra, hogy kvantorral ellátott mondatokból hogyan következtethetünk kvantor nélküli mondatokra, akkor vissza tudjuk vezetni az elsőrendű következtetést az ítéletlogikai következtetésre. Ebben az alfejezetben áttekintjük ennek a módszernek a legfontosabb elemeit; a részletekről a 9.5. alfejezetben fogunk szólni.

Az első gondolat az, hogy éppen úgy, ahogyan egy egzisztenciális kvantorral ellátott mondatot felcserélhetünk a mondat egy példányával, egy univerzális kvantorral ellátott mondatot is felcserélhetünk a mondat összes lehetséges példányosításainak halmazával. Például tételezzük fel, hogy a tudásbázisunk minden mondatot tartalmazza:

$$\begin{aligned} \forall x \text{ Király}(x) \wedge \text{Mohó}(x) &\Rightarrow \text{Gonosz}(x) \\ \text{Király(János)} \\ \text{Mohó(János)} \\ \text{Fivér(Richárd, János)} \end{aligned} \tag{9.1}$$

Ekkor alkalmazzuk az univerzális páldányosítást az első mondatra úgy, hogy az összes lehetséges alapterm-helyettesítést felhasználjuk a tudásbázisszótáróból, ami ebben az esetben az  $\{x/János\}$  és az  $\{x/Richárd\}$ . Így ezt kapjuk:

$$\begin{aligned} \text{Király(János)} \wedge \text{Mohó(János)} &\Rightarrow \text{Gonosz(János)} \\ \text{Király(Richárd)} \wedge \text{Mohó(Richárd)} &\Rightarrow \text{Gonosz(Richárd)} \end{aligned}$$

és megválhatunk az univerzális kvantorral ellátott mondattól. Mármost, a tudásbázist alapvetően ítéletkalkulus-belinek tekinthetjük, ha az alap atomi mondatokat, mint a *Király(János)*, *Mohó(János)* stb., ítéletlogikai szimbólumoknak tekintjük. Így tehát a 7. fejezetben bemutatott bármely teljes ítéletlogikai algoritmust alkalmazhatjuk, hogy olyan következtetéshez juthassunk, mint a *Gonosz(János)*.

Ez a technika az **ítéletlogikai állításokra való visszavezetés (propositionalization)**, amelyet teljesen általánossá tehetünk, mint azt a 9.5. alfejezetben be is fogjuk mutatni. Ez azt jelenti, hogy minden elsőrendű tudásbázis és lekérdezés átalakítható ítéletlogikai mondatokra úgy, hogy a tudásbázis vonzatai nem változnak. Így tehát van egy teljes következtetési folyamatunk vonzatok vezetésére. Vagy talán mégsem? Van egy probléma: amikor a tudásbázis tartalmaz függvényszimbólumot, a lehetséges alaptermek helyettesítéseinek halmaza végtelen! Például ha a tudásbázis megemlíti az *Apja* szimbólumot, akkor végtelen számú egymásba ágyazott termeket hozhatunk létre, mint például az *Apja(Apja(Apja(János)))*. Az ítéletlogikai algoritmusainknak tehát nehézségeik lesznek a végtelen nagyságú mondathalmazokkal.

Szerencsére ismert egy Jacques Herbrandnak köszönhető híres téTEL az előbbi problémára (Herbrand, 1930), ami azt mondja ki, hogy ha egy mondat következik az eredeti, elsőrendű tudásbázisból, akkor létezik olyan bizonyítás, amely csak egy véges méretű részhalmazt használ fel az ítéletkalkulus-belire átalakított tudásbázisból. Mivel bármely ilyen részhalmazban az alaptermeknek van egy maximális beágyazási mélysége, meg lehet találni a részhalmazt úgy, hogy először generáljuk az összes példányt a konstans szimbólumokhoz (*Richárd* és *János*), azután az összes 1-es mélységű termet (*Apja(Richárd)* és *Apja(János)*), majd az összes 2-es mélységű termet, és így tovább mindaddig, amíg képesek leszünk megalkotni a vonzatmondat ítéletlogikai bizonyítását.

Felvázoltuk az elsőrendű következtetés egy megközelítését az ítéletlogikai állításokra való visszavezetés révén, amely **teljes (complete)** – tehát bármely következményként kapott mondatot bizonyítani tudunk. Ez jelentős eredmény, ha figyelembe vesszük, hogy a lehetséges modellek száma végtelen. Másrészt viszont nem tudhatjuk, hogy a mondat *vonzat-e*, amíg a bizonyítás nincs kész. Mi történik, ha a mondat nem vonzata a *TB-nek*? Meg tudjuk-e ezt állapítani? Nos, ahogy ez majd kiderül, az elsőrendű logika esetében erre a kérdésre a válasz nemleges. A bizonyítási eljárás tovább folytatódik, egyre mélyebben beágyazott termeket generálva, és nem tudhatjuk, hogy a folyamat egy végtelen ciklusba kerül, vagy hamarosan megleljük a bizonyítást. Ez nagyon hasonló a Turing gépek leállási problémájához. Alan Turing és Alonzo Church egymástól függetlenül és eltérő módon bizonyították be ennek a kérdésnek az eldönthetetlenségre vonatkozó téTEL (Turing, 1936; Church, 1936). Az elsőrendű logikában a maga után vonzás kérdése **félíg eldönthető (semidecidable)**, ami azt jelenti, hogy létezik olyan algoritmus, amely igent mond minden vonzatmondatra, de nem létezik olyan algoritmus, amely emellett képes nemet mondani a nem levezethető mondatokra.

## 9.2. EGYESÍTÉS ÉS KIEMELÉS

Az előző alfejezet bemutatta az elsőrendű következtetésnek azt az értelmezését, amely az 1960-as évekig létezett. Az éles szemű olvasók (és bizonyára a hatvanas évek elejének logikával foglalkozó kutatói) észrevehettek, hogy az ítéletlogikai megközelítés nem nagyon

hatékony. Például ha megnézzük a *Gonosz(x)* lekérdezést és az ehhez tartozó tudásbázist a (9.1) egyenletben, feleslegesnek tűnik olyan mondatok generálása, mint a *Király(Richárd)  $\wedge$  Mohó(Richárd)  $\Rightarrow$  Gonosz(Richárd)*. Valójában a *Gonosz(János)* mondatra vonatkozó következtetés bárki számára teljesen nyilvánvaló a következő mondatok alapján:

$$\begin{aligned} \forall x \text{ Király}(x) \wedge \text{Mohó}(x) &\Rightarrow \text{Gonosz}(x) \\ \text{Király(János)} \\ \text{Mohó(János)} \end{aligned}$$

Most megmutatjuk, hogyan tehetjük ezt a számítógép számára is egyértelművé.

## Egy elsőrendű következtetési szabály

Az a következtetés, hogy János gonosz a következőképpen megy végbe: találjon egy  $x$ -et, amelyre igaz, hogy  $x$  egy király és  $x$  mohó, majd következtesse ki, hogy  $x$  gonosz. Általánosabban, ha van valamely  $\theta$  helyettesítés, amely az implikáció premisszáját megegyezővé teszi a tudásbázisban már létező mondatokkal, akkor a  $\theta$  helyettesítés elvégzése után az implikációkövetkezmény részét hozzáadhatjuk a tudásbázishoz. Ebben az esetben az  $\{x/János\}$  helyettesítés eléri ezt a célt.

Tulajdonképpen több feladatot is elvégezhetünk a következtetés ezen lépései. Tételezzük fel, hogy a *Mohó(János)* mondat ismerete helyett csak azt tudjuk, hogy mindenki mohó:

$$\forall y \text{ Mohó}(y) \tag{9.2}$$

Ebben az esetben is szeretnénk azt a következtetést levonni, hogy *Gonosz(János)*, mivel tudjuk, hogy János egy király (ez adott), és hogy János mohó (mert mindenki mohó). Egy megfelelő helyettesítés megtalálására van szükségünk ennek elvégzéséhez, mind az implikációs mondatban, mind az ehhez illesztendő mondatokban található változókra. Ebben az esetben az  $\{x/János, y/János\}$  helyettesítés alkalmazása az implikáció *Király(x)* és *Mohó(x)* premisszáihoz és a *Király(János)* és *Mohó(y)* tudásbázis mondatokhoz létrehozza az azonosságot. Így tehát ki tudjuk következtetni az implikáció konklúzióját.

Ez a következtetés elvégezhető egyetlen következtetési szabály alkalmazásával, amelyet általánosított Modus Ponensnek (**Generalized Modus Ponens**) nevezünk: a  $p_i, p'_i$  és  $q$  atomi mondatokra, amelyekre létezik olyan  $\theta$  helyettesítés, hogy  $\text{HELYETTESÍT}(\theta, p'_i) = \text{HELYETTESÍT}(\theta, p_i)$  minden  $i$ -re, akkor:

$$\frac{p'_1, p'_2, \dots, p'_n, (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{HELYETTESÍT}(\theta, q)}$$

A szabály  $n + 1$  premisszát tartalmaz:  $n$  számú  $p'_i$  atomi mondatot és egy implikációt. A konklúzió a  $q$  konzékvenciára történő helyettesítés alkalmazásának az eredménye. A mi példánkra ezt így alkalmazhatjuk:

$$\begin{array}{ll} p'_1 - \text{Király(János)} & p_1 - \text{Király}(x) \\ p'_2 - \text{Mohó}(y) & p_2 - \text{Mohó}(x) \\ \theta - \{x/János, y/János\} & q - \text{Gonosz}(x) \\ \text{HELYETTESÍT}(\theta, q) - \text{Gonosz(János)} & \end{array}$$

Könnyű megmutatni, hogy az általánosított Modus Ponens helyes következtetési szabály. Először is, megfigyelhetjük, hogy bármely  $p$  mondatra (amelyeknek változóiról feltételezzük, hogy univerzális kvantorokkal vannak ellátva) és bármely  $\theta$  helyettesítésre:

$$p \models \text{HELYETTESÍT}(\theta, p)$$

Ez hasonlóan igazolható, mint az univerzális példányosítás szabály, és alkalmazható olyan  $\theta$ -ra, amely kielégíti az általánosított Modus Ponens szabály feltételeit. Így tehát, a  $p'_1, \dots, p'_n$ -ből következtethetjük, hogy:

$$\text{HELYETTESÍT}(\theta, p'_1) \wedge \dots \wedge \text{HELYETTESÍT}(\theta, p'_n)$$

és a  $p_1 \wedge \dots \wedge p_n \Rightarrow q$  implikációból ezt következtethetjük, hogy:

$$\text{HELYETTESÍT}(\theta, p_1) \wedge \dots \wedge \text{HELYETTESÍT}(\theta, p_n) \Rightarrow \text{HELYETTESÍT}(\theta, q)$$

Mivel a  $\theta$  az általánosított Modus Ponensben úgy van definiálva, hogy:  $\text{HELYETTESÍT}(\theta, p'_i) = \text{HELYETTESÍT}(\theta, p_i)$  minden  $i$ -re; ebből kifolyólag a két mondatból az első pontosan illeszkedik a másodiknak a premisszájához. Így tehát a  $\text{HELYETTESÍT}(\theta, q)$  következik a Modus Ponensből.

Az általánosított Modus Ponens egy **kiemelt** (*lifted*) változata a Modus Ponensnek – átemeli a Modus Ponent az ítéletlogikából az elsőrendű logikába. Látni fogjuk a fejezet későbbi részében, hogy kifejleszthetjük az előrefelé láncolás, a hátrafelé láncolás és a 7. fejezetben bemutatott rezolúciós algoritmusok kiemelt változatait is. A kiemelt következtetési szabályok legfontosabb előnye az ítéletlogikára történő átalakítással szemben az, hogy csak azokat a helyettesítéseket hajtják végre, amelyek bizonyos következtetések végrehajtását teszik lehetővé. Egy potenciális korlát az, hogy egy bizonyos szempontból az Általánosított Modus Ponens kevésbé általános, mint a Modus Ponens (lásd 265. oldal): a Modus Ponens bármely  $\alpha$  mondatot megenged az implikáció bal oldalán, ezzel szemben az Általánosított Modus Ponens speciális formátumot kíván meg erre a mondatra. Csak abban az értelemben nevezhető általánosíttnak, hogy tetszőleges számú  $p'_i$ -re alkalmazható.

## Egyesítés

A kiemelt következtetési szabályok olyan helyettesítések megtalálását igénylik, amelyek a különböző logikai kifejezéseket látszólag azonossá teszik. Ezt a folyamatot **egyesítési lépések** (**unification**) nevezik, ami kulcsfontosságú eleme minden elsőrendű következtetési algoritmusnak. Az EGYESÍT algoritmus vesz két mondatot, és visszaad egy rájuk vonatkozó **egyesítést** (**unifier**), ha létezik ilyen:

$$\text{EGYESÍT}(p, q) = \theta, \text{ ahol } \text{HELYETTESÍT}(\theta, p) = \text{HELYETTESÍT}(\theta, q)$$

Nézzünk meg néhány példát arra, hogy az EGYESÍT-nek hogyan kell viselkednie. Tételezzük fel, hogy van egy *Ismer(János, x)* lekérdezésünk: kit ismer János? Erre a kérdésre néhány választ úgy találhatunk, hogy megkeressük az összes mondatot a tudásbázisban, amely egyesíthető az *Ismer(János, x)*-szel. Itt vannak az egyesítési eredmények négy különböző, a tudásbázisban előforduló mondatra.

$\text{EGYESÍT}(\text{Ismer}(János, x), \text{Ismer}(János, Júlia)) = \{x/Júlia\}$

$\text{EGYESÍT}(\text{Ismer}(János, x), \text{Ismer}(y, Lajos)) = \{x/Lajos, y/János\}$

$\text{EGYESÍT}(\text{Ismer}(János, x), \text{Ismer}(y, \text{Anyja}(y))) = \{y/János, x/\text{Anyja}(János)\}$

$\text{EGYESÍT}(\text{Ismer}(János, x), \text{Ismer}(x, \text{Erzsébet})) = \text{sikertelen}$

Az utolsó egyesítés sikertelen, mert az  $x$  nem tudja a János és az Erzsébet értékeit egyszerre felvenni. Emlékezzünk arra, hogy az  $\text{Ismer}(x, \text{Erzsébet})$  azt jelenti, hogy „Mindenki ismeri Erzsébetet”, így képesnek kellene lennünk azt kikövetkeztetni, hogy János ismeri Erzsébetet. A probléma abból származik, hogy a két mondat történetesen ugyanazt a változónévét, az  $x$ -et használja. A problémát úgy kerülhetjük el, hogy az egyesítendő mondatokban átnevezzük a változókat (standardizing apart) úgy, hogy elkerüljük a nevek egybeeséseit. Például átnevezhetjük az  $x$ -et az  $\text{Ismer}(x, \text{Erzsébet})$ -ben például  $z_{17}$ -re (egy tetszőleges új változónévre) anélkül, hogy a jelentését megváltoztatnánk. Most már működik az egyesítésünk:

$\text{EGYESÍT}(\text{Ismer}(János, x), \text{Ismer}(z_{17}, \text{Erzsébet})) = \{x/\text{Erzsébet}, z_{17}/János\}$

A 9.7. feladat tovább vizsgálja az átnevezés szükségességének problémáját.

**function** EGYESÍT( $x, y, \theta$ ) **returns** egy helyettesítést, hogy az  $x$  és az  $y$  megegyező legyen

**inputs:**  $x$ , egy változó, lista vagy egy összetétel  
 $y$ , egy változó, lista vagy egy összetétel  
 $\theta$ , az eddig felépített helyettesítés (választható, az üres kizárva)

```

if  $\theta = \text{kudarc}$  then return kudarc
else if  $x = y$  then return  $\theta$ 
else if  $\text{VÁLTOZÓ?}(x)$  then return EGYESÍT-VÁLT( $x, y, \theta$ )
else if  $\text{VÁLTOZÓ?}(y)$  then return EGYESÍT-VÁLT( $y, x, \theta$ )
else if  $\text{ÖSSZETÉTEL?}(x)$  and  $\text{ÖSSZETÉTEL?}(y)$  then
return EGYESÍT(ARCOK[x], ARCOK[y], EGYESÍT(VÁL[x], VÁL[y],  $\theta$ ))
else if  $\text{LISTA?}(x)$  and  $\text{LISTA?}(y)$  then
return EGYESÍT(MARADÉK[x], MARADÉK[y], EGYESÍT(Első[x], Első[y],  $\theta$ ))
else return kudarc

```

**function** EGYESÍT-VÁLT( $vált, x, \theta$ ) **returns** egy helyettesítés

**inputs:**  $vált$ , egy változó  
 $x$ , bármely kifejezés  
 $\theta$ , az eddig felépített helyettesítések

```

if  $\{vált/ért\} \in \theta$  then return EGYESÍT( $ért, x, \theta$ )
else if  $\{x/ért\} \in \theta$  then return EGYESÍT( $ért, vál, \theta$ )
else if  $\text{ELŐFORDUL-PRÓBA?}\{vált, x\} \theta$ -re then return kudarc
else return hozzáad( $\{vált/x\} \theta$ -hoz)

```

**9.1. ábra.** Az egyesíti algoritmus. Az algoritmus összehasonlítja a bemenetek felépítését elemről elemre. A  $\theta$  helyettesítés, amely az EGYESÍT argumentuma, útközben épül fel, és arra használjuk, hogy meggyőződjünk arról, hogy a későbbi összehasonlítások konzisztensek lesznek az általunk előzőleg létrehozott lekötésekkel. Egy összetett kifejezésben, mint például az  $F(A, B)$ , a VÁL függvény kiveszi az  $F$  függvény-szimbólumot, és az ARCOK függvény kiveszi az  $(A, B)$  argumentumlistát.

Még egy további nehézségen kell túljutni. Azt mondjuk, hogy az EGYESÍT olyan helyettesítéseket ad vissza, amelyek a két argumentumot látszólag azonossá teszi. Azonban több különböző ilyen egyesítés létezhet. Például az EGYESÍT(*Ismer(János, x)*, *Ismer(y, z)*) visszaadhatja azt is, hogy: {*y/János, x/z*}, vagy azt is, hogy: {*y/János, x/János, z/János*}. Az első egyesítési lépés azt eredményezi, hogy: *Ismer(János, z)*, ezzel szemben a második azt adja, hogy: *Ismer(János, János)*. A második eredményt megkaphatjuk az elsőből is egy további helyettesítés hozzáadásával: {*z/János*}. Láthatjuk, hogy az első egyesítés általánosabb, mint a második, mert kevesebb korlátozást ad meg a változók értékeire. Azt állapíthatjuk meg, hogy minden egyesítendő kifejezéspárra létezik egy legáltalánosabb egyesítés (**most general unifier**), amely egyedi a változók átnevezésében. Ebben az esetben ez: {*y/János, x/z*}.

A legáltalánosabb egyesítések kiszámításához a 9.1. ábrán láthatunk egy algoritmust. Az eljárás igen egyszerű: egy rekurzív algoritmussal egymással párhuzamosan tárjuk fel a két kifejezést, felépítve az egyesítést, kivéve akkor, ha a struktúrákban a két megfelelő elem nem illeszkedik. Van egy drága lépés: amikor egy változót kell egy komplex termhez illesztünk, meg kell vizsgálni, hogy a változó előfordul-e a termben; ha igen, akkor az illesztés sikertelen. mert nem tudunk konzisztens egyesítést megalkotni. Ez a lépés az úgynevezett előfordulási próba (*occur check*), amely az algoritmus komplexitását az egyesítendő kifejezés méretével négyzetesen növekvővé teszi. Néhány rendszer, beleértve az összes logikai programozási rendszert, egyszerűen kihagyja az előfordulási próbát, és ennek eredményeként a következtetésük nem helyes. Más rendszerek időben lineáris komplexitású, ennél összetettebb algoritmusokat használnak.

## Tárolás és visszakeresés

A KUELENT és a KÉRDEZ függvények, amelyek szerepe, hogy informálják és lekérdezések a tudásbázist, két egyszerűbb függvényt, a TÁROL és a BETÖLT függvényeket használják fel. A TÁROL(*s*) eltárol egy *s* mondatot a tudásbázisban, míg a BETÖLT(*q*) visszaadja az összes egyesítést, amelyet a *q* lekérdezés felhasználásával lehet létesíteni a tudásbázis illeszthető mondataival. A probléma, amivel illusztráltuk az egyesítést – az összes olyan tény megtalálása, amely egyesíthető az *Ismer(János, x)*-szel – a BETÖLT egyik példája.

A TÁROL és a BETÖLT megvalósításának legegyszerűbb módja, hogy a tudásbázisban az összes tényt egyetlen hosszú listán tároljuk, és egy *q* lekérdezés megválaszolásakor meghívjuk az EGYESÍT(*q, s*)-t minden *s* mondatra a listán. Ez az eljárás nem hatékony, de működik, és egyelőre elég a fejezet további részének megértéséhez. Az alfejezet hátralévő része áttekinti azokat az eljárásokat, amelyekkel a visszakeresést hatékonyabbá tehetjük, így ezt első olvasásra átugorhatjuk.

A BETÖLT függvényt azzal tehetjük hatékonyabbá, hogy biztosítjuk, hogy az egyesítési lépést csak olyan mondatokkal kíséreljük meg, amelyeknél van egyáltalán esély az egyesítésre. Például nincs értelme, hogy megpróbáljuk az egyesítést az *Ismer(János, x)*-re és a *Fivér(Richárd, János)*-re. Elkerülhetjük az ilyen egyesítési lépéseket, ha indexeljük (**indexing**) a tényeket a tudásbázisban. Egy egyszerű séma, amit **predikátum indexelésnek** (**predicate indexing**) nevezünk, betesz a összes *Ismer* tényt egy verembe és az

összes *Fivér* tényt egy másikba. A vermek egy hash-táblában<sup>2</sup> tárolhatók, hogy hatékonyan elérhessük őket.

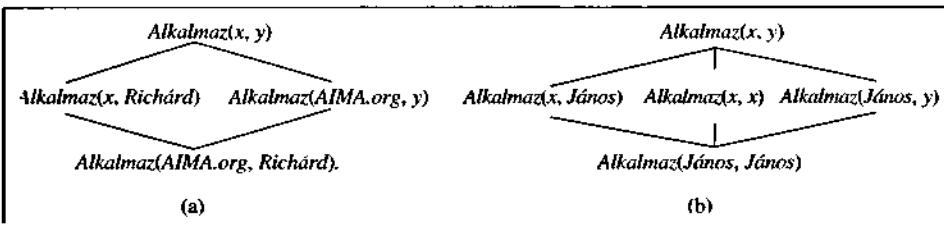
A predikátumok indexelése hasznos, ha sok predikátumszimbólum van, de csak kevés állítás az egyes szimbólumokhoz. Néhány alkalmazásban azonban sok mondat tartozik egy adott predikátumszimbólumhoz. Például tételezzük fel, hogy az adóhatóságok nyomon akarják követni, hogy ki alkalmaz kit, és ezeket a kapcsolatokat egy *Alkalmaz(x, y)* predikátum használatával írjuk le. Ez egy igen nagy verem lenne, akár munkaadók millióival és alkalmazottak tízmillióival. Egy olyan kérdés megválaszolása, mint az *Alkalmaz(x, Richárd)*, a predikátumok indexelésével az egész verem átböngészését igényelné.

Egy ilyen lekérdezés esetében segítenie, ha a tényeket mind a predikátummal, mind a második argumentummal indexelnénk, például úgy, hogy egy kombinált hash-tábla kulcsot használunk. A kulcsot egyszerűen előállíthatjuk a lekérdezésből, és pontosan visszakereshetjük közvetlenül azokat a tényeket, amelyek egyesíthetők a lekérdezéssel. Más lekérdezésekhez, mint amilyen az *Alkalmaz(AlMA.org, y)*, szükségünk van arra, hogy a tényeket a predikátum és az első argumentum kombinálásával indexeljük. Így aztán a tényeket tárolhatjuk többszörös indexkulcsok alatt, azonnal hozzáférhetővé téve minden, a tényel egyesíthető lekérdezés számára.

Ha adott egy tárolandó mondat, akkor létre lehet hozni az indexeket az összes lehetséges vele egyesíthető lekérdezéshez. Arra a tényre, hogy *Alkalmaz(AlMA.org, Richárd)*, ezek a lekérdezések:

<i>Alkalmaz(AlMA.org, Richárd)</i>	Az AlMA.org alkalmazza Richárdot?
<i>Alkalmaz(x, Richárd)</i>	Ki alkalmazza Richárdot?
<i>Alkalmaz(AlMA.org, y)</i>	Kit alkalmaz az AlMA.org?
<i>Alkalmaz(x, y)</i>	Ki alkalmaz kit?

Ezek a lekérdezések egy **bennfoglalási hálót** (subsumption lattice) alkotnak, amint az a 9.2. (a) ábrán látható. A hálónak van néhány érdekes tulajdonsága. Például a háló bármely csomópontjának a gyereke a szülőjéből egy egyszerű behelyettesítéssel megkapható, és a „legmagasabb” közös leszármazottja bármely két csomópontnak a legáltalánosabb egyesítőjük használatával érhető el. A háló részeiből bármely alaptény szisztematikusan létre-



9.2. ábra. (a) A bennfoglalási háló, amelynek a legalacsonyabb csomópontja ez a mondat: *Alkalmaz(AlMA.org, Richárd)*. (b) A bennfoglalási háló arra a mondatra, hogy: *Alkalmaz(János, János)*.

<sup>2</sup> A hash-tábla információk tárolására és visszakeresésére szolgáló olyan adatstruktúra, amelyeket rögzített kulcsokkal indexelünk. Gyakorlati okokból egy hash-táblát tekinthetünk olyannak, mint amelynek a tárolási és a visszakerdezési ideje állandó, akkor is, ha a tábla nagyon számú elemet tartalmaz.

hozható (9.5. feladat). Egy ismétlődő konstansokkal rendelkező mondatnak egy kissé más hálója van, ezt a 9.2. (b) ábrán láthatjuk. A függvényszimbólumok és a változók az eltárolandó mondatokban még érdekesebb hálófelépítéseket tesznek szükségessé.

A séma, amelyet leírtunk, nagyon jól működik, amikor a háló csak kisszámú csomópontot tartalmaz. Egy  $n$  argumentumú predikátumra a háló  $O(2^n)$  csomópontot tartalmaz. Ha a függvényszimbólumok használata megengedett, akkor a csomópontok száma szintén exponenciális a tárolandó mondat termjeinek függvényében. Ez rendkívül nagy számú indexet eredményezhet. Egy ponton az indexelés elveszti előnyeit az összes index tárolásának és fenntartásának problémája miatt. Ez ellen egy rögzített stratégia bevezetésével próbálhatunk védekezni úgy, hogy csak azokon a kulcsokon tartunk fenn indexeket, amelyek predikátumokból és az egyes argumentumokból állnak. Egy másik megoldás, hogy egy adaptív tervet használunk, amely minden olyan indexeket hoz létre, amelyek megfelelnek az aktuális lekérdezéstípus igényeinek. A legtöbb MI-rendszerben a tárolni kívánt tények száma elég kicsi ahhoz, hogy a hatékony indexelés problémáját megoldottnak tekintsük. Az ipari és kereskedelmi adatbázisok esetében jelentős technikai fejlesztések történtek a feladat megoldására.

### 9.3. ELŐREFELÉ LÁNCOLÁS

Az ítéletlogikai határozott klózokra már megadtunk egy előrefelé láncolási algoritmust a 7.5. alfejezetben. A gondolat egyszerű: kezdjük a tudásbázisban szereplő atomi mondatokkal, és alkalmazzuk a Modus Ponens előrefelé haladva, új atomi mondatokat hozzáadva, egészen addig, amíg további következtetések már nem végezhetők. Most mutatjuk, hogyan alkalmazzuk az algoritmust az elsőrendű határozott klózokra, és hogyan valósíthatjuk ezt meg hatékonyan. A határozott klózok, mint például a *Helyzet  $\Rightarrow$  Válasz* különösen hasznosak az olyan rendszerek számára, amelyek újonnan érkezett információk alapján végeznek következtetéseket válaszként. Számos rendszert tervezhetünk ilyen módon, és ezekben az esetekben az előrefelé láncolással történő következtetés sokkal hatékonyabb lehet, mint a rezolúciós tételelbizonyítás. Ebből az következik, hogy gyakran érdemes megpróbálni olyan tudásbázist építeni, amely csak határozott klózokat használ, és így elkerülhetjük a rezolúcióval járó nehézségeket.

### Elsőrendű határozott klózok

Az elsőrendű határozott klózok nagyon emlékeztetnek az ítéletlogikai határozott klózokra (lásd 272. oldal). Az ilyen klózok olyan literálok diszjunkciói, amelyek közül pontosan egy diszjunkt pozitív. Egy határozott klóz vagy egy atomi mondat, vagy egy implikáció, amelynek a feltétel része pozitív literálok konjunkciója, és amelynek következménye egyetlen pozitív literál. A következő mondatok elsőrendű határozott klózok:

$$\text{Király}(x) \wedge \text{Mohó}(x) \Rightarrow \text{Gonosz}(x)$$

$$\text{Király}(\text{János})$$

$$\text{Mohó}(y)$$

Eltérően az ítéletlogikai literáloktól, az elsőrendű literálok tartalmazhatnak változókat, amely esetben a változókat univerzális kvantorral ellátottak tételezzük fel. (A határozott klózok írásánál általában elhagyjuk az univerzális kvantorokat.) A határozott klóz egy megfelelő normál forma, hogy az általánosított Modus Ponenssel alkalmazhassuk. Nem minden tudásbázist lehet átalakítani határozott klózok halmazává, az egyetlen pozitív literális korlátja miatt, de sokat igen. Gondoljuk át a következő problémát:

A törvény kimondja, hogy bűntény az, ha egy amerikai polgár fegyvert ad el egy Amerikával ellen-séges nemzetnek. A Nono ország, amely ellensége Amerikának, fel van szerelve rakétákkal, és ezeket a rakétákat mind West ezredes adta el, aki amerikai.

Be fogjuk bizonyítani, hogy West bűnöző. Először leírjuk a tényeinket elsőrendű határozott klózokként. A következő alfejezet fogja bemutatni, hogy az előrefelé láncolás algoritmus hogyan oldja meg a problémát.

„...bűntény az, ha egy amerikai polgár fegyvert ad el egy Amerikával ellenséges nemzetnek”:

$$\text{Amerikai}(x) \wedge \text{Fegyver}(y) \wedge \text{Ellenséges}(z) \wedge \text{Elad}(x, z, y) \Rightarrow \text{Bűnöző}(x) \quad (9.3)$$

„Nono... fel van szerelve rakétákkal.” Azt a mondatot, hogy  $\exists x \text{ Birtokol}(\text{Nono}, x) \wedge \text{Rakéta}(x)$  átalakítjuk két határozott klózzá Egzisztenciális Eliminációval, egy új konstans, az  $M_1$  bevezetésével:

$$\text{Birtokol}(\text{Nono}, M_1) \quad (9.4)$$

$$\text{Rakéta}(M_1) \quad (9.5)$$

„És ezeket a rakétákat mind West ezredes adta el”:

$$\text{Rakéta}(x) \wedge \text{Birtokol}(\text{Nono}, x) \Rightarrow \text{Elad}(\text{West}, x, \text{Nono}) \quad (9.6)$$

Tudnunk kell még, hogy a rakéták fegyvereik:

$$\text{Rakéta}(x) \Rightarrow \text{Fegyver}(x) \quad (9.7)$$

És hogy Amerika ellensége „ellenségesnek” számít:

$$\text{Ellensége}(x, \text{Amerika}) \Rightarrow \text{Ellenséges}(x) \quad (9.8)$$

„West ezredes..., aki amerikai”:

$$\text{Amerikai}(\text{West}) \quad (9.9)$$

„Nono ország, amely ellensége Amerikának...”

$$\text{Ellensége}(\text{Nono}, \text{Amerika}) \quad (9.10)$$

A tudásbázis nem tartalmaz függvényszimbólumokat, és így egy példája a Datalog (Datalog) tudásbázisok osztályának – ami függvényszimbólumok nélküli elsőrendű határozott klózok halmaza. Látni fogjuk, hogy a függvényszimbólumok hiánya sokkal könnyebbé teszi a következtetést.

## Egy egyszerű előrefelé láncolási algoritmus

Az első előrefelé láncolási algoritmus, amit megvizsgálunk, nagyon egyszerű lesz, amint azt láthatjuk a 9.3. ábrán. Az algoritmus az ismert tényekből kiindulva végrehajtja az összes olyan szabályt, amelynek premisszái ki vannak elégítve, és ezek következményeit hozzáadja az ismert tényekhez. Ez az eljárás addig ismétlődik, amíg a kérdést meg nem választottuk (feltételezve, hogy csak egy válaszra van szükségünk), vagy amikor már nem tudunk új tényeket a tudásbázishoz hozzáadni. Vegyük észre, hogy egy tény nem számít „újnak”, ha csak átnevezése (rename) egy már ismert ténynek. Egy mondat átnevezése egy másiknak, ha az alkalmazott változók neveitől eltekintve megegyeznek. Például a *Szereti(x, Fagylalt)* és a *Szereti(y, Fagylalt)* mondatok egymás átnevezésének számítanak, mert csak az *x* és az *y* választásában különböznek. A két mondat jelentése megegyezik: mindenki szereti a fagylaltot.

Az előző bűntényproblémákat fogjuk felhasználni az ERL-EL-KÉRDEZ algoritmus bemutatásához. Az implikációs mondatok a (9.3), (9.6), (9.7) és a (9.8). Két iterációs lépés szükséges:

- Az első iterációban a (9.3) szabály nem elégítette ki a premisszákat.  
A (9.6) szabályt kielégíti az  $\{x/M_1\}$  és az *Elad(West, M<sub>1</sub>, Nono)* hozzáadása.  
A (9.7) szabályt kielégíti az  $\{x/M_1\}$  és a *Fegyver(M<sub>1</sub>)* hozzáadása.  
A (9.8) szabályt kielégíti az  $\{x/Nono\}$  és az *Ellenséges(Nono)* hozzáadása.
- A második iterációban a (9.3) szabályt kielégíti az  $\{x/West, y/M_1, z/Nono\}$ , és hozzáadjuk azt, hogy: *Bűnöző(West)*.

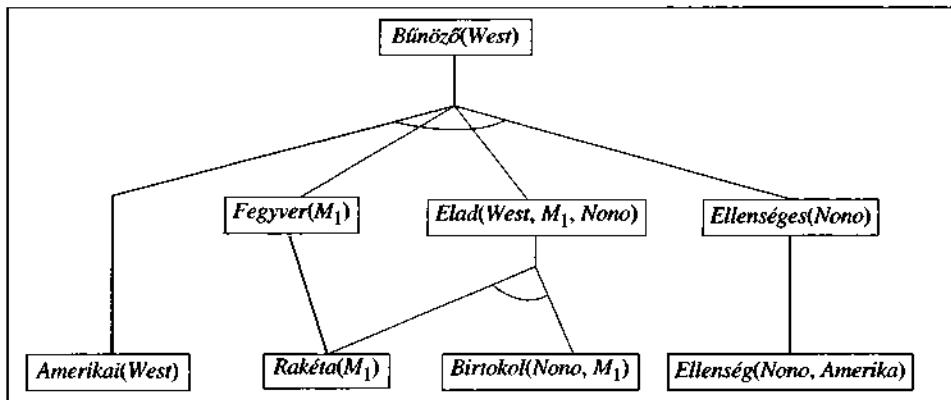
```

function ERL-EL-KÉRDEZ(TB, α) returns egy helyettesítés, vagy hamis
  inputs: TB, a tudásbázis, elsőrendű határozott klózok halmaza
          α, a lekérdezés, egy atomi mondat
  local variables: új, a minden egyes iterációk során kikövetkeztetett új mondatok

  repeat until az új üres
    new  $\leftarrow \{\}$ 
    for each r mondatra in TB do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{VÁLTOZÓ-ÁTNEVEZ}(r)$ 
      for each θ olyan mint HELYETTESÍT(θ, p1 ∨ … ∨ pn) = HELYETTESÍT(θ, p'_1 ∨ … ∨ p'_n)
          néhány p'_1, …, p'_n-re a TB-ben
        q' ← HELYETTESÍT(θ, q)
        if q' nem egy már a TB-ben meglévő mondat átnevezése vagy új then do
          hozzáad q' az új-hoz
          φ ← EGYESÍT(q', α)
          if φ nem sikertelen then return φ
    hozzáad új-at a TB-hez
  return hamis

```

**9.3. ábra.** Egy koncepcionálisan egyszerű, de kevésbé hatékony előrefelé láncolási algoritmus. minden egyes iterációban hozzáadja a *TB*-hez az összes atomi mondatot, amelyet egy lépésben az implikációs mondatokból és a *TB*-ben már meglévő atomi mondatokból kikövetkeztethetünk.



**9.4. ábra.** Az előrefelé láncolás algoritmus által generált bizonyítási fa a bűntény példára. A kiinduló tények az alsó sorban jelennék meg, az első iterációban kikövetkeztetett tények a középső sorban, míg a második iterációban kikövetkeztetett tények a legfelső sorban.

A 9.4. ábra bemutatja a generált bizonyítási fát. Vegyük észre, hogy új következtetések már nem lehetségesek ezen a ponton, mivel minden mondatot, amelyet kikövetkeztethetnénk az előrefelé láncolással, már explicit módon tartalmaz a *TB*. Egy ilyen tudásbázist a következetési folyamat fix pontjának (*fixed point*) nevezzük. Az elsőrendű határozott klózok előrefelé láncolásával létrehozott fix pontok hasonlítanak az ítéletlogikai előrefelé láncolással (lásd 273. oldal) generáltakhoz. A leglényegesebb különbség az, hogy egy elsőrendű fix pont tartalmazhat univerzális kvantorral ellátott atomi mondatokat.

A ERL-EL-KÉRDEZ-t könnyű kielemezni. Először is megállapítható, hogy az eljárás **helyes** (*sound*), mivel minden következtetés csak az általánosított Modus Ponens alkalmazása, amelyről már igazoltuk, hogy helyes. Másodszor, **teljes** (*complete*) a határozott klózokat tartalmazó tudásbázisokra, ami azt jelenti, hogy képes minden olyan lekérdezést megválaszolni, amely következik bármely határozott kódokból álló tudásbázisból. A Datalog tudásbázisok esetére, amelyek nem tartalmaznak függvényszimbólumokat, a teljesség bizonyítása meglehetősen egyszerű. Először megszámoljuk a *TB*-hez hozzáadható tényeket, amely szám meghatározza az iterációk maximális számát. Legyen a  $k$  ez a maximális érték (argumentumok száma) az adott predikátumokra,  $p$  a predikátumok száma és  $n$  a konstansszimbólumok száma. Egyértelmű, hogy nem lehet  $pn^k$  különböző alapténynél több, tehát ennyi iteráció után az algoritmus el fog érni egy fix pontot. Ezután már az ítéletlogikai előrefelé láncolás bizonyításánál leírtakhoz nagyon hasonlóan érvelhetünk (lásd 273. oldal.) Az ítéletlogikai teljes eljárás átalakítását egy elsőrendű teljes eljárássá a 9.5. alfejezetben mutatjuk majd meg.

Függvényszimbólumokat is tartalmazó általános határozott klózok esetében az ERL-EL-KÉRDEZ végletesen számú új tényt generálhat, így ilyenkor óvatosabbnak kell lennünk. Ha egy lekérdezésre adható válaszmondat levezethető a *TB*-ből, akkor Herbrand tételekkel alkalmaznunk, hogy biztosítsuk, hogy az algoritmus megtalál egy bizonyítást (lásd a 9.5. alfejezetet a rezolúciós esetre). Ha a lekérdezésre nincs válasz, akkor az algoritmus néhány esetben nem tud leállni. Például ha a tudásbázis a Peano-axiómákat tartalmazza,

$$\text{TermSzám}(0)$$

$$\forall n \text{ TermSzám}(n) \Rightarrow \text{TermSzám}(S(n))$$

akkor az előrefelé láncolás hozzáadja a  $\text{TermSzám}(S(0))$ ,  $\text{TermSzám}(S(S(0)))$ ,  $\text{TermSzám}(S(S(S(0))))$  mondatokat és így tovább. Általánosságban ezt a problémát nem lehet kikerülni. Hasonlóan, mint az általános elsőrendű logikában, a határozott klózokkal való következetés félleg eldönthető.

## Hatókony előrefelé láncolás

A 9.3. ábrán látható előrefelé láncolási algoritmust inkább a megértés megkönnyítése céljából mutattuk be, és nem mint egy hatékonyan végrehajtható algoritmust. A komplexitásnak három lehetséges forrása van. Először is, az algoritmus „belső hurka” elvégzi az összes lehetséges egyesítés megtalálását, ahol egy szabály premisszája egyesíthető a  $TB$  egy alkalmas tényhalmazával. Ezt gyakran **mintaillesztésnek** (*pattern matching*) nevezzük, és igen költséges lépés. Másodszor, az algoritmus újra ellenőriz minden szabályt minden iterációban, hogy megvizsgálja, hogy a premisszák ki vannak-e elégítve, még olyankor is, amikor nagyon kevés változtatást végezünk a tudásbázisban az egyes ciklusokban. Végül, az algoritmus számos olyan tényt is generálhat, melyek irrelevánsak a cél szempontjából. Mindhárom problémaforrást meg fogjuk vizsgálni.

### Szabályok illesztése az ismert tényekhez

A szabályok premisszának a  $TB$  tényeihez történő illesztése egyszerű problémának tűnhet. Például tételezzük fel, hogy a következő szabályt akarjuk alkalmazni:

$$\text{Rakéta}(x) \Rightarrow \text{Fegyver}(x)$$

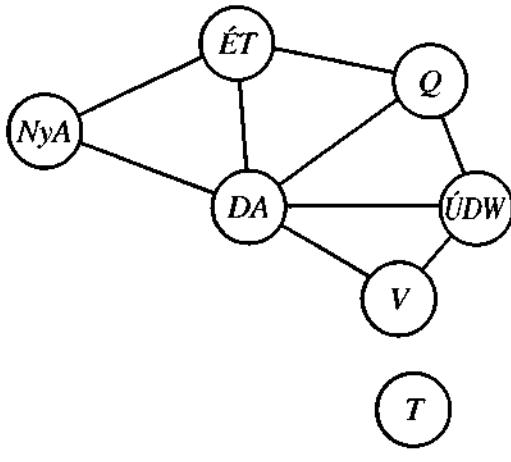
Ezután meg kell találnunk az összes tényt, amely egyesíthető a  $\text{Rakéta}(x)$ -szel. Egy megfelelően indexelt tudásbázisban ez elvégezhető a tények számával lineáris időben. Vizsgálunk meg egy szabályt, mint amilyen például a:

$$\text{Rakéta}(x) \wedge \text{Birtokol}(\text{Nono}, x) \Rightarrow \text{Elad}(\text{West}, x, \text{Nono})$$

Itt is megkereshetjük a Nono által birtokolt összes objektumot objektumonként konstans idő alatt. Ezután minden egyes objektumra meg tudjuk vizsgálni, hogy az rakéta-e.

Amennyiben a tudásbázis sok Nono által birtokolt objektumot tartalmaz és nagyon kevés rakétát, akkor azonban célszerűbb lenne először az összes rakétát megkeresni, és aztán ellenőrizni, hogy azokat Nono birtokolja-e. Ez a **konjunkt sorrendezés** (*conjunct ordering*) probléma: találunk egy olyan sorrendet, amely a szabály premissza részének konjunktjait megoldja úgy, hogy ezzel a teljes költséget minimalizáljuk. Megmutatható, hogy az optimális sorrend megtalálása önmagában is NP-nehéz, de egy jó heurisztika elérhető. Például ilyen a **legkorlátozottabb változó** (*most constrained variable*) heurisztikája, amelyet az 5. fejezetben a kényszerkielégítési problémákra használtunk. Ez azt a logikai követi, hogy rendezzük sorba a konjunktokat úgy, hogy először a rakétákat keressük, ha kevesebb rakéta van, mint Nono által birtokolt objektum.

A kapcsolat a mintaillesztés és a kényszer kielégítése között valójában nagyon szoros. Bármelyik konjunktot úgy tekinthetünk, mint az általa tartalmazott változók korlátozását – például, a  $\text{Rakéta}(x)$  egy unáris korlátozás az  $x$ -en. Ezt a gondolatot kiterjesztve,



(a)

$Kil(nya, ét) \wedge Kil(nya, da) \wedge$   
 $Kil(ét, q) \wedge Kül(ét, da) \wedge$   
 $Kül(q, údw) \wedge Kül(q, da) \wedge$   
 $Kül(údw, v, ét) \wedge Kül(údw, da) \wedge$   
 $Kül(v, da) \Rightarrow Színezhető()$

$Kil(Piros, Kék) \quad Kül(Piros, Zöld)$   
 $Kül(Zöld, Piros) \quad Kül(Zöld, Kék)$   
 $Kül(Kék, Piros) \quad Kül(Kék, Zöld)$



(b)

**9.5. ábra.** (a) Ausztrália térképének kiszínezését bemutató kényszergráf (5.1. ábra). (b) A térképszínező kényszerkielégítési probléma egy határozott klózzal reprezentálva. Végük észre, hogy a változók tárgyterülete a KÜL-re megadott alaptények konstansai által implicit módon definiáltak.

*minden véges tárgyterületű kényszerkielégítési problémát kifejezhetünk egyszeri határozott klózként, néhány társított alapténnyel kiegészítve.* Vizsgáljuk meg az 5.1. ábrán látható térképszínezési problémát, amelyet a 9.5. (a) ábrán újra bemutatunk. Egy ezzel megegyező formulát adtunk meg egy határozott klóz formájában a 9.5. (b) ábrán. Világos, hogy a *Színezhető()* konklúzió csak akkor kikövetkezhető, ha a kényszerkielégítési problémának van egy megoldása. Mivel a kényszerkielégítési problémák általában magukba foglalják a 3SAT problémákat különleges esetekként, levonhatjuk azt a következetést, hogy *egy határozott klóz illesztése egy tényhalmazhoz NP-nehéz*.

Elég elkeserítőnek tűnhet, hogy az előrefelé láncolás tartalmaz egy NP-nehéz illesztési problémát a belső hurokban. Hárrom módja van annak, hogy felvidítsuk magunkat:

- Emlékezhetünk arra, hogy a legtöbb szabály a valódi tudásbázisokban kisméretű és egyszerű (mint a bűntény példa szabályai), és nem nagy és komplex (mint a 9.5. ábrán látható kényszerproblémánál). Az adatbázisok területén fel szokták tételezni, hogy mind a szabályok mérete, mind a predikátumok argumantumszáma egy konstans-sal megadható korlát alatt marad, és így csak az **adatkomplexitás (data complexity)** miatt kell aggódni – vagyis a következetés komplexitása miatt, ami az adatbázisban lévő alaptények számának függvénye. Könnyű megmutatni, hogy az előrefelé láncolás adatkomplexitása polinomiális.
- Tekinthetjük a szabályok azon csoportját, amelyekre az illesztés hatékony tud lenni. Alapjában véve minden Datalog klózt tekinthetünk úgy, mint ami egy kényszerkielégítési problémát határoz meg, így az illesztés kivitelezhető akkor, ha a megfelelő kényszerkielégítési probléma is nyomon követhető. Az 5. fejezet leírja a kényszerkielégítési problémák néhány praktikusan megoldható családját. Például ha a kényszergráf (egy olyan gráf, amelynek a csomópontjai változók és az élei kényszerek) fát formáz, akkor a kényszerkielégítési probléma lineáris időben megoldható. Pontosan

ugyanez a szabály áll fenn a szabályillesztésre. Például ha eltávolítjuk Dél-Ausztráliát a 9.5. ábráról, akkor az új klóz a következő lesz:

$$Kü\ell(nya, ét) \wedge Kü\ell(ét, q) \wedge Kü\ell(q, údw) \wedge Kü\ell(údw, v) \Rightarrow Színezhető()$$

Ez megfelel az 5.11. ábrán bemutatott redukált kényszerkielégítési problémának. A faszerkezetű kényszerkielégítési problémák megoldására használt algoritmusokat közvetlenül alkalmazhatjuk a szabályillesztés problémájára.

- És végül dolgozhatunk azon, hogy megszüntessük a felesleges szabályillesztési kísérleteket az előrefelé láncolási algoritmusban, amely a következő alfejezet témája lesz.

## Inkrementális előrefelé láncolás

Amikor a bűntény példán bemutattuk az előrefelé láncolás működését, akkor csaltunk, nevezetesen abban, hogy kihagytunk néhány szabályillesztést, amelyet a 9.3. ábrán bemutatott algoritmus elvégzett. Például a második iterációban a:

$$Rakéta(x) \Rightarrow Fegyver(x)$$

szabály (ismét) illeszthető a *Rakéta(M<sub>1</sub>)*-hez, és természetesen a *Fegyver(M<sub>1</sub>)* konklúziót már ismerjük, így semmi sem történik. Az ilyen felesleges szabályillesztést elkerülhetjük, ha figyelembe vesszük a következő megfigyelést: *Minden, a t-edik iterációban kikövetkeztethető új tény levezetéséhez szükséges legalább egy, a t – 1 ciklusban kikövetkeztetett új tény felhasználása*. Ez azért igaz, mert bármely olyan következtetés, amely nem igényel egy új tényt a t – 1 ciklusból, már elvégezhető lett volna a t – 1 ciklusban.

Ez a megfigyelés természetes módon elvezet minket egy inkrementális előrefelé láncolási algoritmushoz, ahol a t ciklusban csak akkor ellenőrzünk egy szabályt, ha annak premisszája tartalmaz egy p<sub>i</sub> konjunktot, amely egyesíthető egy p<sub>i</sub>' tényivel, és amelyre újonnan következtettünk a t – 1 ciklusban. A szabályillesztő lépés aztán rögzíti a p<sub>i</sub>-t, hogy illeszkedjen a p<sub>i</sub>'-hez, de lehetővé teszi, hogy a szabály többi konjunktja illeszkedjen bármely megelőző ciklus tényeihez. Ez az algoritmus pontosan ugyanazokat a tényeket generálja minden egyes ciklusban, mint amelyeket a 9.3. ábrán látható algoritmus, de annál sokkal hatékonyabb.

Megfelelő indexeléssel könnyű megtalálni azokat a szabályokat, amelyeket egy adott tény kielégíthetővé tehet. Valójában számos rendszer egy ilyen frissítési módban működik, ahol az előrefelé láncolás minden egyes olyan tényre aktualizálódik, amelyet KIELENT-ettünk a rendszernek. A következtetések sorban végigveszik a szabályok halmazát, amíg el nem érik a fix pontot, és ez a folyamat a következő új tény nél újra kezdődik.

A tudásbázisban lévő szabályoknak tipikusan csak egy kis töredékét eredményezi egy adott tény hozzáadása. Ez azt jelenti, hogy jelentős mennyiségű felesleges munkát végzünk néhány ki nem elégített premisszát is tartalmazó részleges illesztések ismételt létrehozásával. A bűntény példánk túl kicsi ahhoz, hogy ezt megfelelően bemutassuk, de vegyük észre, hogy egy részleges illesztést az első ciklusban már létrehoztunk az:

$$Amerikai(x) \wedge Fegyver(y) \wedge Elad(x, y, z) \wedge Ellenséges(z) \Rightarrow Bünöző(x)$$

szabály és az Amerikai(West) tény között. Ezt a részleges illesztést aztán kiselejtezzük, de újraépítjük a második ciklusban is (amikor a szabály sikeres). Hatékonyabb lenne megőrizni, és fokozatosan kiegészíteni újabb részleges illesztésekkel, amikor az új tények beérkeznek, ahelyett hogy kiselejteznénk őket.

A rete algoritmus<sup>3</sup> volt az első, amely alaposan foglalkozott ezzel a problémával. Az algoritmus feldolgozza a tudásbázis szabályait, hogy létrehozzon egy adatfolyam-hálózatot, amelyben minden csomópont egy literál egy szabály premissza részéből. Változó lekötések áramlanak a hálózaton, megszűrve azokat, amelyek nem illeszkednek egy literálhoz. Ha egy szabályban két literálnak ugyanaz a változója – például  $Elad(x, y, z)$   $\wedge$   $Ellenséges(z)$  a bűnöző példában –, akkor a literálokhoz tartozó lekötések egy egyenlőségi csomópontron mennek keresztül. Egy változó lekötésnek, amely elér egy csomópontot egy  $n$  argumentumú literálnál úgy, mint az  $Elad(x, y, z)$ , várakoznia kell, mielőtt a folyamat újra kezdődne, amíg a többi változóhoz tartozó lekötéseket is létrehozzuk. Egy adott pontban a rete háló állapota megadja a szabályok összes, addig elvégzett részleges illesztéseit, és figy elkerülhető a jelentős újraszámolás.

A rete hálók, és más, hatékonyságot növelő fejlesztések mindig is jelentős szerepet játszottak az úgynevezett produkciós rendszerekben (*production systems*), amelyek az első széles körben használt előrefelé láncolási rendszerek voltak.<sup>4</sup> Az XCON szakértői rendszer (eredetileg R1-nek hívták, McDermott, 1982) egy produkciós rendszer felépítését felhasználva készült. Az XCON néhány ezer szabályt tartalmazott számítógéptartozékok konfigurációinak megtervezésére a DEC cégtől vásárlónak számára. Ez volt az egyik első igazi kereskedelmi siker a szakértő rendszerek feltörekvő piacán. Sok más hasonló rendszer épült ugyanezt a technológiát felhasználva, amelyet be is építettek egy általános célú programozási nyelvbe, az OPS-5-be.

A produkciós rendszerek a **kognitív architektúrákban** (*cognitive architectures*), mint például az ACT (Anderson, 1983) vagy a SOAR (Laird és társai, 1987), is népszerűek. A kognitív architektúrák az emberi gondolkodás modelljei. Az ilyen rendszerekben a rendszer „munkamemoriája” az ember rövid távú memóriáját modellezzi, és a produkált következmények a hosszú távú memória részei. A működés minden egyes ciklusában a produkciókat a tények munkamemoriájához illesztik. Egy olyan következmény, amelynek feltételeit kielégítik, hozzátehet és kitörölhet tényeket a működő memóriából. Az adatbázisokkal ellentétben a produkciós rendszerekben gyakran sok szabály és viszonylag kevés tény van. Megfelelően optimalizált illesztési technológiával néhány modern rendszer képes valós időben működni több mint egymillió szabállyal is.

## Irreleváns tények

Úgy tűnik, hogy a rossz hatékonyság problémájának az utolsó forrása az előrefelé láncolásban a megközelítésből fakad, ez már az ítéletlogikai kontextusban is felmerült. (Lásd 7.5. alfejezet). Az előrefelé láncolás az ismert tényeken alapuló összes lehetséges következtetést elvégzi, akkor is, ha azok irrelevánsak az elérendő célhoz. A bűntény példánkban nem voltak olyan szabályok, amelyek irreleváns konklúziókat vontak volna

<sup>3</sup> A rete hálót jelent latinul, angol kiejtésben (riti) a „treaty”-vel römel.

<sup>4</sup> A produkció kifejezés a produkciós rendszerekben egy feltétel-cselekvés szabályt jelent.

maguk után, így az irányítottság hiánya nem jelentett problémát. Más esetekben (például ha több szabályunk van, amelyek leírják az amerikaik étkezési szokásait és a rakéták árat is), a ERL-EL-KÉRDEZ számos irreleváns konklúziót fog generálni.

Az irreleváns konklúziók elkerülésének egy lehetséges módja a hátrafelé láncolás használata, amint az a 9.4. alfejezetben látható lesz. Másik megoldás, hogy az előrefelé láncolást a kiválogatott szabályok egy részhalmazára korlátozzuk. Ezt a megközelítést az ítéletlogikai kontextusban már tárgyaltuk. Egy harmadik megközelítési használnak a deduktív adatbázisok területén, ahol az előrefelé láncolás elterjedt eszköznek számít. Az alapötlet az, hogy írjuk át a szabályhalmazt felhasználva a célállítást, hogy aztán csak releváns változó kapcsolatokat – amelyek az úgynevezett **mágikus halmazhoz** (*magic set*) tartoznak – vegyük figyelembe az előrefelé következetésben. Például ha a cél állítás a *Bűnöző(West)*, akkor a szabályt, amely a *Bűnöző(x)*-re következtet, át kell írni úgy, hogy tartalmazzon egy további konjunktot, amely korlátozza az *x* értékét:

$$\text{Mágikus}(x) \wedge \text{Amerikai}(x) \wedge \text{Fegyver}(y) \wedge \text{Elad}(x, y, z) \wedge \text{Ellenséges}(z) \Rightarrow \text{Bűnöző}(x)$$

A *Mágikus(West)* tény is hozzáadódik a *TB*-hez. Ily módon, még akkor is, ha a tudásbázis amerikaiak millióiról tartalmaz is adatokat, csak West ezredest fogjuk figyelembe venni az előrefelé láncolási folyamatban. A mágikus halmazok definíálásának és a tudásbázis átírásának teljes folyamata túl összetett ahhoz, hogy most itt részletezzük, de az alapötlet egyfajta „generikus” hátrafelé való következetés elvégzése a célból kiindulva azért, hogy megtaláljuk, mely változókapcsolatokat kell korlátozni. A mágikus halmaz megközelítést egy hibrid algoritmusnak tekinthetjük az előrefelé következetés és a hátrafelé haladó előfeldolgozó folyamat között.

## 9.4. HÁTRAFELÉ LÁNCOLÁS

A logikai következetési algoritmusoknak a második nagy családja a 7.5. alfejezetben bemutatott hátrafelé láncolás (*backward chaining*) megközelítést használja. Ezek az algoritmusok a célból kiindulva hátrafelé dolgoznak, láncszerűen haladva a szabályokon keresztül, hogy megtalálják a bizonyítási alátámasztó már ismert tényeket. Először bemutatjuk az alapvető algoritmust, majd leírjuk, hogyan használják ezt a logikai programozásban (*logic programming*), amely a legelterjedtebb formája az automatizált következetéseknek. Azt is látni fogjuk, hogy a hátrafelé láncolásnak vannak hátrányai is az előrefelé láncolással szemben, és meg fogjuk vizsgálni, milyen módon kerekedhetünk felül ezeken. Végül megmutatjuk a logikai programozás és a kényszerek kielégítésének problémája közötti szoros kapcsolatot.

### Egy hátrafelé láncolási algoritmus

A 9.6. ábra egy egyszerű hátrafelé láncolási algoritmust mutat be, az ERL-HL-KÉRDEZ-t. Az algoritmust célok listájával hívjuk meg, amely kezdetben egy elemet tartalmaz, az eredeti lekérdezést, és az eljárás visszaadja az összes olyan helyettesítéshalmazt, amelyek kielégítik a lekérdezést. A célok listáját úgy tekinthetjük, mint egy feldolgozásra váró „vermet”. A bizonyítás aktuális ága akkor lesz sikeres, ha a verem összes elemét

```

function ERL-HL-KÉRDEZ(TB, célok,  $\theta$ ) returns helyettesítések egy halmaza
inputs: TB, egy tudásbázis
célok, egy lekérdezést alkotó konjunkciók egy listája, ahol  $\theta$  már alkalmazva van
 $\theta$ , az aktuális helyettesítés, kezdetben az üres helyettesítés {}
local variables: válaszok, helyettesítések egy halmaza, kezdetben üres

if célok üres then return {}  

 $q' \leftarrow \text{HELYETTESÍTÉS}(\theta, \text{Első}(célök))$   

for each r mondatra in TB, ahol VALTOZÓ-ÁTNEVEZ(r) = ( $p_1 \wedge \dots \wedge p_n \Rightarrow q$ )  

    és  $\theta' \leftarrow \text{EGYESÍT}(q, q')$  sikeres  

    új_célok  $\leftarrow [p_1, \dots, p_n | \text{MARADÉK}(célök)]$   

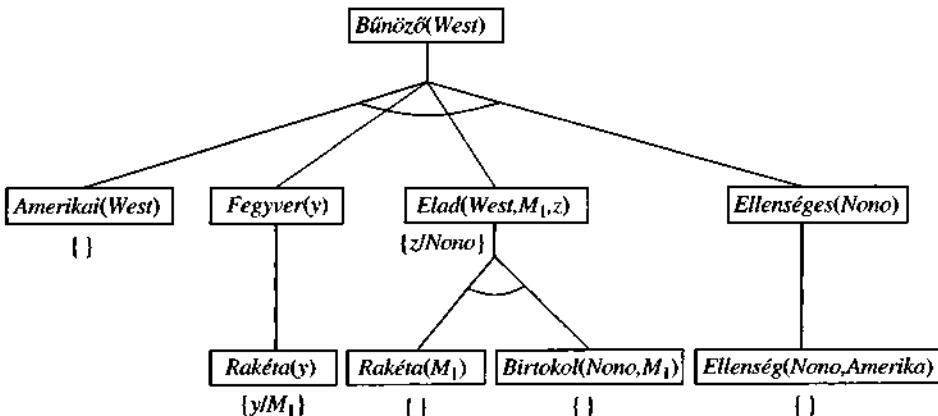
    válaszok  $\leftarrow \text{ERL-HL-KÉRDEZ}(TB, \text{új_célok}, \text{KOMPOZÍCIÓ}(\theta', \theta)) \cup \text{válaszok}$   

return válaszok

```

9.6. ábra. Egy egyszerű hátrafelé láncolási algoritmus

ki tudjuk elégíteni. Az algoritmus először veszi a lista első célját, és megtalálja az összes olyan klózt a tudásbázisban, amelynek a pozitív literálja, vagyis a feje (**head**) egyesíthető a céllal. minden ilyen klóz létrehoz egy új rekurzív hívást, amelyben a premissza, vagyis a klóz **törzse (body)** hozzá lesz adva a célveremhez. Emlékezzünk, hogy a tények olyan klózok, amelyeknek fejük van, de törzsük nincs, így amikor egy cél egyesül egy ismert tényivel, nem adunk új alcélokat a veremhez, és így a célt megoldjuk. A 9.7. ábra a *Bűnöző(West)* származtatásának a bizonyítási fáját mutatja be, a (9.3) mondataiból kiindulva a (9.10)-en keresztül.



9.7. ábra. Egy hátrafelé láncolással létrehozott bizonyítási fa, annak bizonyítására, hogy West egy bűnöző. A fát mélyiségi kereséssel kell kiértékelni, balról jobbra. Hogy bebizonyítsuk a *Bűnöző(West)* klózt, be kell bizonyítanunk az alatta elhelyezkedő négy konjunktot is. Néhányan közülük megtalálhatók a tudásbázisban, míg mások további hátrafelé láncolást igényelnek. Mindegy egyes sikeres egyesítés lekötéseit láthatjuk a megfelelő részcél mellett. Jegyezzük meg, hogy mihelyt egy konjunkcióban egy részcél sikeres, helyettesítéseit a következő részcélokhoz alkalmazzuk. Így tehát, mire az ERL-HL-KÉRDEZ eljut az utolsó konjunkciójig, az eredetileg *Ellenséges(z)*-ig, az már a *Nono*-hoz lesz lekötve.

Az algoritmus a helyettesítések egy kompozícióját (**composition**) használja. A  $KOMPOZÍCIÓ(\theta_1, \theta_2)$  olyan helyettesítés, amelynek hatása ugyanaz, mintha sorban minden helyettesítést alkalmaznánk. Tehát:

$$\text{HELYETTESÍT}(KOMPOZÍCIÓ(\theta_1, \theta_2), p) = \text{HELYETTESÍT}(\theta_2, \text{HELYETTESÍT}(\theta_1, p))$$

Az algoritmusban a  $\theta$ -ban tárolt aktuális változóleketést összevonjuk azokkal a lekötésekkel, amelyeket a célnak a klózfejjel való egyesítése eredményez, megkapva így egy új aktuális lekötéshalmazt a visszafelé történő híváshoz.

A hátrafelé láncolás, ahogyan bemutattuk, egy tiszta mélységi keresési algoritmus. Ez azt is jelenti, hogy a helyigénye a bizonyítás méretének lineáris függvénye (most figyelmen kívül hagyva a megoldások összegyűjtéséhez szükséges helyet). Ez azt is jelenti, hogy a hátrafelé láncolás alkalmazása (eltérően az előrefelé láncolástól) olyan ismert problémákkal jár, mint amilyen az ismétlődő állapotok és a nem teljesség. Később meg fogjuk vizsgálni ezeket a problémákat és néhány lehetséges megoldást, de először nézzük meg, hogyan használják a hátrafelé láncolást a logikai programozási rendszerekben.

## Logikai programozás

A logikai programozás közel áll egy olyan technológiához, amely megtestesíti azt a deklaratív ideált, amelyet a 7. fejezetben írtunk le, miszerint a rendszereket úgy kell létrehozni, hogy a tudást formális nyelven fejezzük ki, és a problémákat egy következetési folyamat végigfuttatásával oldjuk meg. Ezt az ideális megközelítést Robert Kowalski egyenlete foglalja össze:

$$\text{Algoritmus} = \text{Logika} + \text{Vezérlés}$$

A Prolog messze a legszélesebb körben használt logikai programozási nyelv. Több százezer használója van. Elsősorban gyors prototípusnyelvként használták, valamint szimbólummanipulációs feladatokhoz, mint például fordítóprogramok írására (Van Roy, 1990) vagy természetes nyelvek elemzésére (Pereira és Warren, 1980). Számos szakértőrendszer írtak Prologban jogi, orvosi, pénzügyi és más tárgyterületeken.

A Prolog programok határozott klózok halmazai, jelölési rendszere kissé eltér a standard elsőrendű logikáétól. A Prolog nagybélűket használ a változókra és kisbélűket a konstansokra. A klózok leírásában a fej megelőzi a törzset; a „.” -t használja a bal oldali implikációra, vesszők választják el a literálokat a törzsben, és pont jelzi a mondat végét:

bűnöző(X) :- amerikai(X), fegyver(Y), elad(X, Y, Z), ellenséges(Z)

A Prolog a listák jelölésére és az aritmetikára tartalmazza a „szintaktikai nyalánkságot”. Példaként íme egy Prolog program a `csatol(X, Y, Z)` -re, amely akkor sikeres, ha a Z lista az X és az Y listák csatolásának az eredménye:

```
csatol([], Y, Y)
csatol([A1|2 X, Y, A2|Z]) :- csatol(X, Y, Z)
```

Természetes nyelven úgy olvashatóak ezek a klózok, hogy (1) egy üres listát hozzácsatolva az Y listához, egy ugyanolyan Y listát kapunk eredményként, és a (2) szerint

az  $[A|Z]$  az  $[A|X]$  Y-hoz csatolásának az eredménye. A csatol ezen definíciója eléggyé hasonlónak tűnik a Lispben megtalálható megfelelő definícióhoz, de valójában ez sokkal kifejezőbb. Például feltehetjük a következő kérdést: csatol( $A, B, [1, 2]$ ): melyik két listát kell csatolni ahhoz, hogy összeállítsuk az  $[1, 2]$ -t? A programot futtatva ezeket a megoldásokat kapjuk vissza:

```
A = []      B = [1, 2]
A = [1]    B = [2]
A = [1, 2] B = []
```

A Prolog-programok véghajtása mélyiségi keresést alkalmazó hátrafelé láncoláson keresztül történik, ahol a klózokat olyan sorrendben alkalmazzák, ahogyan a tudásbázisban szerepelnek. A Prolog néhány aspektusa a standard logikai következtetésen kívül esik.

- Beépített aritmetikai függvényeket tartalmaz. Az ilyen függvényszimbólumokat használó literálokat a kód véghajtásával „bizonyítja” a program, és nem végez további következtetéseket. Például az „X egyenlő 4+3” akkor sikeres, ha az X értékét a 7-hez kötjük. Ugyanakkor, az a célállítás, hogy: „5 egyenlő X+Y” sikertelen lesz, mert a beépített függvények nem képesek önállóan egy téteszűleges egyenletmegoldást elvégezni.<sup>5</sup>
- Vannak olyan beépített predikátumok, amelyeknek mellékhatásai vannak, ha véghajtjuk őket. Ilyenek a bemeneti-kimeneti predikátumok és a tényeket a *TB*-hez hozzáadó, illetve onnan törlő kijelent/visszavon predikátumok. Ezeknek a predikátumoknak nincs megfelelőjük a logikában, és hatásukat sokszor nem egyszerű végigkövetni – például ha a tényeket egy olyan bizonyítási fa egyik ágán jelentjük ki, amely végül sikertelennek bizonyul.
- A Prolog lehetővé tesz olyedi formájú tagadást, amelyet **sikertelenségi tagadásnak** (**negation as failure**) nevezünk. Egy negált cél, mint a nem P, akkor tekintető bizonyítottanak, ha a rendszernek nem sikerül bebizonyítania a P-t. Így a következő mondat:

```
élő(X) :- nem halott(X)
```

úgy olvasható, hogy: „Mindeni élő, ha nem bizonyítható, hogy halott.”

- A Prolognak van egy egyenlőségi operátora (=), de ez nem rendelkezik a logikai egyenlőség teljes hatékonyságával. Egy egyenlőséget tartalmazó célállítás akkor sikeres, ha a két term *egyesíthető*, máskülönben sikertelen. Tehát az  $X + Y = 2 + 3$  akkor sikeres, ha az X lekötött értéke a 2, és az Y lekötött értéke a 3, de a *hajnalcsillag=esthajnalcsillag* sikertelen. (A klasszikus logikában az utóbbi egyenlet lehet igaz, vagy lehet nem igaz.) Az egyenlőségről sem tényt, sem szabályt nem tudunk hozzáadni a *TB*-hez.
- A Prolog egyesítési algoritmusára nem tartalmazza az **előfordulási próbát**. Ennek az a következménye, hogy lehetséges nem helyes következtetést végezni. Ez azonban ritkán okoz problémát, leginkább olyankor, amikor a Prologot matematikai tételek bizonyítására használjuk.

<sup>5</sup> Jegyezzük meg, hogy ha a Peano-axiómák meg vannak adva, akkor az ilyen célok megoldhatók egy Prolog-programon belüli következtetéssel.

A Prolog tervezésénél meghozott döntések kompromisszumot jelentenek a deklarativitás és a végrehajtási hatékonyúság között – mármint amit hatékonyságon értettünk a Prolog tervezésének idején. Visszatérünk majd ehhez a témahez, miután megvizsgáltuk, hogyan valósítják meg a Prologot.

## A logikai programok hatékony megvalósítása

Egy Prolog program végrehajtása kétféle módon történhet: lehet interpretált és lehet lefordított. Az interpretáció általában az ERL-HL-KÉRDEZ algoritmus (lásd 9.6. ábra) futtatását jelenti a programmal mint tudásbázissal. Azt mondjuk, hogy „általában”, mivel a Prolog különböző interpretációjai különböző megoldásokat tartalmaznak a sebesség maximalizálására. Itt mi csak két implementációt nézünk meg.

Először is ahelyett, hogy minden lépésben összeállítanánk az összes lehetséges válasz listáját minden egyes részcélhoz, a Prolog interpreter egy választ hoz létre, és egy „ígéretet” ad meg arra, hogy a többi lehetséges választ is generálni fogja, miután az aktuális választ teljes mértékben feltártá. Ezt az ígéretet **választási pontnak** (*choice point*) nevezzük. Mikor a mélyiségi keresés befejezte az aktuális válaszból fakadó lehetséges megoldások feltárását, és visszalép a választási ponthoz, a választási pontot kiterjesztjük úgy, hogy megadjon egy új választ a részcélhoz, és egy új választási pontot. Ez a megközelítés időt és tárolóhelyet takarít meg. Ezenkívül nagyon egyszerű határfelületet biztosít a hibakereséshez, mert egy adott pillanatban csak egyetlen megoldási útvonal vizsgálata történik meg.

Másodsor, az ERL-HL-KÉRDEZ egyszerű alkalmazása a helyettesítések generálásakor és kompozíciójakor jól gazdálkodik az idővel. A Prolog a helyettesítéseket úgy alkalmazza, hogy olyan logikai változókat használ, amelyek emlékezni tudnak az aktuális lekötéseikre. A program futásának bármely pontján a programban lévő minden egyes változó vagy szabad, vagy valamely értékhez lekötött. Ezek a változók és értékek együtt definílják implicit módon a helyettesítést a bizonyítás aktuális ágához. A vizsgált ágak kiterjesztése csak további változókötéseket hoz létre, mivel egy új lekötés hozzáadása egy már lekötött változóhoz az egyesítés sikertelenségét eredményezi. Mikor a keresésben sikertelennek bizonyul egy ág, a Prolog visszalép egy még előző választási ponthoz, és ekkor esetleg fel kell oldania néhány változót. Ezt úgy hajta végre, hogy nyomon követi a veremben – amelynek **útvonal** (*trail*) a neve – lekötött összes változót. Mivel minden egyes új változó az EGYESÍT-VÁLT-tal kapcsolódik, a változót beteszi az útvonalba. Amikor egy cél sikertelen, és ideje visszatérni egy megelőző választási ponthoz, minden egyes változó szabaddá válik. minthogy kivesszük az útvonalból.

Még a leghatékonyabb Prolog interpreteknek is néhány ezer gépi utasításra van szükségük egy következetési lépés végrehajtásához; az index kikeresésének, az egyesítésnek, a rekurzív hívási verem felépítésének a költsége miatt. Valójában az interpreter úgy viselkedik, mint aki még soha sem láitta a programot. Például újra meg kell találnia a célohoz illeszkedő klózokat. Egy lefordított Prolog program ezzel szemben egy olyan következetési eljárás, amelyet egy specifikus klózhalmazra készítettek, és így tudja, mely klózok illeszthetők az adott célohoz. A Prolog fordító alapjában véve minden egyes különböző predikátumra egy miniatűr tételbizonyítót generál, kiküszöbölvé így az

interpretációs lépések nagy részét. Lehetséges az is, hogy az egyesítési eljárást **nyitott kóddal (open-code)** lássuk el az egyes eljárás hívásokra, elkerülve így a termék struktúrájának explicit elemzését. A nyitott kódú egyesítés részleteit megtalálhatók a (Warren és társai, 1977)-ben.

A mai számítógépek utasításkészletei szegényesnek tűnnek összehasonlítva a Prolog szemantikájával, így tehát a Prolog fordítói egy átmeneti nyelvre fordítják le ahelyett, hogy közvetlenül gépi nyelvre fordítanák. A legnépszerűbb átmeneti nyelv a Warren Absztrakt Gép (Warren Abstract Machine) vagy WAM, amelyet David H. D. Warrenről neveztek el, aki az első Prolog fordító egyik megvalósítója volt. A WAM a Prolog számára alkalmas absztrakt utasításkészlet, és interpretálható vagy lefordítható gépi nyelvre. Más fordítók, mint például a Lisp vagy a C, egy magas szintű nyelvre fordítják le a Prologot, és ezután használják a nyelvnek a fordítóját, hogy azt a gépi nyelvre fordítsa. Például, a **Csatol** definícióját a 9.8. ábrán bemutatott kódra lehet fordítani. Van még néhány kérdés, amit fontos megemlíteni:

- Ahelyett hogy keressélni kellene a tudásbázisban a **Csatol** klózokat, a klózokat eljárassá lehet alakítani, és a következetéseket egyszerűen úgy hajtjuk végre, hogy meghívjuk az eljárást.
- Mint azt korábban leírtuk, az aktuális változók lekötéseit egy útvonalhoz rendeljük. Az eljárás első lépése elmenti az útvonal pillanatnyi állapotát úgy, hogy azok visszaállíthatóak lesznek a **VISSZAÁLLÍT-ÚTVONAL**-lal, ha az első klóz sikertelen. Ez fel fog oldani minden lekötést, amelyet az első **EYESÍT** hívás generált.
- A legtrükkösebb rész a **folytatások (continuations)** listájának használata a választási pontok implementálására. Gondolhatunk úgy egy folytatásra, mint egy eljárás és egy argumentumlista becsomagolására, melyek együtt határozzák meg, hogy mit kell legközelebb csinálni, amikor az aktuális cél sikeres lesz. Nem elég csak úgy egyszerűen visszatérni egy olyan eljárásból, mint a **CSATOL**, amikor a cél sikeres, mivel itt sokféle módon lehet sikeres az eljárás, és minden egyes esetet fel kell deríteni. A folytatás argumentum megoldja ezt a problémát, mert minden egyes esetben lehívhatjuk, ha egy cél sikeres. A **CSATOL** kódban, ha az első argumentum üres, akkor a **CSATOL** predikátum sikeres. Ezután **HÍVJUK** a folytatást, az útvonalhoz rendelt lekötésekkel, hogy végrehajtsa, amit a folytatásban legközelebb meg kell tenni. Például ha a **CSATOL** hívása a legfelső szinten történik, a folytatás kinyomtatja a változók lekötéseit.

```
procedure CSATOL(ax, y, az, folytatás)
```

```
    útvonal ← GLOBÁLIS-ÚTVONAL-MUTATÓ()
    if ax = [] és EYESÍT(y, az) then Hív(folytatás)
    VISSZAÁLLÍT-ÚTVONAL(útvonal)
    a ← ÚJ-VÁLTOZÓ(); x ← ÚJ-VÁLTOZÓ(); z ← ÚJ-VÁLTOZÓ()
    if EYESÍT(ax, [a | x]) és EYESÍT(az, [a | z]) then CSATOL(x, y, z, folytatás)
```

**9.8. ábra.** A **Csatol** predikátum fordításának pszeudokódos reprezentációja. Az ÚJ-VÁLTOZÓ függvény egy új változót ad vissza, amely különbözik a korábban visszaadott változóktól. A **Hív(folytatás)** eljárás folytatja a végrehajtást a megadott folytatással.

Mielőtt Warren a Prolog fordítót elkészítette volna, a logikai programozás túl lassú volt általános célú felhasználáshoz. A Warren és mások által létrehozott fordítók lehetővé tették, hogy a Prolog kód olyan sebességet érjen el, amelyek – különböző standard tesztek szerint is – versenyképessé tettek a C nyelvvel (Van Roy, 1990). Természetesen az a tény, hogy egy tervkészítő vagy egy elemzőt egy természetes nyelvhez néhány tücat Prolog sorban meg lehet írni, lényegesen vonzóbbá teszi a Prologot a C-nél a legtöbb kisebb MI-kutatási projekt prototípusának az elkészítésénél.

A párhuzamosítás szintén jelentős gyorsítást eredményezhet. Két elsődleges tere van a párhuzamosításnak. Az első, amelynek neve **VAGY-párhuzamosság (OR-paralellism)**, abból a lehetőségből származik, hogy egy cél számos különböző klózzal lehet egyesíthető a tudásbázisban. Mindegyik egyesítés egy potenciális megoldáshoz vezető független ág megjelenését eredményezi a keresési helyen, és minden ilyen ágat párhuzamosan is megoldhatunk. A második, amelynek neve **ÉS-párhuzamosság (AND-paralellism)**, abból származik, hogy az implikáció törzsében minden egyes konjunktot egy párhuzamos implikációval is meg lehet oldani. Az ÉS-párhuzamosságot nehezebb megvalósítani, mert a megoldás az egész konjunkcióra konzisztsen lekötésekkel igényel az összes változóra. Így minden egyes konjunktív ágnak kommunikálnia kell a többi ággal, hogy biztosítsák a globális megoldást.

## Redundáns következtetés és végtelen hurkok

Most nézzük meg a Prolog nyelv Achilles-sarkát, a mélységi keresés és az ismételt állapotokat és végtelen útvonalakat tartalmazó fák illeszkedésének hiányát. Vizsgáljuk meg a következő logikai programot, amely előönti, hogy létezik-e egy útvonal két pont között egy irányított gráfban:

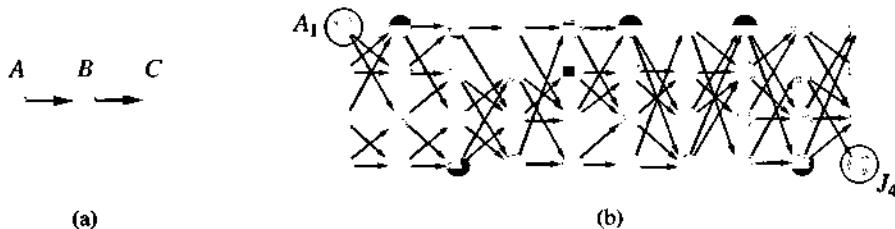
```
útvonal(X, Z) :- kapcsolat(X, Z)
útvonal(X, Z) :- útvonal(X, Y), kapcsolat(Y, Z)
```

Egy egyszerű, három csomópontos gráfot, amelyet a `kapcsolat(a,b)` és a `kapcsolat(b,c)` definiál, mutat be a 9.9. (a) ábra. Az `útvonal(a,c)` lekérdezésére a program a 9.10. (a) ábrán látható bizonyítási fát generálja. Másrészt viszont, ha a két klózt ebbe a sorrendbe tesszük:

```
útvonal(X, Z) :- útvonal(X, Y), kapcsolat(Y, Z)
útvonal(X, Z) :- kapcsolat(X, Z)
```

akkor a Prolog a 9.10. (a) ábrán bemutatott végtelen keresést végzi. A Prolog tehát, mint egy tételelbizonyító, határozott klózokból álló *TB*-re **nem teljes (incomplete)** (még a Datalog programok esetében sem teljes, mint azt a példánk mutatja), mivel léteznék olyan tudásbázisok, amelyekben nem tud bizonyítani vonzat mondatokat. Vegyük észre, hogy az előrefelé láncolással nincs ilyen probléma: ha egyszer az `útvonal(a,b)`, az `útvonal(b,c)` és az `útvonal(a,c)` útvonalak ki lettek következtetve, az előrefelé láncolás leáll.

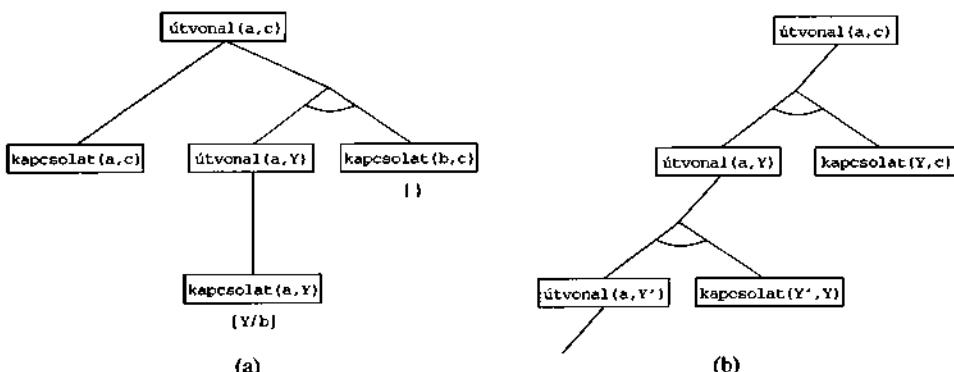
A mélységi keresésen alapuló hátrafelé láncolásnak is akad problémája a felesleges számításokkal. Például amikor meg akarunk találni egy útvonalat az  $A_1$ -ből a  $J_4$ -be a 9.9. (b) ábrán, a Prolog 877 következtést hajt végre, amelyek legnagyobb



**9.9. ábra.** (a) Az A-ból a C-be vezető útvonal megtalálása a Prologot egy végtelen hurokba vezetheti. (b) Egy gráf, amelyben minden csomópontot két véletlenszerű következő rétegbeli csomóponthoz kapcsolunk a következő rétegen. Egy útvonal megtalálása az A<sub>1</sub>-ból a J<sub>4</sub>-be 877 következtést igényel.

részét a csomópontokhoz tartozó összes lehetséges útvonal megtalálása képezi, amelyből pedig nem lehet elérni a célt. Ez hasonló a 3. fejezetben megtárgyalt ismételt állapot problémához. A következtetési lépések teljes száma a generált alaptények számának exponenciális függvénye lehet. Ha az előrefelé láncolást alkalmazunk ehelyett, akkor legfeljebb  $n^2$  útvonal (X, Y) tény generálható az n csomópont összekötésével. A 9.9. (b) ábrán látható problémához csak 62 következtetési lépés szükséges.

Az előrefelé láncolás a gráfkeresési problémáknál a **dinamikus programozás** (**dynamic programming**) egyik példája, amelyben a részproblémákra vonatkozó megoldások előállítása inkrementálisan történik a kisebb részproblémáktól indulva, végig tárolva a részmegoldásokat, hogy elkerüljük az újraszámolást. Ugyanezt a hatást érhetjük el egy hátrafelé láncoló rendszerben a memók gyűjtésének (**memoization**) használatával, ami azt jelenti, hogy a részcélok megoldásait tároljuk, amint megtaláltuk őket, hogy azután újra felhasználhassuk ezeket a megoldásokat, amikor a részcél visszatér, megtakarítva így az előző számítás meigmétlését. Ezt a megközelítést használják a **táblázatos logikai programozási** (**tailed logic programming**) rendszerek, amelyeknek hatékony tárolási és visszakeresési mechanizmusuk van a memók gyűjtésének megvalósítására. A táblázatos logikai programozás egyesíti a hátrafelé láncolás célirányosságát az előre-



**9.10. ábra.** (a) Az A-ból a C-be vezető útvonal bizonítása. (b) Végtelen mélységű bizonítási fa, amely akkor generálódik, ha a klózok „rossz” sorrendben vannak.

felé láncolás dinamikus programozási hatékonyságával. Ez az eljárás teljes a Datalog programokra is, ami azt eredményezi, hogy a programozónak kevesebbet kell aggódnia a végétlen hurkok problémája miatt.

## Korlátozott logikai programozás

Az előrefelé láncolás tárgyalásánál (lásd 9.3. alfejezet) bemutattuk, hogy a kényszerkielégítési problémákat hogyan lehet határozott klózokként kódolni. A standard Prolog képes megoldani az ilyen problémákat, pontosan úgy, mint az 5.3. ábrán bemutatott visszalépéses algoritmus.

Mivel a visszalépéses módszer sorba veszi a változók tárgyterületeit, ezért csak véges tárgyterületű (*finite domain*) kényszerkielégítési problémákra működik. A Prolog termjeiben véges számú megoldásnak szabad léteznie bármely szabad változókkal rendelkező célra. (Például, a kül (q, sa) célnak, amely azt jelenti ki, hogy Queens-landnek és Dél-Ausztráliának különböző színűnek kell lenni, hat megoldása van, ha három szín megengedett.) A végétlen méretű tárgyterületekre alkalmazott módszerek – például az egész számokat felvevő vagy valós értékű változókat tartalmazó problémáknál alkalmazható módszerek – teljesen különböző algoritmust igényelnek, mint például a kényszerpropagálást vagy a lineáris programozást.

A következő klóz akkor tehető sikeresre, ha a három szám kielégíti a háromszög egyenlőtlenségét:

```
háromszög(X, Y, Z) :-  
  X>=0, Y>=0, Z>=0, X+Y>=Z, Y+Z>=X, X+Z>=Y
```

Ha a Prolognak a háromszög (3, 4, 5) lekérdezést adjuk, akkor ez jól működik. Ha viszont azt kérdezzük: háromszög (3, 4, Z), akkor erre nem talál megoldást, mert a  $Z \geq 0$  részcélt a Prolog nem tudja kezelni. A nehézséget az okozza, hogy a Prologban a változóknak két állapot valamelyikében kell lenniük: egy változó szabad vagy egy bizonyos termhez lekötött.

Egy változó lekötése egy termhez a korlátozás egyik speciális fajtájának tekinthető, nevezetesen egy egyenlősékgényszernek. A **korlátozott logikai programozás** (*constraint logic programming*, CLP) lehetővé teszi, hogy a változók *korlátozva* legyenek, ahelyett hogy *lekötöttek* lennének. Egy korlátozott logikai program megoldása a korlátozóknak a tudásbázisból származtatott legspecifikusabb halmaza a lekérdezés változóin. Például a háromszög (3, 4, Z) lekérdezésre a megoldás a  $7 \geq Z \geq 1$  kényszer. A standard logikai programok csak a CLP speciális esetei, amelyekben a megoldás kényszereinek egyenlőségi kényszereknek – azaz lekötéseknek – kell lenniük.

A CLP-rendszerök különféle kényszermegoldó algoritmusokat is tartalmaznak a nyelvben megengedett kényszerek kezelésére. Például egy valós értékű változókon értelmezett, lineáris egyenlőtlenségeket megengedő rendszer tartalmazhat egy lineáris programozási algoritmust ezen kényszerek feloldására. A CLP-rendszerök egy sokkal rugalmasabb megközelítést is adoptálnak a standard logikai programozási lekérdezések megoldására. Például a mélységi, balról jobbra történő visszalépéses keresés helyett használhatják bármelyik ennél hatékonyabb, az 5. fejezetben tárgyalt algoritmust, beleértve a heurisztikus konjunktok sorba rendezését, a visszaugrást, a vágási

halmaz kondicionálást és így tovább. A CLP-rendszerek tehát vegyesen alkalmazzák a kényszerekkielégítési algoritmusok, a logikai programozás és a deduktív adatbázisok elemeit.

A CLP-rendszerek szintén hasznosítják az 5. fejezetben bemutatott különböző kényszерprogramozási keresési optimalizációs módszereket, mint a változók és értékek sorba rendezését, az előrefelé ellenőrzést vagy az intelligens visszalépés módszerét. Számos rendszert úgy terveztek meg, hogy a programozónak nagyobb kontrollja legyen a következetések keresésének a sorrendje felett. Például az MRS nyelv (Genesereth és Smith, 1981; Russell, 1985) lehetővé teszi a programozó számára, hogy **metaszabályokat** (*metarules*) írjon a konjunktok megvizsgálási sorrendjének meghatározására. A felhasználó például írhat egy olyan szabályt, amely azt mondja ki, hogy a legkevesebb változóval rendelkező cél legyen először kipróblálva, vagy írhat tárgyterület-specifikus szabályokat bizonyos predikátumokhoz.

## 9.5. REZOLÚCIÓ

A logikai rendszerek általunk bemutatandó három családjának az utolsó tagja a **rezolúció** (*resolution*) alapul. Láttuk a 7. fejezetben, hogy a propozíciós rezolúció egy cáfolásteljes következetetű folyamat az ítéletlogikában. Ebben az alfejezetben meg fogjuk vizsgálni, hogyan lehet kiterjeszteni a rezolúciót az elsőrendű logikára.

A teljes bizonyítási eljárás létezésének kérdése a matematikusokat érdeklő közvetlenül. Ha egy teljes bizonyítási eljárás megtalálható matematikai állításokra, ebből két dolog következik. Először is, az összes téTEL előfeltételeit mechanikusan elő tudjuk állítani; másodszor, a teljes matematika felépíthető alapxiómák halmazának logikai következményeként. A teljesség kérdésének vizsgálata így a 20. századi matematika néhány legfontosabb eredményének a megszületéséhez vezetett. 1930-ban Kurt Gödel német matematikus bebizonyította az első **teljességi téTEL** (*completeness theorem*) az elsőrendű logikára, megmutatva, hogy minden kikövetkeztetett mondatnak létezik véges bizonyítása. (Gyakorlatban is felhasználható bizonyítási módszert viszont nem találtak egészen addig, amíg J. A. Robinson nem publikálta a rezolúciós algoritmust 1965-ben.) 1931-ben Gödel bebizonyította a még híresebb **nemteljességi téTEL** (*incompleteness theorem*). A téTEL kimondja, hogy egy logikai rendszer, amely tartalmazza az indukció elvét – amely nélkül a diszkrét matematika igen kis része építhető fel – szükségszerűen nem teljes. Ebből az következik, hogy léteznek olyan kikövetkeztetett mondatok, amelyeknek a rendszeren belül nincs véges bizonyítása. Lehet, hogy a tű ott van a metaforikus szénakazalban, de nincs olyan eljárás, amely garantálná, hogy megtaláljuk.

Gödel tétele ellenére, a rezolúcióalapú téTELbizonyításokat széles körben alkalmazták matematikai tételek levezetésére, beleértve néhány olyan téTEL is, amelyre előzőleg nem volt ismert bizonyítás. A téTELbizonyításokat – más alkalmazások mellett – használták például hardvertervezés verifikálására vagy logikailag helyes programok generálására.

## Az elsőrendű logika konjunktív normál formája

Mint az ítéletlogika esetében, az elsőrendű rezolúció is megköveteli, hogy a mondatok **konjunktív normál formában** (**conjunctive normal form**) (CNF) legyenek, tehát minden mondat klózok konjunkciójá, ahol minden egyes klóz literálok diszjunkciójá alkot.<sup>6</sup> A literálok tartalmazhatnak változókat, amelyeket univerzális kvantorral ellátottak feltételezünk. Például a

$$\forall x \text{ Amerikai}(x) \wedge \text{Fegyver}(y) \wedge \text{Elad}(x, y, z) \wedge \text{Ellenséges}(z) \Rightarrow \text{Bűnöző}(x)$$

mondat CNF-formára átalakítva ilyen lesz:

$$\neg \text{Amerikai}(x) \vee \neg \text{Fegyver}(y) \vee \neg \text{Elad}(x, y, z) \vee \neg \text{Ellenséges}(z) \vee \text{Bűnöző}(x)$$

 **Minden elsőrendű logikai mondat átalakítható a következtetés szempontjából egyenértékű CNF-mondattá.** Ami azt jelenti, hogy egy CNF-mondat akkor kielégíthetetlen, ha az eredeti mondat is kielégíthetetlen, ezáltal a CNF-mondatok között ellentmondásokat keresve létre tudunk hozni bizonyításokat.

A CNF-mondatok átalakításának eljárása nagyon hasonló az ítéletlogikai eljáráshez, amelyet a 269. oldalon láthatunk. A legfontosabb különbség abból fakad, hogy ki kell vonnunk a mondatokból az egzisztenciális kvantorokat. Ezt az eljárást a következő mondat lefordításával illusztráljuk: „Mindenkit, aki az összes állatot szereti, valaki szeret”, vagyis:

$$\forall x [\forall y \text{ Állat}(y) \Rightarrow \text{Szereti}(x, y)] \Rightarrow [\exists y \text{ Szereti}(y, x)]$$

A lépések a következők:

- **Az implikációk eliminálása.**

$$\forall x [\neg \forall y \neg \text{Állat}(y) \vee \text{Szereti}(x, y)] \vee [\exists y \text{ Szereti}(y, x)]$$

- **A  $\neg$  befelé mozgatása.** A negált összekötőjelekre vonatkozó már ismert szabályok mellett szükségünk van a negált kvantorokra vonatkozó szabályokra is. Így ezt kapjuk:

$$\neg \forall x p\text{-ből lesz } \exists x \neg p$$

$$\neg \exists x p\text{-ből lesz } \forall x \neg p$$

A mondatunk a következő átalakításokon megy keresztül:

$$\forall x [\exists y \neg (\neg \text{Állat}(y) \vee \text{Szereti}(x, y))] \vee [\exists y \text{ Szereti}(y, x)]$$

$$\forall x [\exists y \neg \neg \text{Állat}(y) \wedge \neg \text{Szereti}(x, y)] \vee [\exists y \text{ Szereti}(y, x)]$$

$$\forall x [\exists y \text{ Állat}(y) \wedge \neg \text{Szereti}(x, y)] \vee [\exists y \text{ Szereti}(y, x)]$$

Figyeljük meg, hogyan alakítottuk át az univerzális kvantort ( $\forall y$ ) az implikáció premisszájában egzisztenciális kvantorrá. A mondatot most fgy lehet olvasni „Vagy van olyan állat, amelyet  $x$  nem szeret, vagy (ha ez nem áll fenn) valaki szereti  $x$ -et.” Világos, hogy az eredeti mondat jelentését megőriztük.

<sup>6</sup> Egy klózt implikációként is reprezentálhatunk a bal oldali atomok konjunkciójával és a jobb oldali atomok diszjunkciójával, mint azt a 7.12. feladatban láthatjuk. Ezt a formát, amelyet Kowalski-formának (Kowalski form) szoktak nevezni, gyakran sokkal könnyebb olvasni, ha az implikációk fráza jobbról balra történik (Kowalski, 1979b).

- A változók átnevezése.** Az olyan mondatokban, mint a  $(\forall x \ P(x)) \vee (\exists x \ Q(x))$ , amelyek ugyanazt a változó nevet kétszer is használják, változtassuk meg az egyik változó nevét. Ezzel elkerüljük, hogy később keveredés lehessen, amikor elhagyjuk a kvantorokat. Így ezt kapjuk:

$$\forall x \ [\exists y \ \text{Állat}(y) \wedge \neg \text{Szereti}(x, y)] \vee [\exists z \ \text{Szereti}(z, x)]$$

- Skolemizáció.** A skolemizáció (skolemization) az az eljárás, amelynek során eliminációval eltávolítjuk az egzisztenciális kvantorokat. Egyszerű esetben ez megegyezik a 9.1. alfejezetben található egzisztenciális példányosítási szabállyal: fordítsuk le a  $\exists x \ P(x)$ -et  $P(A)$ -ra, ahol az  $A$  egy új konstans. Ha ezt a szabályt alkalmazzuk a mintamondatunkra, akkor azonban ezt kapjuk:

$$\forall x \ [\text{Állat}(A) \wedge \neg \text{Szereti}(x, A)] \vee \text{Szereti}(B, x)$$

amelynek teljesen téves a jelentése: azt mondja ki, hogy mindenki vagy nem tud szeretni egy bizonyos  $A$  állatot, vagy egy bizonyos  $B$  entitás szereti őt. Valójában, az eredeti mondatunk lehetővé teszi minden egyes személy számára, hogy ne szeressen egy másik állatot, vagy egy másik személy szeresse őt. Így tehát, azt szeretnénk, hogy a Skolem entitások az  $x$ -től függjenek:

$$\forall x \ [\text{Állat}(F(x)) \wedge \neg \text{Szereti}(x, F(x))] \vee \text{Szereti}(G(x), x)$$

Itt az  $F$  és a  $G$  **Skolem-függvények**. Az általános szabály az, hogy a Skolem-függvények argumentumai minden univerzális kvantorral ellátott változók, amelyeknek a hatókörében az egzisztenciális kvantor értelmezett. Hasonlóan az Egzisztenciális Példányosításhoz, a skolemizált mondat is pontosan akkor kielégíthető, ha az eredeti mondat is kielégíthető.

- Az univerzális kvantorok elhagyása.** Ezen a ponton minden megmaradt változó univerzális kvantorral van ellátva. Mi több, a mondat egyenértékű azzal a mondattal, amelyben az összes univerzális kvantort balra rendezték. Így tehát most már elhagyhatjuk az univerzális kvantorokat:

$$[\text{Állat}(F(x)) \wedge \neg \text{Szereti}(x, F(x))] \vee \text{Szereti}(G(x), x)$$

- Az  $\wedge$  elosztása a  $\vee$  felett.**

$$[\text{Állat}(F(x)) \vee \text{Szereti}(G(x), x)] \wedge [\neg \text{Szereti}(x, F(x)) \vee \text{Szereti}(G(x), x)]$$

Ennek a lépésnek az elvégzéséhez szükség lehet az egymásba ágyazott konjunkciók és diszjunkciók kibontására is.

A mondat most CNF formájú, és két klózból áll. Eléggé olvashatatlan lett. (Némiképpen segíthet, ha megmagyarázzuk, hogy az  $F(x)$  függvény egy, az  $x$  személy által potenciálisan nem szeretett állatra vonatkozik, míg a  $G(x)$  olyan valakire utal, aki szeretheti  $x$ -et.) Szerencsére, nekünk ritkán kell CNF-mondatokat vizsgálnunk – a fordítási folyamat könnyen automatizálható.

## A rezolúciós következtetési szabály

Az elsőrendű klózokra vonatkozó következtetési szabály egyszerűen a 268. oldalon megadott propozíciós rezolúciós szabály kiterjesztett változata. Két klóz, amelyek standardizálva vannak, tehát nem tartalmaznak azonos változókat, akkor rezolválható, ha tartalmaznak komplementens literálokat. Az ítéletlogikai literálok akkor komplementek, ha az egyik a negációja a másiknak; míg az elsőrendű literálok akkor komplementek, ha az egyik *egyesül* a másik negációjával. Így ezt kapjuk:

$$\frac{\ell_1 \vee \dots \vee \ell_k, m_1 \vee \dots \vee m_n}{\text{HELYETTESÍT}(\theta, \ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

ahol az  $\text{EGYESÍT}(\ell_i, \neg m_j) = \theta$ . Például rezolválhatjuk ezt a két klózt:

$$[\text{Állat}(F(x)) \vee \text{Szereti}(G(x), (x))] \text{ és } [\neg \text{Szereti}(u, v) \vee \neg \text{Megöli}(u, v)]$$

a  $\theta = \{u/G(x), v/x\}$  egyesítő felhasználásával a komplementens literálokat, a  $\text{Szereti}(G(x), x)$ -t és a  $\neg \text{Szereti}(u, v)$ -t eliminálva kapjuk meg a **rezolvens (resolvent)** klózt:

$$[\text{Állat}(F(x)) \vee \neg \text{Megöli}(G(x), x)]$$

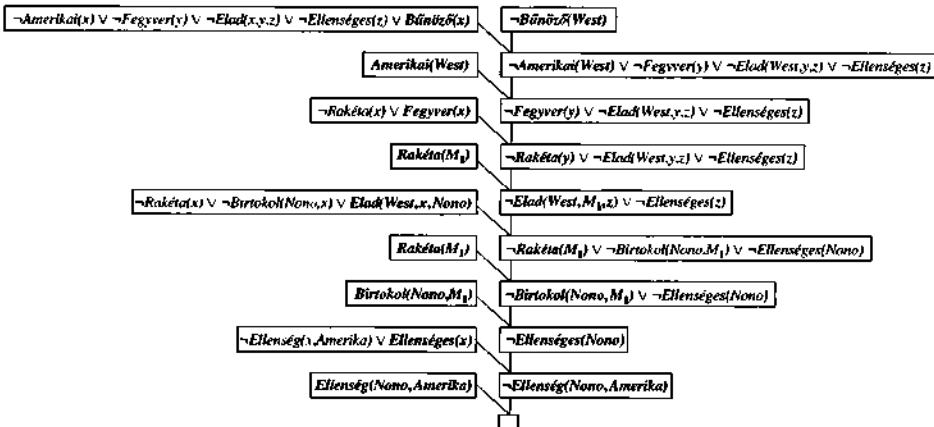
A bermutatott szabály neve **bináris rezolúciós (binary resolution)** szabály, mert pontosan két literált old fel. A bináris rezolúciós szabály önmagában nem eredményez egy teljes következtetési folyamatot. A teljes rezolúciós szabály literálok részhalmazait rezolválja minden egyes egyesíthető klózban. Egy alternatív megközelítés a faktorálásnak (**factoring**), a felesleges literálok eltávolításának a kiterjesztése az elsőrendű logikára. A propozíciós faktorálás két literált egyére redukál, ha azok *azonosak*. Az elsőrendű faktorálás két literált egyére redukál, ha azok *egyesíthetők*. Az egyesítőt a teljes klózra kell alkalmazni. A bináris rezolúció és a faktorálás kombinációja már teljes eljárást eredményez.

## Példabizonyítások

A rezolúció a  $TB \models \alpha$  állítást úgy igazolja, hogy bebizonyítja, hogy a  $TB \wedge \neg \alpha$  nem kielégíthető, vagyis a bizonyítás az üres klóz származtatásával történik. Az algoritmikus megközelítés megegyezik az ítéletlogikában lévővel, amelyet a 7.12. ábrán már bemutattunk, így ezt itt most nem ismétljük meg. Ehelyett inkább két példabizonyítást adunk meg. Az első a bőntény példa a 9.3. alfejezetből. A mondatok CNF-ben a következők:

$$\begin{aligned} &\neg \text{Amerikai}(x) \vee \neg \text{Fegyver}(y) \vee \neg \text{Elad}(x, z, y) \vee \neg \text{Ellenséges}(z) \vee \text{Bűnöző}(x) \\ &\neg \text{Rakéta}(x) \vee \neg \text{Birtokol}(Nono, x) \vee \text{Elad}(West, x, Nono) \\ &\neg \text{Ellenség}(x, Amerika) \vee \text{Ellenséges}(x) \\ &\neg \text{Rakéta}(x) \vee \text{Fegyver}(x) \\ &\text{Birtokol}(Nono, M_1) \quad \text{Rakéta}(M_1) \\ &\text{Amerikai}(West) \quad \text{Ellenség}(Nono, Amerika) \end{aligned}$$

Hozzáadjuk a mondatok halmazához a negált célt is  $\neg \text{Bűnöző}(West)$ . A rezolúciós bizonyítást a 9.11. ábra mutatja be. Figyeljük meg a szerkezetét: a bizonyításnak egy



9.11. ábra. Egy rezolúciós bizonyítás arra, hogy West bűnöző

egyszerű „gerince” van, amely a célklózzal kezdődik, és folyamatosan rezolvál a tudásbázisból származó klózokkal, mindaddig, amíg az üres klózt nem generálja. Ez jellemző a Horn-klózokat tartalmazó tudásbázisokon végzett rezolúcióra. Valójában a fő gerinc mentén található klózok *pontosan* megfelelnek a 9.6. ábrán látható hátrafelé láncolási algoritmusban a *célváltozók* egymást követő értékeinek. Ez azért van így, mert minden azt a klózt választjuk ki rezolválásra, amelynek a pozitív literálja egyesíthető a gerincen lévő „aktuális” klóz bal szélén elhelyezkedő literáljával, és pontosan ez történik a hátrafelé láncolásban is. Így tehát a hátrafelé láncolás valójában a rezolúció egy speciális esete, egy különleges vezérlési stratégiával, amely meghatározza, hogy melyik rezolúciót hajtsuk végre legközelebb.

A második példánk kihasználja a skolemizációt, és nem határozott klózokat is tartalmaz. Ez valamivel bonyolultabb bizonyítási struktúrát eredményez. Természetes nyelven megfogalmazva a probléma a következő:

Mindenkit, aki az összes állatot szereti, szeret valaki.

Bárkit, aki megöl egy állatot, senki sem szeret.

Jankó szereti az összes állatot.

Vagy Jankó, vagy a Kíváncsiság ölte meg a macskát, akinek a neve Tuna.

A Kíváncsiság ölte meg a macskát?

Először is leírjuk az eredeti mondatokat, némi háttérteudást, és a G negált célt az elsőrendű logikában:

- $\forall x [\forall y \text{ Állat}(y) \Rightarrow \text{Szereti}(x, y)] \Rightarrow [\exists y \text{ Szereti}(y, x)]$
- $\forall x [\exists y \text{ Állat}(y) \wedge \text{Megöli}(x, y)] \Rightarrow [\forall z \neg \text{Szereti}(z, x)]$
- $\forall x \text{ Állat}(x) \Rightarrow \text{Szereti}(\text{Jankó}, x)$
- $\text{Megöli}(\text{Jankó}, \text{Tuna}) \vee \text{Megöli}(\text{Kíváncsiság}, \text{Tuna})$
- $\text{Macskák}(Tuna)$
- $\forall x \text{ Macskák}(x) \Rightarrow \text{Állat}(x)$
- $\neg \text{Megöli}(\text{Kíváncsiság}, \text{Tuna})$

Most pedig alkalmazzuk az átalakítási procedírát, hogy minden egyes mondatot CNF-be konvertáljunk:

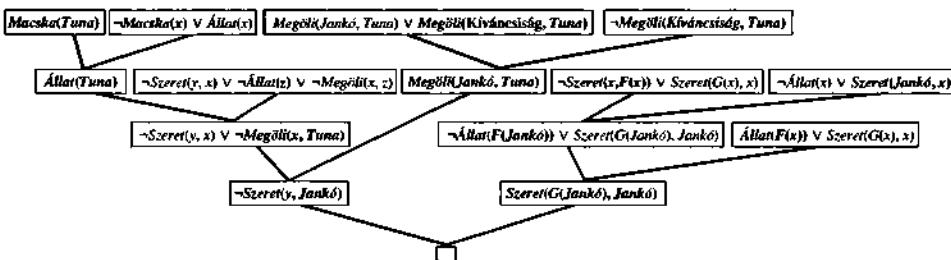
- A1.  $\text{Állat}(F(x)) \vee \text{Szereti}(G(x), x)$
- A2.  $\neg\text{Szereti}(x, F(x)) \vee \text{Szereti}(G(x), x)$
- B.  $\neg\text{Állat}(y) \vee \neg\text{Megöli}(x, y) \vee \neg\text{Szereti}(z, x)$
- C.  $\neg\text{Állat}(x) \vee \text{Szereti}(\text{Jankó}, x)$
- D.  $\text{Megöli}(\text{Jankó}, \text{Tuna}) \vee \text{Megöli}(\text{Kíváncsiság}, \text{Tuna})$
- E.  $\text{Macska}(\text{Tuna})$
- F.  $\neg\text{Macska}(x) \vee \text{Állat}(x)$
- G.  $\neg\text{Megöli}(\text{Kíváncsiság}, \text{Tuna})$

A 9.12. ábrán látható a rezolúciós bizonyítása annak, hogy a Kíváncsiság ölte meg a macskát. Magyarul a bizonyítást így írhatjuk át:

Tegyük fel, hogy a Kíváncsiság nem ölte meg Tunát. Tudjuk, hogy vagy Jankó, vagy a Kíváncsiság tette; tehát biztosan Jankó tehette. Mármost, Tuna egy macska, és a macskák állatok, tehát Tuna egy állat. Mivel bárkit, aki megöl egy állatot, senki sem szeret, tudjuk, hogy Jankót nem szereti senki. Másrészt, Jankó minden állatot szeret, tehát valaki szereti őt, tehát ez az ellentmondás áll fenn. Így tehát a Kíváncsiság ölte meg a macskát.

A bizonyítás képes megválaszolni a „A Kíváncsiság ölte meg a macskát?” kérdést, de gyakran ennél általánosabb kérdéseket akarunk felenni, mint például: „Ki ölte meg a macskát?” A rezolúció meg tudja válaszolni ezt is, de ennek a válasznak a levezetése egy kicsit több munkát igényel. A célállítás a  $\exists w \text{ Megöli}(w, \text{Tuna})$ , amely negáláskor CNF-ben  $\neg\text{Megöli}(w, \text{Tuna})$  lesz. Megismételve a bizonyítást a 9.12. ábrán az új negált céllal, hasonló bizonyítási fát kapunk, de a következő helyettesítéssel: { $w/\text{Kíváncsiság}$ } az egyik lépében. Így ebben az esetben, annak a kiderítése, hogy ki ölte meg a macskát, nem áll másból, mint a lekérdezés változóiban lévő lekötések nyomon követése a bizonyításban.

Sajnos, a rezolúció csak **nem konstruktív bizonyítást** (*nonconstructive proofs*) tud előállítani az egzisztenciális célmondatokra. Például a  $\neg\text{Megöli}(w, \text{Tuna})$  rezolválható a  $\text{Megöli}(\text{Jankó}, \text{Tuna}) \vee \text{Megöli}(\text{Kíváncsiság}, \text{Tuna})$ -val, hogy a  $\text{Megöli}(\text{Jankó}, \text{Tuna})$  mondatot megkapjuk, amely megint rezolválható ezzel:  $\neg\text{Megöli}(w, \text{Tuna})$ , és így kapjuk meg az üres klózit. Figyeljük meg, hogy ebben a bizonyításban a  $w$ -nek két különböző lekötése van. A rezolúció megmutatja azt, hogy igen, valaki megölte Tunát – vagy Jankó, vagy a Kíváncsiság. Ez nem túl meglepő! Egy lehetséges megoldás korlátozni a rezolúciós lépéseket úgy, hogy a kérdés változóinak csak egy lekötésük lehessen egy



**9.12. ábra.** Egy rezolúciós bizonyítása annak, hogy a Kíváncsiság ölte meg a macskát. Figyeljük meg a faktorálás használatát a  $\text{Szereti}(G(\text{Jankó}), \text{Jankó})$  klóz származtatásánál.

adott bizonyításban, majd képesnek kell lennünk arra, hogy visszakövessük a lehetséges lekötéseket. Egy másik megoldás egy speciális válasz literál (**answer literal**) hozzáadása a negált célhoz, amelyből ez lesz:  $\neg\text{Megöli}(w, \text{Tuna}) \vee \text{Válasz}(w)$ . Ilyenkor a rezolúciós folyamat minden generál egy pontosan egy literált tartalmazó választ, amikor egy ilyan klóz generálódik. A 9.12. ábrán látható bizonyításra ez a Válasz(Kíváncsiság). A nem konstruktív bizonyítás ezt a klózt generálná: Válasz(Kíváncsiság)  $\vee$  Válasz(Jankó), amely nem ad számunkra választ.

## A rezolúció teljessége

Ebben a részben bebizonyítjuk a rezolúció teljességét. A fejezet elővasását nyugodtan kihagyhatja bárki, aki megelégszik azzal, hogy elhiszi ezt az állítást.

Be fogjuk mutatni, hogy a rezolúció **cáfolástelejes** (**refutation-complete**), ami azt jelenti, hogy *ha* egy mondathalmaz kielégíthetetlen, akkor a rezolúció minden képes vezetni egy ellentmondást. A rezolúció nem alkalmazható arra, hogy mondatok egy halmazának összes logikai következményét generálja, de használható arra, hogy megmondjuk, hogy egy adott mondat következménye-e a mondathalmaznak. Így alkalmazható egy adott kérdésre adható összes válasz megtalálására, felhasználva a korábban a fejezetben már bemutatott célnegálás módszert.

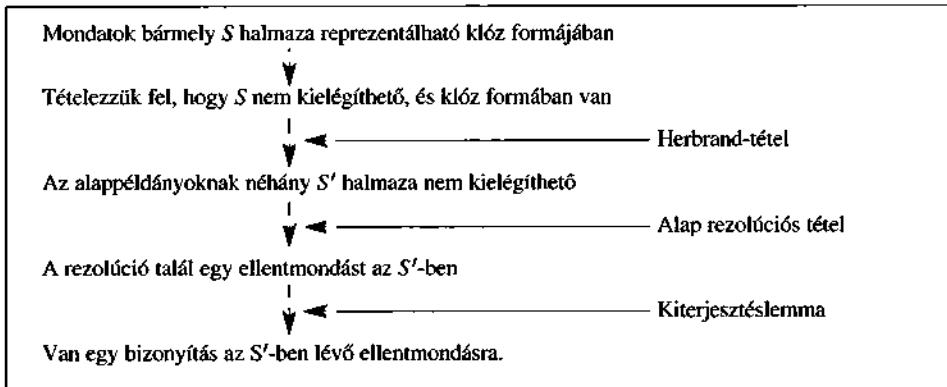
Tényleg tekintjük, hogy minden (egyenlőséget nem tartalmazó) elsőrendű logikai mondat átírható CNF-klózok halmazára. Ez a mondat alakján végzett indukciós bizonyítással igazolható, az atomi mondatból, mint alapesetből kiindulva (Davis és Putnam, 1960). Célunk tehát bizonyítani a következőt: *ha S egy kielégíthetetlen klózhalmaz, akkor az S-en elvégzett véges számú rezoluciós lépés alkalmazása ellentmondáshoz vezet*.

Bizonyításunk váza követi Robinson eredeti bizonyítását, néhány Geneserethtől és Nilssontól származó egyszerűsítés hozzáadásával (Genesereth és Nilsson, 1987). A 9.13. ábra mutatja be a bizonyítás alapvető struktúráját, ami a következő:

1. Először megvizsgáljuk, hogy ha S kielégíthetetlen, akkor létezik S klózai között bizonyos *alappéldányoknak* egy olyan halmaza, ami szintén kielégíthetetlen. (Herbrand-tétel).
2. Ekkor az alap rezoluciós tételezhet (ground resolution theorem) fordulunk, amelyet a 7. fejezetben adtunk meg, és amely kijelenti, hogy a propoziciós rezolúció teljes az alapmondatokra.
3. Ezután alkalmazzuk a *kiterjesztéslemmát* (lifting lemma) annak megmutatására, hogy bármely propoziciós rezoluciós bizonyításhoz, amely alapmondatok halmazát használja, található olyan elsőrendű rezoluciós bizonyítás, amely azokat az elsőrendű logikai mondatokat használja, amelyekből az alapmondatokat megkaptuk.

Az első lépés elvégzéséhez szükségünk lesz három új fogalomra:

- **Herbrand-univerzum:** Ha S a klózok egy halmaza, akkor a  $H_S$  a Herbrand-univerzuma S-nek, vagyis az összes alaptermből álló halmaz, amelyet létrehozhatunk a következőkből:
  - (a) Az S függvény szimbólumai, ha vannak ilyenek.
  - (b) Az S konstansszimbólumai, ha vannak ilyenek. Ha nincs ilyen, akkor az A konstans szimbólum.



**9.13. ábra.** A rezolúcióteljesség bizonyításának a szerkezete

Például, ha az  $S$  csak a  $\neg P(x, F(x, A)) \wedge \neg Q(x, A) \vee R(x, B)$  klózt tartalmazza, akkor a  $H_S$  az alapmondatok következő végtelen halmaza:

$$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \dots\}$$

- **Telítődés:** Ha  $S$  a klózok egy halmaza, és  $P$  az alaptermek egy halmaza, és  $P(S)$  az  $S$  telített halmaza  $P$ -re vonatkoztatva, akkor  $P(S)$  az összes alapkloz halmaza, amelyet úgy kapunk meg, hogy alkalmazzuk az összes lehetséges konzisztens helyettesítését a  $P$ -beli alaptermeknek az  $S$ -beli változókkal.
- **Herbrand-bázis:** Egy  $S$  klózhalmaznak a hozzá tartozó Herbrand-univerzumra vonatkozó telítődését  $S$  Herbrand-bázisának nevezzük, és  $H_S(S)$ -sel jelöljük. Például, ha  $S$  csak a korábban példaként használt klózt tartalmazza, akkor a  $H_S(S)$  a következő végtelen klózhalmaz :

$$\begin{aligned} &\{\neg P(A, F(A, A)) \vee \neg Q(A, A) \vee R(A, B), \\ &\quad \neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B), \\ &\quad \neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B) \\ &\quad \neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \dots\} \end{aligned}$$

Ezek a definíciók lehetővé teszik, hogy a **Herbrand-tétel** (Herbrand's theorem) egyik formáját állítsuk (Herbrand, 1930):

Ha a klózok egy  $S$  halmaza nem kielégíthető, akkor létezik egy véges  $H_S(S)$  részhalmaz, amely szintén nem kielégíthető.

Legyen  $S'$  ez a véges részhalmaza az alapmondatoknak. Mármost, alkalmazhatjuk az alap rezolúciós téttel (lásd 272. oldal), hogy kimutassuk, hogy a **rezolúciós! lezárt (resolution! closure)  $RC(S')$**  tartalmazza az üres klózt. Ez azt jelenti, hogy a propozíciós rezolúció futtatása a teljesséig az  $S'$ -en egy ellentmondást fog eredményezni.

## Gödel nemteljesség tétele

A matematikai indukciós sémákkal (**mathematical induction schema**) kibővíte az elsőrendű logikát, Gödel nemteljesség tétele (**incompleteness theorem**) megmutatja, hogy léteznek olyan igaz aritmetikai mondatok, amelyek nem bizonyíthatók.

A nemteljesség elmélet igazolása mintegy 30 oldalt kívánna, így túlmutat ennek a könyvnek a keretein, de a bizonyítás lényegét felvázoljuk a következőkben. Először a számok logikai elméletét vezetjük be. Ebben az elméletben egyetlen konstans létezik, a 0, és egyetlen függvény, az  $S$  (az utód-függvény). Az így létrehozott modellben  $S(0)$  fejezi ki az 1-et,  $S(S(0))$  jelöli a 2-t és így tovább. A nyelv tehát alkalmas az összes természetes szám kifejezésére. A nyelv szótára tartalmazza ezen-kívül a +, a  $\times$  és az  $\exp$  (exponenciális) függvényszimbólumokat és a logikai összekötőjelek és a kvantorok szokásos halmazát. Először figyeljük meg, hogy a nyelv mondatai megszámozhatók. (Definiálunk alfabetikus sorrendet a szimbólumok között, és rendezzük alfabetikus sorrendbe a nyelv 1, 2, ...,  $n$ , ... hosszúságú mondatait.) Így minden  $\alpha$  mondathoz hozzárendelhetünk egy egyedi  $\# \alpha$  természetes számot, a **Gödel-számot** (**Gödel number**). Fontos tehát, hogy a számelmélet minden saját mondatához tartalmaz egy nevet. Hasonlóan megszámozhatunk minden egyes  $P$  bizonyítást egy  $G(P)$  Gödel-számmal, mivel egy bizonyítás nem más, mint mondatok véges sorozata.

Most tegyük fel, hogy adott egy  $A$  halmaz, amely a természetes számokról igaz állításokat megfogalmazó mondatokat tartalmaz. Mivel az  $A$  halmaz megadható egész számoknak egy adott halmazával (az  $A$  halmazban található mondatok Gödel-számaival), ezért létre tudunk hozni a definiált nyelvben egy  $\alpha(j, A)$  mondatot, amely a következő állítás:

$\forall i$  i nem a Gödel-száma a  $j$  Gödel-számú mondat olyan bizonyításának, amely bizonyítás csak  $A$ -beli premisszákat használ.

Legyen a  $\sigma$  az  $\alpha(\# \sigma, A)$  mondat, vagyis egy olyan mondat, amely kifejezi önmagának a bizonyíthatatlanságát  $A$ -ból. (Igazolható, hogy ilyen mondat minden létezik, de nem egyszerű ezt beláttni.)

Kövessük végig a következő tügés érvét. Tegyük fel, hogy  $\sigma$  bizonyítható  $A$ -ból, de akkor  $\sigma$  hamis, mivel  $\sigma$  épp azt mondja ki, hogy nem igazolható. Ebben az esetben viszont létezik egy hamis mondat, ami bizonyítható  $A$ -ból, így  $A$  nem tartalmazhatna csak igaz mondatokat – ez az előzetes feltételezésünkkel ellentétes. Tehát  $\sigma$  nem bizonyítható  $A$ -ból. Ez viszont éppen az, amit  $\sigma$  állít saját magáról, tehát  $\sigma$  egy igaz mondat.

Tehát megmutattuk (29 és fél oldalnyi levezetést átugorva), hogy bármely a számelméletben definiálható igaz mondatokat tartalmazó halmaz esetében, bizonyos alapaxiómák feltételezése mellett, léteznek olyan igaz mondatok, amelyek *nem* bizonyíthatók az axiómákból. Ez az eredmény, más következmények mellett azt jelenti, hogy a matematika *bármely axiómarendszerében* megfogalmazható olyan téTEL, amely *nem* bizonyítható az adott rendszer axiómáiból. Ez a matematikának egy igen fontos eredménye. A Gödel-tétel hatását a mesterséges intelligencia területére sokan vitatták, köztük maga Gödel is. A 26. fejezetben visszatérünk erre a kérdésre.

Most, hogy megállapítottuk, hogy minden van egy rezoluciós bizonyítás, amely magában foglal néhány véges részhalmazt a Herbrand  $S$  bázisból, a következő lépés annak megmutatása, hogy létezik egy rezoluciós bizonyítás, amely magának az  $S$ -nek a klózait tartalmazza, amelyek nem szükségszerűen alapkiszámok. Azzal kezdjük, hogy megvizsgáljuk a rezoluciós szabály egy egyszerű alkalmazását. Robinson alaplemmája magába foglalja a következő tényt:

Legyen  $C_1$  és  $C_2$  két klóz, amelyeknek nincsenek közös változói. Legyenek  $C'_1$  és  $C'_2$  az alappéldányai  $C_1$ -nek és  $C_2$ -nek. Ha  $C'$  rezolvense  $C'_1$ -nek és  $C'_2$ -nek, akkor létezik egy olyan  $C$  klóz, amelyik (1) rezolvense  $C_1$ -nek és  $C_2$ -nek és (2)  $C'$  alappéldánya  $C$ -nek.

Ezt **kiterjesztéslemmának** (**lifting lemma**) nevezik, mivel kiterjeszti a bizonyítási lépést az alap klózokról az általános elsőrendű klózokra. A kiterjesztési lemma bizonyí-

tásához Robinsonnak szüksége volt az egyesítés módszerének megalkotására és a legáltalánosabb egyesítő tulajdonságainak vezetésére is. A bizonyítás áttekintése helyett a következőben illusztráljuk a lemmát:

$$\begin{aligned} C_1 &= \neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B) \\ C_2 &= \neg N(G(y), z) \vee P(H(y), z) \\ C'_1 &= \neg P(H(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C'_2 &= \neg N(G(B), F(H(B), A)) \vee P(H(B), F(H(B), A)) \\ C' &= \neg N(G(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B) \\ C &= \neg N(G(y), F(H(y), A)) \vee \neg Q(H(y), A) \vee R(H(y), B) \end{aligned}$$

Látható, hogy  $C'$  valóban  $C$  alaptermje. Általánosságban ahhoz, hogy  $C'_1$ -nek és  $C'_2$ -nek legyen rezolvense, úgy kell létrehozni őket, hogy először a  $C_1$ -beli és a  $C_2$ -beli komplement literálok legáltalánosabb egyesítőjét alkalmazzuk  $C_1$ -re és  $C_2$ -re. A kiterjesztési lemma felhasználásával könnyű hasonló állításokat vezetni a rezoluciós szabály alkalmazásának bármely sorozatára:

Bármely az  $S'$  lezárásához tartozó  $C'$  klózhoz létezik egy  $S$  lezárásában levő  $C$  klóz, amelyre a  $C'$  klóz a  $C$  klóz alappéldánya, és a  $C$  vezetése azonos hosszúságú a  $C'$  vezetésével.

Ebből a tényből következik, hogy ha  $S'$  lezárásában megjelenik az üres klóz, akkor ez szintén megtalálható az  $S$  rezoluciós lezárásában. Ez azért van, mivel az üres klóz semmilyen más klóznak sem alappéldánya. Összefoglalva: megmutattuk, hogyha  $S$  nem kielégíthető, akkor létezik az üres klóznak egy véges méretű rezoluciós szabályt alkalmazó vezetése.

Az elmelet bizonyításában az alapklózokról az elsőrendű klózokra történő kiterjesztés az állítás erejének igen jelentős növelése. Ez abból következik, hogy az elsőrendű bizonyításban már csak annyiszor kell a változókat helyettesíteni, amennyiszer ez a bizonyításhoz szükséges, míg az alapklózmódszereknél szükséges volt nagyszámú önkényes példányosítást megvizsgálni.

## Az egyenlőség kezelése

Az ebben a fejezetben az eddig leírt következtetési módszerek közül egyik sem foglalkozott az egyenlőséggel. Hárrom jól elkülöníthető megközelítést alkalmazhatunk. Az első az, hogy axiómákkal látjuk el az egyenlőséget – vagyis leírunk mondatokat az egyenlőségi relációról a tudásbázisban. El kell mondanunk, hogy az egyenlőség reflexív, szimmetrikus, tranzitív, és hogy az egyenlők helyettesíthetők egyenlőkkel bármely predikátumban vagy függvényben. Ezért van szükségünk három alapaxiómára, majd ezután egy további minden egyes predikátumhoz vagy függvényhez:

$$\begin{aligned} \forall x \quad x = x \\ \forall x, y \quad x = y \Rightarrow y = x \\ \forall x, y, z \quad x = y \wedge y = z \Rightarrow x = z \\ \forall x, y \quad x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\ \forall x, y \quad x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\ \vdots \end{aligned}$$

$$\begin{aligned} \forall w, x, y, z \quad w = y \wedge x = z &\Rightarrow (F_1(w, x) = F_1(y, z)) \\ \forall w, x, y, z \quad w = y \wedge x = z &\Rightarrow (F_2(w, x) = F_2(y, z)) \\ \vdots \end{aligned}$$

Mikor ezek a mondatok adottak, egy standard következtetési eljárás, mint amilyen a rezolúció elvégezhet olyan feladatokat, amelyek egyenlőségi következtetést igényelnek, mint például a matematikai egyenletek megoldása.

Az egyenlőség kezelésének másik módja egy további következtetési szabály alkalmazása. A legegyszerűbb szabály, a **dermoduláció** (**demodulation**) vesz egy egységlőket,  $x = y$  és helyettesíti az  $y$ -t bármely termmel, ami  $x$ -szel egyesíthető egy másik klózban. Formálisabban kifejezve ezt kapjuk:

- **Demoduláció.** Bármely  $x, y$  és  $z$  termekre, ahol  $\text{EGYESÍT}(x, z) = \theta$ , és az  $m_n[z]$  egy literál, ami tartalmazza a  $z$ -t.

$$\frac{x = y, \quad m_1 \vee \dots \vee m_n[z]}{m_1 \vee \dots \vee m_n[\text{HELYETTESÍT}(\theta, y)]}$$

A demodulációt jellemzően arra használják, hogy leegyszerűsítse állítások kollekcióit használó kifejezéseket, mint például  $x + 0 = x$ ,  $x^1 = x$  és így tovább. A szabályt ki lehet terjeszteni, hogy olyan nem egység klózokkal is tudjon foglalkozni, amelyekben egy egyenlőségi literál megjelenik.

- **Paramoduláció (paramodulation).** Bármely  $x, y$  és  $z$  termre, ahol az  $\text{EGYESÍT}(x, z) = \theta$ .

$$\frac{\ell_1 \vee \dots \vee \ell_k \vee x = y, \quad m_1 \vee \dots \vee m_n[z]}{[\text{HELYETTESÍT}(\theta, \ell_1 \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_n[y])]}$$

A demodulációval ellentétben a paramoduláció egy teljes következtetési eljárást eredményez az egyenlőséggel rendelkező elsőrendű logikában.

Egy harmadik megközelítés az egyenlőségi következtetést teljes mértékben egy kiterjesztett egyesítési algoritmuson belül kezeli. Ez azt jelenti, hogy a termek egyesíthetők, ha *bizonyíthatóan* egyenlök egy bizonyos helyettesítés alatt, ahol a „bizonyíthatóan” lehetővé tesz egy bizonyos mennyiségi egyenlőségi következtetést. Például, azok a termek, hogy  $1 + 2$  és  $2 + 1$  normális esetben nem egyesíthetők, de egy egyesítési algoritmus, amely ismeri, hogy  $x + y = y + x$ , tudná őket egyesíteni az üres helyettesítéssel. Az ilyen fajta **egyenleti egyesítés** (**equational unification**) elvégezhető hatékony algoritmusokkal, amelyeket arra terveztek, hogy a felhasznált bizonyos axiómákat használja (kommutativitás, asszociativitás és így tovább), ahelyett hogy explicit következtéseket tenne ugyanazokkal az axiómákkal. A tételekbizonyítások, amelyek ezt a technikát használják, szoros kapcsolatban állnak a korlátozott logikai programozási rendszerekkel, amelyeket a 9.4. alfejezetben írtunk le.

## Rezolúciós stratégiák

Tudjuk, hogy a rezolúció ismételt alkalmazása megtalálja a bizonyítást, ha létezik. Ebben a részben olyan stratégiákat tekintünk át, amelyek *hatékonyan* segítenek megtalálni a bizonyításokat.

## Egyéspreferencia

Ez a stratégia előnybe helyezi azoknak a mondatoknak az alkalmazását, amelyek egyetlen literált tartalmaznak (szokták ezeket **egységlóznak** (**unit clause**) is nevezni). A stratégia alapötlete az, hogy próbálunk meg létrehozni egy üres klózt, így tehát jó ötlet lehet előnyben részesíteni az olyan következtetéseket, amelyek rövidebb klózokat hoznak létre. Egy egységmondat feloldása (mint amilyen a  $P$ ) egy bármely más mondattal (mint amilyen a  $\neg P \vee \neg Q \vee R$ ), mindenig egy olyan klózt eredményez (ebben az esetben:  $\neg Q \vee R$ ), amely rövidebb lesz, mint a másik klóz. Amikor az egységpreferencia-stratégiát először alkalmazták az ítéletkalkulusban, akkor ez drámai gyorsulást eredményezett, megvalósíthatóvá téve olyan bizonyítások elvégzését, amelyek a preferenciák meghatározása nélkül nem voltak kivitelezhetők. Az egységpreferencia azonban önmagában nem csökkenti le elégé az elágazások számát a közepes méretű problémákban azzal, hogy ezek kezelhetők legyenek rezolúcióval. Mindenesetre a módszer egy hasznos heurisztika, amit jól lehet kombinálni más stratégiákkal.

Az **egységrezolúció** (**unit resolution**) a rezolúció egy korlátozott formája, amelyben minden rezolúciós lépésnek tartalmaznia kell egy egységlózot. Az egységrezolúció általában nem teljes, de a Horn-tudásbázisokban teljes. Az egységrezolúciós bizonyítások a Horn-tudásbázisokon emlékeztetnek az előrefelé láncolásra.

## Támogató halmaz

Hasznosak azok a módszerek, amelyek megpróbálják meghatározni, hogy melyik rezolúciót érdemes először elvégezni, de még hatékonyabbak, ha megpróbáljuk kizártani a lehetséges rezolúciók egy csoportját is. A **támogató halmaz** (**set of support**) stratégia pontosan ezt teszi. Az eljárás mondatok egy halmazának kiválasztásával kezdődik, amelyet támogató halmaznak nevezünk. minden rezolúció egy támogató halmazbeli elemet és egy másik, nem halmazbeli elemet kombinál össze, és hozzáadja a rezolvenst a támogató halmazhoz. Ha a támogató halmaz viszonylag kicsi a tudásbázishoz képest, akkor a módszer alkalmazása jelentős mértékben lecsökkenti a keresési teret.

Óvatosan kell alkalmaznunk a megközelítést, mivel az algoritmus nem lesz teljes, ha a támogató halmazt rosszul választjuk meg. Ha az  $S$  támogató halmazt úgy választjuk meg, hogy a maradék mondatok együttesen kielégíthetők, akkor a támogató halmaz stratégiáját alkalmazó rezolúció teljes. Elterjedt módszer, hogy a negált lekérdezés mondatot használjuk támogató halmazzként, azt feltételezve, hogy az eredeti tudásbázis konziszens. (Végül is, ha a tudásbázis nem konziszens, akkor a lekérdezésből következtethető tény is jelentés nélküli.) A támogató halmaz stratégiának további előnye, hogy a létrehozott bizonyítási fák célorientáltak, fgy az olvasók számára könnyen érthetők.

## Bemeneti rezolúció

A **bemeneti rezolúció** (**input resolution**) stratégiában minden rezolúció egy (tudásbázisbeli vagy lekérdező-) mondatot kombinál más mondatokkal. A 9.11. ábrán látható bizonyítás csak bemeneti rezolúciót használ. Ezt az ábrákon sok oldalkapcsolattal rendelkező

vonal jellemzi, ahol a kapcsolódásokon egyedi mondatok találhatók. Egyértelmű, hogy az ilyen alakú bizonyítási fák mérete kisebb, mint bármely más bizonyítási fáé. Horn formájú tudásbázisok esetében a Modus Ponens egyfajta bemeneti rezolúciós stratégia, mivel minden az eredeti  $TB$  egy mondatát kombinálja egy másik mondattal. Így nem meglepő, hogy a bemeneti rezolúció teljes a Horn formájú tudásbázisok esetében, de nem teljes általános esetben. A **lineáris rezolúció** (*linear resolution*) stratégia egy olyan általánosítás, amely megengedi, hogy  $P$  és  $Q$  együtt szerepeljenek a rezolúcióban, ha  $P$  eredeti eleme a  $TB$ -nek, vagy ha  $P$  leszármazottja  $Q$ -nak a bizonyítási fában. A lineáris rezolúció teljes eljárás.

### Bennfoglalás

A **bennfoglalás** (*subsumption*) módszere kizára a keresésből azokat a mondatokat, amelyek benne foglaltatnak (például mert specifikusabbak) más tudásbázisbeli mondatokban. Például ha  $P(x)$  megtalálható a tudásbázisban, akkor nincs értelme hozzáadni  $P(A)$ -t, még kevesebb értelme van hozzáadni  $P(A) \vee Q(B)$ -t. A bennfoglalás segít kis méreten tartani a  $TB$ -t, amely a keresési tér méretének csökkentését eredményezi.

## Tételbizonyítók

A tételbizonyítók (amelyeket automatizált következtetőknek is szoktunk nevezni) két szempontból különböznek a logikai programozási nyelvktől. Először is a legtöbb logikai programozási nyelv csak Horn klózokkal dolgozik, ezzel szemben a tételbizonyítások elfogadják a teljes elsőrendű logikát. Másodszor, a Prolog programok egymásba fűzik a logikát és a kontrollt. Ha a programozó választása az  $A : -B$ ,  $C$ -re esik az  $A : -C$ ,  $B$  helyett, ez a program végrehajtására van hatással. A legtöbb tételbizonyításban a mondatok választott szintaktikai formája nincs hatással a kapott eredményekre. A tételbizonyításoknak is szükségük van az információ kontrolljára, hogy hatékonyan működhessenek, de ezt az információt általában külön tárolják a tudásbázistól. ahelyett, hogy magának a tudásreprezentációnak a részeként jelenne meg. A legtöbb kutatás a tételbizonyítások tárgyában magában foglalja az általában hasznos kontrollstratégiaikat, amelyek a sebességet is növelhetik.

### Egy tételbizonyítás szerkesztése

Ebben a részben leírjuk az OTTER (Organized Techniques for Theoremproving and Effective Research) nevű tételbizonyítót (McCune, 1992), különös figyelmet fordítva a vezérlési stratégiájára. Ahhoz, hogy egy problémát készítsen elő az OTTER számára, a felhasználónak négy részre kell osztania a tudásbázist:

- Egy klózhalmazra, melyet **támogató halmaznak** (*set of support*) (vagy *sos*-nek) nevezünk, és amely definiálja a problémáról szóló legfontosabb tényeket. minden rezolúciós lépés a támogató halmaz egy tagját rezolválja egy másik axiómához képest, tehát a keresést a támogató halmazra koncentráljuk.

- A használható axiómákra (**usable axioms**), amely egy halmaz a támogató halmazon kívül. Ez háttérudást biztosít a problématerületről. A határ megválasztása a probléma részei (tehát ami az *sos*-en belül van) és a háttér (tehát a használható axiómák) között a felhasználó megítélésén múlik.
- Az egyenletek egy halmazára, amelyeket **átírásoknak** (**rewrites**) vagy **demodulátoroknak** (**demodulators**) nevezünk. Habár a demodulátorok egyenletek, minden balról jobbra irányban alkalmazzák őket. Így tehát egy kanonikus formát határoznak meg, amelyben az összes term leegyszerűsödik. Például az  $x + 0 = x$  demodulátor kimondja, hogy minden termet az  $x + 0$  formában az  $x$  termmel kell helyettesíteni.
- Paraméterek és klózok egy halmazára, amelyek meghatározzák a vezérlési stratégiát. A felhasználónak egy heurisztikus funkciót szükséges specifikálni, hogy kontrollálja a keresést, és egy szűrő funkciót, hogy kitöröljön néhány lényegtelen részcélt.

Az OTTER úgy működik, hogy folyamatosan rezolválja a támogatóhalmaz egy elemét az egyik használható axiómával szemben. A Prologgal ellentétben, ez egy „a legjobbat-először” keresési használ. A heurisztikus funkciója megméri minden egyes klóz „súlyát”, ahol a könnyebb klózokat részesíti előnyben. A heurisztika egzakt kiválasztása a felhasználón műlik, de általában a klóz súlyának korrelálnia kell a méretével vagy a nehézségével.

```

procedure OTTER (sos, használható)
  inputs: sos, egy támogató halmaz – a problémát meghatározó klózok (egy globális változó)
          használható, háttérudás, amely potenciálisan releváns a problémával

  repeat
    klóz ← az sos legkönnyebb tagja
    áteszi a klóz-t az sos-ból a használható-ba
    FELDOLGOZ(KÖVETKEZTET(klóz, használható), sos)
  until sos = [] or egy cátolatot talál

function KÖVETKEZTET(klóz, használható) returns klózokat
  rezolvál klózok-at a használható minden egyes tagjával
  return az eredményezett klózokat a FILTER alkalmazása után

procedure FELDOLGOZ(klózok, sos)
  for each klóz in klózok do
    klóz ← EGYSERŰSÍT(klóz)
    egybevon megegyező literálokat
    selejtez klózit, ha az tautológia
    sos ← [klóz|sos]
    if a klóz-nak nincs literálja then egy cátolatot talált
    if a klóz-nak egy literálja van then keres egység cátolatot
  
```

**9.14. ábra.** Az OTTER tételebizonyítás vázlata. A heurisztikus kontroll alkalmazzák a „legkönnyebb” klóz kiválasztására és a FILTER funkcióban, amely kitörli a jelentéktelen klózokat a további vizsgálatból.

Az egységlázokat könnyűként kezeljük; így tehát a keresést az egységszabály-stratégia általánosításának tekinthetjük. minden egyes lépésnél az OTTER átteszi a „legkönnyebb” klózat a támogatóhalmazból a használható listára, és hozzáadja a használható listához a legkönnyebb klóz és a használható lista elemeinek rezolválásakor keletkezett néhány közvetlen következményt. Az OTTER leáll, amikor talál egy cífolatot, vagy amikor nincs több klóz a támogatóhalmazban. Az algoritmus a 9.14. ábrán látható részletesebben.

## A Prolog kiterjesztése

Egy tételebizonyító létrehozásának alternatív módja, ha a Prolog fordítóból indulunk ki, és ezt kiterjesztjük úgy, hogy egy helyes és teljes következetetőgépet kapunk a teljes elsőrendű logikához. Ezt a megközelítést alkalmazták a Prolog Technológia Tételbizonyítóban (Prolog Technology Theorem Prover, PTTP) (Stickel, 1988). A PTTP négy jelentős változtatást tartalmaz a Prologhoz képest, hogy helyreállítsa a teljességet és a kifejezőképességet:

- Az előfordulási próba visszakerül az egyesítési folyamatba, hogy azt biztosabbá tegye.
- A mélységi keresést felváltja egy iteratívan mélyülő keresés. Ez a keresési stratégiát teljessé teszi, és csak egy konstans értékkel vesz több időt igénybe.
- A negált literálok (mint a  $\neg P(x)$ ) engedélyezettek. A megvalósításban két különálló folyamat van, az egyik a  $P$ -t akarja bebizonyítani, a másik a  $\neg P$ -t.
- Egy  $n$  atommal rendelkező klóz  $n$  különböző szabályként kerül tárolásra. Például az  $A \Leftarrow B \wedge C$ -t úgy is tárolnánk, mint  $\neg B \Leftarrow C \wedge A$  és úgy is, mint  $\neg C \Leftarrow B \wedge \neg A$ . Ez a technika, amelyet zárasnak (locking) nevezünk, azt eredményezi, hogy az aktuális célt csak minden egyes klóz fejével kell egyesíteni, de még így is lehetővé teszi a negálás helyes kezelését.
- A következetést teljessé teszik (még a nem Horn-klázokra is) a lineáris bemeneti szabály hozzáadásával: ha a z aktuális cél egyestil a veremben lévő egyik cél negáltjával, akkor azt a célt rezolválnak kell tekinteni. Ez egy módja az ellentmondásokkal való következetetésnek. Tételezzük fel, hogy az eredeti cél a  $P$ , és ezt redukáltuk következetések sorozatával a  $\neg P$  célra. Ez a  $\neg P \Rightarrow P$  mondatra vezet, amely logikailag egyenértékű a  $P$ -vel.

A felsorolt változtatások ellenére a PTTP megőrzi azokat a tulajdonságokat, amelyek a Prologot gyorsá teszik. Az egyesítések még így is a változók közvetlen változtatásával történnek úgy, hogy a lekötések feloldása a visszalépés során az útvonal lezárásával együtt történik. A keresési stratégia itt is a bemeneti rezolváció alapszik, ami azt jelenti, hogy minden rezolváció a probléma egyik eredeti kijelentésében megadott klózzal szemben történik (és nem egy származtatott klózzal). Ez lehetővé teszi a probléma eredeti kijelentésében adott összes klóz lefordítását.

A PTTP fő hátránya az, hogy a felhasználó elveszít minden vezérlési lehetőséget a megoldások keresése során. minden következetési szabályt a rendszer mind az eredeti, mind a kontrapozitív formájában felhasznál. Ez nehezen értelmezhető kereséseket eredményez. Például figyeljük meg ezt a szabályt:

$$(f(x, y) = f(a, b)) \Leftarrow (x = a) \wedge (y = b)$$

Prolog szabályként tekintve, ez is egy következetes módja annak a bizonyításának, hogy két  $f$  term megegyezik. De a PTTP még a kontrapozitív állítást is generálhatja:

$$(x \neq a) \Leftarrow (f(x, y) \neq f(a, b)) \wedge (y = b)$$

Úgy tűnik, hogy ez igen fáradtságos módja annak, hogy bizonyítsuk két term,  $x$  és  $a$  különbözőségét.

### Tételbizonyítások mint segédeszközök

Eddig úgy tekintettünk a következtető rendszerekre, mint független ágensre, amelynek önállóan kell döntéseket hoznia és cselekednie. A tételbizonyítások másik felhasználása, amikor segítőként használjuk, és tanácsokkal lát el, mondjuk egy matematikust. Ebben a használati módban a matematikus felügyelőként viselkedik, feltérképezi a következő lépés meghatározásának stratégiáját, hogy mi legyen, és megkeríti a tételbizonyítót, hogy töltse ki a részleteket. Egy bizonyos fokig mellékessé teszi a félig eldöntetőség problémáját, mivel a felügyelő megszüntethet egy lekérdezést, és próbálhat egy másik megközelítést, ha a lekérdezés túl sok időt vesz igénybe. A tételbizonyítás **bizonyítás-ellenőrzőként (proof checker)** is működhet, ahol a bizonyítást mi adjuk meg vázlatosan, nagyobb lépések sorozataként, és a rendszer tölti ki az egyedi következetések részleteit, amelyek igazolják lépéseink helyességét.

A Socratic következtető (**Socratic reasoner**) egy tételbizonyító, amelynek a KÉRDEZ függvénye nem teljes, de amely minden előre meghatározott kérdést tud megfelelően válaszolni. Így tehát a Socratic következtetők jó segítők, feltéve, ha van egy felügyelő, aki a megfelelő KÉRDEZ hívásokat összeállítja. Az ONTIC (McAllester, 1989) egy Socratic következtető rendszer matematikai tételbizonyításhoz.

### A tételbizonyítások gyakorlati felhasználása

A tételbizonyítók újszerű matematikai eredményeket produkáltak. A SAM (Semi-Automated Mathematics) program volt az első, amely bebizonyított egy lemmát a rácselméletben (Guard és társai, 1969). Az AURA program szintén nyitott kérdéseket válaszolt meg a matematika különféle területein (Wos és Winker, 1983). A Boyer-Moore-tételbizonyítót (Boyer és Moore, 1979) sok éven keresztül használták és bővítették. Natarajan Shankar ezt használta fel, hogy megadja az első teljes, precíz formális bizonyítását Gödel nemteljesség tételenek (Shankar, 1986). Az OTTER program az egyik legerősebb tételbizonyító. Használták a kombinatorikus logika néhány nyitott kérdésének megoldására. A legismertebb ezek közül a Robbins-algebra. 1933-ban Herbert Robbins egy egyszerű axiómasorozatot javasolt, amely a Boole-algebra definíálására látszott alkalmASNak, de nem találtak hozzá bizonyítást (több matematikus, köztük Alfred Tarski, jelentős munkája ellenére). 1996. október 10-én, nyolc nap számítás után az EQP (az OTTER egyik változata) megtalálta a bizonyítást (McCune, 1997).

A tételbizonyításokat alkalmazhatjuk a hardver- és szoftvertervezésben a **verifikáció (verification)** és a **szintézis (synthesis)** során felmerülő problémákra, mivel minden tárgyterülethez lehetséges megfelelő axiómarendszer definiálni. Így tehát a tételbizonyí-

tási kutatást megtalálhatjuk a hardvertervezés, a programozási nyelvek és a szoftverfejlesztés területein is – nem csupán az MI-ben. A szoftver esetében az axiómák meg-határozzák a programozási nyelv minden egyes szintaktikus elemének tulajdonságait. (A programokról történő következtetés elégéhez hasonló az akciókról történő következtetéshez a szituációkalkulusban.) Egy algoritmus akkor tekinthető igazoltnak, ha a kimeneti megfelelnek a specifikációnak minden bemenet mellett. Az RSA nyilvános kulcskódolási algoritmust és a Boyer–Moore-féle húrillesztési algoritmust is ilyen módon igazolták (Boyer és Moore, 1984). A hardver esetében az axiómák leírják a jelek és az áramkör elemei közötti interakciókat (lásd a 8. fejezetbeli példát). Egy 16 bites összeadó tervét az AURA igazolta (Wojcik, 1983). A logikai következtetők, amelyeket speciálisan igazolásokra terveztek, egész CPU-kat voltak képesek verifikálni, beleértve ezek időzítési tulajdonságait is (Sivas és Bickford, 1990).

Az algoritmusok formális szintézise volt a tételbizonyítások egyik első felhasználása, amint azt Cordell Green (Green, 1969a) felvázolta, aki Simon korábbi ötleteire támaszkodott (Simon, 1963). Az alapötlet az volt, hogy egyféllel más módon bizonyítsák azt a tételeket, hogy „létezik egy  $p$  program, amely eleget tesz egy bizonyos specifikációnak”. Ha a bizonyítást úgy korlátozzuk, hogy konstruktív legyen, a program maga is kinyerhető lesz a bizonyításból. Habár ez egy teljesen automatizált **deduktív szintézis (deductive synthesis)**, legalábbis így nevezzük, még nemigen használható általános célú programok készítésére, ám a kézzel irányított deduktív szintézis már sikeresen tervezett néhány újszerű és bonyolult algoritmust. A speciális célú programok szintézise szintén aktív kutatási terület. Az AURA tételbizonyítót sikerkel alkalmazták a hardverszintézis területén olyan áramkörök tervezésére, amelyek kompaktabbak, mint minden azzal megelőző terv (Wojciechowski és Wojcik, 1983). Sok áramkör tervezésénél az ítéletlogika elegendő, mivel a legfontosabb ítéletállítások halmaza rögzített, ugyanis ezek az áramkör elemeit írják le. Az ítéletlogikai következtetés alkalmazása a hardverszintézisben ma már egy standard technika, amelynek sok nagyméretű hálózati alkalmazása létezik (lásd például Nowick és társai, 1993).

Ugynézeket a technikákat mostanában kezdik el alkalmazni a szoftverek verifikációjához is, például a SPIN modell ellenőrző programmal (Holzmann, 1997). A Remote Agent űrhajó vezérlési programot például sikerült verifikálni a repülés előtt és után (Havelund és társai, 2000).

## 9.6. ÖSSZEFOLGLALÁS

Az elsőrendű logikai következtetés egy elemzését mutattuk be, továbbá számos algoritmust, melyek ilyen következtetéseket képesek végrehajtani.

- Egy első megközelítés következtetési szabályokat használ kvantorok példányosítására azért, hogy átalakítsa a következtetési problémát az ítétkalkulusra. Ez a megközelítés jellemzően nagyon lassú megoldásokat eredményez.
- A egyesítés használata, azaz a megfelelő helyettesítések megkeresése változókhöz, megszünteti a példányosítás lépését az elsőrendű bizonyításokban, és így sokkal hatékonyabbá teszi a folyamatot.
- A **Modus Ponens** kiterjesztett változata az egyesítést használja, létrehozva egy természetes és nagy modellező erejű következtetési szabályt, az általánosított **Modus**

**Ponens (generalized Modus Ponens).** Az előrefelé láncolás (forward chaining) és a hátrafelé láncolás (backward chaining) algoritmusok határozott klózok halmazára alkalmazzák ezt a szabályt.

- Az általánosított Modus Ponens teljes a határozott klózokra, habár a következetési probléma félíg eldönthető (semidecidable). A Datalog programok esetében, amely programok csak függvénymentes határozott klózokat tartalmaznak, a következetés eldönthető.
- Az előrefelé láncolást a deduktív adatbázisokban (deductive databases) használják, ahol sikeresen kombinálják ezt a módszert a relációs adatbázis más műveleteivel. Ezenkívül az előrefelé láncolást használják a produkciós rendszerekben (production systems) is, amelyek hatékonyan képesek frissíteni a tudásbázisukat nagyméretű szabályhalmazokkal is.
- Az előrefelé láncolás teljes a Datalog programokra, és polinomiális időben fut.
- A hátrafelé láncolást a logikai programozási rendszerekben (logic programming systems) használják, mint amilyen például a Prolog, amely kifinomult fordítási technikákat alkalmaz, hogy biztosítsa a gyors következetést.
- A hátrafelé láncolásban sok a felesleges következetés, és előfordulhatnak végtelen hurkok. Ezeket a memók gyűjtésével (memoization) lehet kezelni.
- Az általánosított rezolúciós (resolution) következetés teljes bizonyítási rendszer az elsőrendű logikában. Az eljárás konjunktív normál formájú tudásbázisokat használ fel.
- Számos stratégia ismert a rezolúciós rendszerek keresési terének a csökkentésére, anélkül hogy a teljességet veszélybe sodornánk. A hatékony rezolúcióalapú tételebizonyításokat felhasználták, hogy érdekes matematikai tételeket bebizonyítsanak, és hogy verifikáljanak, valamint szintetizáljanak szoftvereket és hardvereket.

## Irodalmi és történeti megjegyzések

A logikai következetéseket széles körben vizsgálták már az ókori görög matematikusok is. Az Arisztotelész által legtöbbet vizsgált következetési módszer a **szillogizmus (syllogism)** volt. A szillogizmus „situációkra” és „hangulatokra” bontható a termek mondatbeli sorrendjétől függően (amelyeket predikátumoknak nevezünk ma) és az egyes termek általanosságának mértékétől függően (amit ma a kvantorok segítségével értelmezünk), illetve attól függően, hogy a termek negálva voltak-e. A legalapvetőbb szillogizmus, amely az első situáció első hangulata, a következő:

Minden *S* az *M* is.

Minden *M* a *P* is.

Ezért minden *S* a *P* is.

Arisztotelész más szillogizmusok érvényességét is megpróbálta igazolni, visszavezetve ezeket az első situációra. Pontosan leírta az egyes szillogizmusokhoz tartozó hangulatokat és situációkat, de a szillogizmusok igazolása már kevésbé volt precíz.

Gottlob Frege, aki teljes elsőrendű logikát fejlesztett ki 1879-ben, következetési rendszerét logikailag érvényes sémák nagyméretű gyűjteményére és egyetlen következetési szabályra alapozta, a Modus Ponensre. Frege kihasználta azt a tényt, hogy ennek

a formának „A  $P$ -ból következtesd a  $Q$ -t” következtetési szabályának a hatását szimulálhatja a Modus Ponens alkalmazásával a  $P$ -re, egy logikailag érvényes sémával, a  $P \Rightarrow Q$ -val. A kifejezésnek ezt az „axiomatikus” stílusát, amely a Modus Ponens, plusz számos logikailag érvényes sémát használ, Frege után még számos matematikus használta; a leginkább említésre méltó, hogy a *Principia Mathematica*ban (Whitehead és Russell, 1910) is felhasználták.

A következtetési szabályok, mint az axiomatikus sémáktól különálló dolgok álltak a természletes dedukció (**natural deduction**) központjában, amelyet Gerhard Gentzen és Stanisław Jaskowski vezetett be (Gentzen, 1934; Jaskowski, 1934). A természletes dedukciót azért nevezik „természletesnek”, mert nem kívánja meg a mondatok átalakítását az (olvashatatlan) normál formára, és következtetési szabályaikat úgy terveztek, hogy az emberek számára természletesnek tűnjenek. Prawitz egy egész könyv hosszúságú elemzést adott a természletes dedukcióról (Prawitz, 1965). Gallier az automatikus dedukció elméleti alátámasztásának kifejtéséhez alkalmazta Gentzen szekvenciáit (Gallier, 1986).

A klózforma felfedezése nagy jelentőségű lépés volt az elsőrendű logika matematikai elemzésének kifejlesztésében. Whitehead és Russell (Whitehead és Russell, 1910) kiterjesztették az úgynevet **átviteli szabályt** [maga a kifejezés Herbrandtól származik (Herbrand, 1930)], amit a kvantoroknak a formula elé történő kiemelésére alkalmaztak. A Skolem-konstansokat és a Skolem-függvényeket, viszonylag precíz leírást adva, Thoralf Skolem vezette be (Skolem, 1920). A skolemizáció teljes eljárása, a Herbrand-univerzum fontos fogalmának bevezetésével együtt egy későbbi írásban található (Skolem, 1928).

Herbrand elmélete, amit Jacques Herbrand francia matematikusról neveztek el, fontos szerepet játszott az automatikus következetető rendszerek módszereinek fejlesztésében, mind a Robinsonhoz fűződő rezolúció bevezetése előtt és után (Herbrand, 1930). Ezt tükrözi az, hogy habár maga a fogalom valójában Skolem eredménye, erre „Skolem-univerzum” helyett „Herbrand-univerzumként” hivatkozunk. Herbrandot említhetjük még az egyesítés felfedezőjeként is, mivel az egyesítés algoritmus egy változata megtalálható a (Herbrand, 1930)-ban. Gödel (Gödel, 1930) mutatta meg, Skolem és Herbrand elméleteire építve, hogy az elsőrendű logikának létezik teljes bizonyítási eljárása. Alan Turing és Alonzo Church egymástól függetlenül, igen különböző bizonyításokat alkalmazva megmutatták, hogy az érvényesség kérdése az elsőrendű logikában nem eldönthető (Turing, 1936; Church 1936). Enderton kiválóan megfogalmazott írása egy precíz, de viszonylag érthető stílusban magyarázza el ezeket az eredményeket (Enderton, 1972).

Habár McCarthy (McCarthy, 1958) javasolta először az elsőrendű logika használatát reprezentációs feladatokra és következetésre az MI-ben, az első ilyen rendszert a matematikai tételelbizonyítás iránt érdeklődő matematikai logika kutatói fejlesztették ki. Abraham Robinson javasolta először, hogy használják az ítéletlogikára történő átalakítást és Herbrand-elméletet együtt. Gilmore (Gilmore, 1960) készített először egy olyan programot, amely ezen a megközelítésen alapult. Davis és Putnam (Davis és Putnam, 1960) vezették be a klózformát, és hoztak létre egy olyan programot, amely megkísérelt cáfolatokat találni, oly módon, hogy a Herbrand-univerzum elemeit a változók helyére alapklózokat helyettesítve és ítéletinkonziszenciákat keresve az alapklózok között. Prawitz (Prawitz, 1960) vezette be azt az alapötletet, hogy az ítéletinkonziszencia lekérdezése vezesse a keresési folyamatot, és azt is, hogy csak akkor hozunk létre termeket

a Herbrand-univerzumban, hogyha ez szükséges az ítélet-inkonisztenzia megállapítása céljából. Más kutatók egyéb fejlesztései után, ez a gondolat vezette el J. A. Robinsont (nem rokona az előzőnek) a rezolúció módszerének kifejlesztéséhez (Robinson, 1965). Az úgynevezett „inverz módszer”, amit egyidejűleg fejlesztett ki S. Maslov szovjet kutató, kissé eltérő elveken alapult, mint Robinson rezolúciós módszere, de hasonló számítási előnyöket nyújt a propozicionálizációhoz képest (Maslov, 1964, 1967). Wolfgang Bibel **kapsolati módszere (connection method)** e megközelítés kiterjesztésének tekinthető.

A rezolúció kifejlesztése után az elsőrendű következtetéssel foglalkozó munkák különböző irányokban ágaztak el. Az MI-ben a rezolúciót a kérdés-válasz rendszerekre adoptálta Cordell Green és Bertram Raphael (Green és Raphael, 1968). Egy kevésbé formális megközelítést alkalmazott Carl Hewitt (Hewitt, 1969). Az ő PLANNER nyelve, habár sohasem valósították meg teljességében, előfutára volt a logikai programozásnak, és útmutatásokat tartalmazott az előrefelé és hátrafelé láncoláshoz, valamint a negáltak sikertelenségének vizsgálati módszeréhez. Az eredeti nyelv egy részét, a MICRO-PLANNER-t (Sussman és Winograd, 1970) megvalósították és felhasználták az SHRDLU természetes nyelvek megértését segítő rendszerben (Winograd, 1972). A korai MI-alkalmazások jelentős erőfeszítést tettek olyan adatstruktúrák előállítására, amelyek lehetővé teszik tények hatékony előhívását. Ezeket a munkákat is bemutatják az MI-programozással kapcsolatos cikkek (Charniak és társai, 1987; Norvig, 1992; Forbus és de Kleer, 1993).

A hetvenes évek elejére az **előrefelé láncolás (forward chaining)** jól megalapozottnak tekinthető az MI-ben, egy könnyen érthető alternatívája lett a rezolúciónak. Rendszerek széles körében került felhasználásra, Nevins geometriai tételebizonyítójától (Nevins, 1975) a VAX konfigurációjához használt R1 szakértői rendszerig (McDermott, 1982). Az MI-alkalmazások jellemzően nagyszámú szabályt tartalmaztak, tehát fontos volt kifejleszteni hatékony szabályillesztési technológiát, különösen az inkrementális frissítések problémájára. A **produkciós rendszerek (production systems)** technológiáját azért fejlesztették ki, hogy segítse az ilyen típusú az alkalmazások megvalósítását. Az OPS-5 produkciós rendszer nyelvet (Forgy, 1981; Brownston és társai, 1985) használták az R1-hez és a SOAR kognitív szerkezetéhez (Laird és társai, 1987). Az OPS-5 tartalmazta a rete illesztési folyamatot is (Forgy, 1982). A SOAR, amely új szabályokat generál, hogy megőrizze a megelőző számítások eredményeit, igen nagy szabályhalmazokat állíthat elő – több mint 1 000 000 szabályt a TACAIR-SOAR rendszer esetében, amely katonai repülőgépek szimulációját kontrollálja (Jones és társai, 1998). A CLIPS (Wygant, 1989) egy C-alapú produkciós rendszer nyelv, amelyet a NASA-nál fejlesztettek ki, és hatékony integrációt tett lehetővé más programokkal, eszközökkel és érzékelő rendszerekkel, valamint felhasználták ūrjárművek automatizálására és egyéb harcászati alkalmazásokra.

A **deduktív adatházisokként (deductive databases)** ismert kutatási terület szintén hozzájárult az előrefelé következtetés megértéséhez. Egy Toulouse-i szellemi műhelyben kezdődött a munka 1977-ben, amelyet Jack Minker szervezett, összehozva a logikai következtetési és az adatbázisrendszerekkel foglalkozó szakembereket (Gallaire és Minker, 1978). Egy történeti áttekintés (Ramakrishnan és Ullman, 1995) szerint: a „deduktív [adatbázis-] rendszerek megkísérik a Prolog adaptálását, a »kis mennyiséggű adat« megközelítésről a »nagymennyiséggű adat« világára.” Így tehát, ennek a munkának a célja az, hogy egybeolvassza a relációs adatbázis technológiát, amelyet nagyméretű

tényhalmazok előhívására terveztek, a Prolog-alapú következtetési technológiával, amely jellemzően egyszerre csak egy tényt hív elő. A deduktív adatbázisokról szóló szövegek közé tartoznak az (Ullman, 1989) és a (Ceri és társai, 1990) munkák.

Chandra és Harel (Chandra és Harel, 1980), valamint Ullman (Ullman, 1985) nagy hatású munkái a Datalognak mint egy, a deduktív adatbázisokhoz kidolgozott standard nyelvnek az elfogadásához vezettek. Az „alulról felfelé” irányú következtetés, az előrefelé láncolás szintén standard lett – részben azért, mert elkerüli a leállási problémákat és a felesleges számítások problémáját, amely a hátrafelé láncolásnál előfordul, és részben azért, mert természetesen felhasználható az alapvető relációs adatbázis-operációkhoz. A **mágikus halmazok** (*magic sets*) technikája a szabályok átírására, amelyet Bancilhon (Bancilhon és társai, 1986) fejlesztettek ki, lehetővé tette az előrefelé láncolásnak, hogy a hátrafelé láncolástól kölcsönözze a célorientáltság előnyös tulajdonságát. A „fegyverkezési verseny” kiegyenlítése céljából, a táblázatos logikai programozási módszerek kölcsönveszik a dinamikus programozás előnyét az előrefelé láncolástól.

A logikai következtetések komplexitásáról szerzett tudásunk legnagyobb része a deduktív adatbázisokkal foglalkozó tudományos közösségből származik. Chandra és Merlin (Chandra és Merlin, 1977) mutatták ki először, hogy egy egyszerű nem rekurzív szabály (vagyis egy **konjunktív lekérdezés** (*conjunctive query*) az adatbázisok terminológiájában) illesztése NP-nehéz lehet. Kuper és Vardi (Kuper és Vardi, 1993) javasolták az **adatkomplexitás** (*data complexity*) használatát – annak a mértéknek a használatát, ami a komplexitást, mint az adatbázis méretének egy függvényét méri, miközben a szabály méretét konstansnak tekinti – a lekérdezések megválasztásának mértékére. Gottlob és társai (Gottlob és társai, 1999b) a konjunktív lekérdezések és a kényszerkielégítés közötti kapcsolatot vizsgálták, megmutatva, hogy a hiper-fák lebonthatása hogyan optimizálja az illesztési folyamatot.

Mint azt már előzőleg említettük, a **hátrafelé láncolás** (*backward chaining*) Hewitt PLANNER nyelvében jelent meg a logikai következtetéshez (Hewitt, 1969). A logikai következtetés magától értetődően ettől a munkától függetlenül fejlődött tovább. A lineáris rezolúció egy korlátozott változatát, amelyet **SL-rezolúció**nak (**SL-resolution**) nevezünk, Kowalski és Kuehner (Kowalski és Kuehner, 1971) fejlesztette ki Loveland **modelleliminációs** (**model elimination**) technikájára építve (Loveland, 1968). Ennek a határozott klózokra alkalmazott változata az **SLD-rezolúció** (**SLD-resolution**), amely önmagában alkalmas határozott klózok, mint programok interpretálására (Kowalski, 1974; 1979a; 1979b). Ezalatt, 1972-ben, a francia kutató, Alain Colmerauer kifejlesztette és alkalmazni kezdte a **Prologot**, a természetes nyelvek elemzésének céljából. A Prolog klózait először kontextusmentes nyelvtani szabályoknak szánták (Roussel, 1975; Colmerauer és társai, 1973). Az elméleti háttér túlnyomó részét a logikai programozás részére Colmerauerrrel együttműködve Kowalski fejlesztette ki. A szemantikai definíció, amely a rögzített pontokat használja, Van Emdennek és Kowalskinak köszönhető (Van Emden és Kowalski, 1976). Kowalski (Kowalski, 1988) és Cohen (Cohen, 1988) jó történeti áttekintést nyújtanak a Prolog eredetéről. A *Logikai programozás alapjai* (*Foundations of Logic Programming*) (Lloyd, 1987) egy elméleti elemzése a Prolog és egyéb logikai programozónyelvek megalapozásának.

A hatékony Prolog fordítókat általában a Warren Abstract Machine (WAM) számítási modelljére alapozzák, amelyet David H. D. Warren (Warren, 1983) fejlesztett ki. Van Roy (Van Roy, 1990) megmutatta, hogy a további fordítási technikák alkalmazása, mint

amilyen a típusinterferencia, versenyképessé teszi a Prolog programokat a C programokkal a sebesség tekintetében. A Japán Ötödik Generációs projekt, amely egy 1982-ben kezdődő 10 éves kutatási erőfeszítés volt, teljes egészében a Prologen, mint az intelligens rendszerek kifejlesztésének alapeszközén alapult.

A rekurzív logikai programok felesleges ciklusainak elkerüléséhez egymástól függetlenül Smith (Smith és társai, 1986), valamint Tamaki és Sato (Tamaki és Sato, 1986) fejlesztettek ki módszereket. Az utóbbi tanulmány tartalmazta a memók gyűjtését is a logikai programokra, egy olyan módszert, amelyet teljes mértékben David S. Warren fejlesztett ki a **táblázatos logikai programozás** (**tailed logic programming**) módszerében. Swift és Warren (Swift és Warren, 1994) bemutatták, hogy hogyan terjesszük ki a WAM-ot táblázatok kezelésére, amely képessé tette a Datalog programokat, hogy egy nagyságrenddel gyorsabban fussanak, mint az előrefelé láncolást alkalmazó deduktív adatbázisrendszerek.

A korai elméleti munkákat a korlátozott logikai programozás területén Jaffar és Lassez végezték (Jaffar és Lassez, 1987). Jaffar és társai (Jaffar és társai, 1992a) fejlesztették a CLP(R) rendszert a valós értékű kényszerek kezelésére. Jaffar (Jaffar és társai, 1992b) általánosította a WAM-ot, létrehozva ezzel a CLAM-ot (Korlátozott Logikai Absztrakt Gép, Constraint Logic Abstract Machine) a CLP megvalósításainak specifikálásához. Ait-Kaci és Podelski (Ait-Kaci és Podelski, 1993) bemutatták egy kifinomult LIFE-nak nevezett nyelvet, amely egyesíti a CLP-t a funkcionális programozással és az öröklődés következetéssel. Kohn (Kohn, 1991) egy ambiciózus projektet mutat be a korlátozott logikai programozás használva, a valós idejű vezérlési architektúrákat megalapozva, teljesen automata pilóták alkalmazására.

A logikai programozásról és a Progról számos könyvet írtak. A *Logika a problémamegoldáshoz (Logic for Problem Solving)* (Kowalski, 1979b) egy korai általános tanulmány a logikai programozásról. A Prologgal foglalkozó könyvek közé tartoznak Clocksin és Mellish (Clocksin és Mellish, 1994), Shoham (Shoham, 1994) és Bratko (Bratko, 2001) frásai. Marriott és Stuckey (Marriott és Stuckey, 1998) kitűnő leírását adják a CLP-nek. A 2000-ben történt megszűnéseig a *Journal of Logic Programming* volt a téma legfontosabb folyóirata; helyét mára átvette a *Theory and Practice of Logic Programming*. A logikai programozások konferenciái között a legjelentősebbek az International Conference on Logic Programming (ICLP) és az International Logic Programming Symposium (ILPS).

A kutatók a **matematikai tételelbizonyítások** (**mathematical theorem proving**) területén már az első teljes elsőrendű logikai rendszerek kifejlesztése előtt megkezdődtek. Herbert Gelernter Geometriai Tételelbizonyítója (Gelernter, 1959) heurisztikai keresési módszereket használt diagramokkal kombinálva, hogy kiselejtezzze a hamis részcélokat, és képes volt bebizonyítani néhány nagyon bonyolult eredményt az euklideszi geometriában. Ettől kezdve azonban nem volt jelentős együttműködés a tételelbizonyítás és az MI között.

A korai munkák a teljességre koncentráltak. Robinson korszakalkotó tanulmányát követően a demodulációs és a paramodulációs szabályokat az egyenlőségi következetetésekre Wos (Wos és társai, 1967), valamint Wos és Robinson (Wos és Robinson, 1968) vezették be, ebben a sorrendben. Ezeket a szabályokat a term átírási rendszerek kontextusában is kifejlesztették (Knuth és Bendix, 1970). Az egyenlőségi következetés beépítése az egyenlőségi algoritmusba Gordon Plotkinnak köszönhető (Plotkin, 1972); ez fontos vonása volt a QLISP-nek is (Sacerdoti és társai, 1976). Jouannaud és Kirchner (Jouannaud és Kirchner,

1991) az egyenlőségi egyesítést a termek átírásának szempontjából vizsgálják. Hatékony algoritmusokat a standard egyesítésre Martelli és Montanari (Martelli és Montanari, 1976), valamint Paterson és Wegman (Paterson és Wegman, 1978) fejlesztettek ki.

Az egyenlőségi következetések mellett a tételbizonyítások tartalmaztak különféle speciális célú döntési folyamatokat. Nelson és Oppen (Nelson és Oppen, 1979) javasoltak egy nagy hatású sémát az ilyen eljárásoknak egy általános következetési rendszerbe integrálására. Más módszerek is hasonló problémákkal foglalkoztak, beleértve Stickel (Stickel, 1985) „elméleti rezolúcióját”, valamint Manna és Waldinger (Manna és Waldinger, 1986) „speciális relációit”.

Számos vezérlési stratégiát javasoltak a rezolúcióra, kezdve az egységpreferencia stratégiával (Wos és társai, 1964). A támogató halmaz stratégiát Wos (Wos és társai, 1964) javasolta, hogy biztosítson egy bizonyos fokú célorientáltságot a rezolúcióban. A lineáris rezolúció először Lovelandnél (Loveland, 1970) jelent meg. Genesereth és Nilsson (Genesereth és Nilsson, 1987, 5. fejezet) rövid, de átfogó elemzést nyújtanak a vezérlési stratégiák széles skálájáról.

Guard és társai (Guard és társai, 1969) egy korai SAM tételbizonyítót írnak le, amely segített megoldani egy megoldatlan problémát a rácselmeletben. Wos és Winkler (Wos és Winkler, 1983) áttekintést nyújtanak az AURA tételbizonyító eredményeiről, amelyeket a matematika és a logika különböző területein fellelhető problémák megoldása terén ért el. McCune (McCune, 1992) ezt folytatja, amikor felhasználja ezeket az eredményeket az AURA utódjának, az OTTER-nek alkalmazásában megoldatlan problémák megoldására. Weidenbach (Weidenbach, 2001) bemutatja a SPASS-t, az egyik legerőteljesebb jelenlegi tételbizonyítót. A *Computational Logic* (Boyer és Moore, 1979) című könyv az alapreferencia a Boyer-Moore-tételbizonyítóhoz. Stickel (Stickel, 1988) leírja a Prolog Technológiai Tételbizonyítót (PTTP), amely egyesíti a Prolog fordítás előnyeit a modell elimináció teljességével (Loveland, 1968). A SETHEO (Letz és társai, 1992) egy másik széles körben használt tételbizonyító, amely hasonló megközelítésen alapul; másodpercenként több millió következetést tud végrehajtani egy 2000-es munkaállomáson. A LEANTAP (Beckert és Posegga, 1995) hatékony tételbizonyító, amelyet minden össze 25 Prolog sorral valósítottak meg.

A automata programszintézisről szóló korai munkákat Simon (Simon, 1963), Green (Green, 1969a), valamint Manna és Waldinger (Manna és Waldinger, 1971) készítették. Burstall és Darlington transzformációs rendszere (Burstall és Darlington, 1977) egyenlőség következetést használt a rekurzív programszintézishez. A KIDS (Smith, 1990, 1996) az egyik legerősebb modern rendszer, amely egy szakértő segédeszközökkel működik. Manna és Waldinger (Manna és Waldinger, 1992) egy áttekintő bevezetést adtak a téma-kör aktuális helyzetéről fókuszba állítva a saját deduktív megközelítésüket. Az *Automating Software Design* (Lowry és McCartney, 1991) számos, a témről szóló tanulmányt gyűjtött össze. A logikának a hardvertervezésben való felhasználásáról Kern és Greenstreet (Kern és Greenstreet, 1999) adott egy áttekintést; Clarke (Clarke és társai, 1999) műve pedig a modellellenőrzéssel foglalkozik a hardververifikálásban.

A *Computability and Logic* (Boolos és Jeffrey, 1989) egy jó referencia a teljesség és a nem előnöthetőség téma-köréhez. Számos korai tanulmány a matematikai logikáról megtalálható a *From Frege to Gödel: A Source Book in Mathematical Logic* (van Heijenoort, 1967) című könyvben. A tiszta matematikai logika téma-körének folyóirata a *Journal of Symbolic Logic*. Az automatizált dedukció irányába haladó könyvek közé tar-

tozik a klasszikus *Symbolic Logic and Mechanical Theorem Proving* (Chang és Lee, 1973), és számos más írás, köztük Wos, Bibel és Kaufmann később íródott munkái (Wos és társai, 1992; Bibel, 1993; Kaufmann és társai, 2000). Az *Automation of Reasoning* c. antológia (Siekmann és Wrightson, 1983) sok fontos korai tanulmányt tartalmaz az automatizált dedukcióról. További áttekintő műveket írtak Loveland (Loveland, 1984) és Bundy (Bundy, 1999). A legjelentősebb folyóirat a tételbizonyítások területén a *Journal of Automated Reasoning*; a legfontosabb konferencia az évente megtartott Conference on Automated Deduction (CADE). A tételbizonyítás területén folyó kutatás szintén szoros kapcsolatban áll a logika használatával a programok és programozási nyelvek elemzésében, amely tárgyban a legfőbb konferencia a Logic in Computer Science.

## Feladatok

- 9.1. Vezesse le az alaptételekből, hogy az univerzális példányosítás helyes, és azt, hogy az egzisztenciális példányosítás következetetesi szempontból egyenértékű tudásbázist hoz létre.
- 9.2. A *Szereti*( János, *Fagylalt*) mondatból kiindulva logikusnak tűnik arra következtetni, hogy  $\exists x \ Szereti(x, Fagylalt)$ . Adjon meg egy általános következetetesi szabályt, az egzisztenciális bevezetést (**Existential Introduction**), amely megerősíti ezt a következetést. Gondosan vizsgálja meg a feltételeket, amelyeket ki kell elégíteni a felhasznált változókkal és termekkel.
- 9.3. Tételezzük fel, hogy egy tudásbázis csak egy mondatot tartalmaz:  $\exists x \ OlyanMagasMint(x, Everest)$ . A következők közül melyek a törvényszerű következményei az Egzisztenciális Példányosítás alkalmazásának?
  - (a)  $OlyanMagasMint(Everest, Everest)$
  - (b)  $OlyanMagasMint(Kilimandzsáró, Everest)$
  - (c)  $OlyanMagasMint(Kilimandzsáró, Everest) \wedge OlyanMagasMint(BenNevis, Everest)$   
(szabály kétszeri alkalmazása után).
- 9.4. Adj meg az alábbi atomi mondatpárok legáltalánosabb egyesítőjét, ha egyáltalán létezik ilyen:
  - (a)  $P(A, B, B), P(x, y, z)$
  - (b)  $Q(y, G(A, B)), Q(G(x, x), y)$
  - (c)  $Idősebb(Apja(y), y), Idősebb(Apja(x), János)$
  - (d)  $Ismeri(Apja(y), y), Ismeri(x, x)$
- 9.5. Figyelje meg a 9.2. ábrán bemutatott bennfoglalási rácsokat.
  - (a) Hozza létre a rácsot a következő mondatra: *Alkalmaz(Anyja(János), Apja(Richárd))*.
  - (b) Hozza létre a rácsot a következő mondatra: *Alkalmaz(IBM, y)* („Mindeni az IBM-nél dolgozik”). Figyeljen arra, hogy minden olyan lekérdezést soroljon, amely egyesíthető a mondattal.

- (c) Tételezzük fel, hogy a TÁROL indexel minden egyes mondatot, minden egyes csomópontra a bennfoglalási rácsban. Magyarázza meg, hogy a BETÖLT eljárásnak hogyan kell működnie, amikor néhány mondat ezek közül változókat is tartalmaz; példaként használja a mondatokat az (a) és (b) részfeladatokból, és a következő lekérdezést: *Alkalmaz(x, Apja(x))*.
- 9.6.** Tételezzük fel, hogy betöltsük egy logikai adatbázisba az amerikai választási címjegyzéket, amely felsorolja a korát, a lakóhelyét, a születési dátumát és az anyja nevét minden személynek, társadalombiztosítási számokat használva azonosító adatként. Így tehát György kora így van megadva: *Kora(443-65-1282, 56)*. A következő S1–S5 indexelő sémák melyike tesz lehetővé hatékony megoldást a Q1–Q4 lekérdezésekre (tekintsünk egy normál hátrafelé láncolást)?
- **S1:** egy index minden egyes atomra minden egyes pozícióban.
  - **S2:** egy index minden egyes első argumentumra.
  - **S3:** egy index minden egyes predikátumatomra.
  - **S4:** egy index a predikátum és az első argumentum minden egyes *kombinációjára*.
  - **S5:** egy index a predikátum és a második argumentum minden egyes *kombinációjára*, és egy index minden egyes első argumentumra (nem standard).
  - **Q1:** *Kora(443-44-4321, x)*
  - **Q2:** *Lakik(x, Houston)*
  - **Q3:** *Anyja(x, y)*
  - **Q4:** *Kora(x, 34)  $\wedge$  Lakik(x, PiciVárosUSA)*
- 9.7.** Feltételezzetnénk, hogy az egyesítés során a változó konfliktus problémáját elkerülhetjük úgy, ha minden mondatnál egyszerre átnevezzük az összes változót a tudásbázisban. Mutassa meg, hogy léteznek olyan mondatok, amelyekre ez a megközelítés nem alkalmazható. (*Segítség:* tekintsünk egy olyan mondatot, amelynek egyik része egyesíthető a többivel.)
- 9.8.** Mutassa meg, hogy hogyan írhatunk meg egy tetszőleges méretű adott 3-SAT problémát, egy elsőrendű határozott klózt és nem több, mint 30 alaptényt felhasználva.
- 9.9.** Adja meg az alábbi mondatok olyan logikai reprezentációját, amely alkalmas arra, hogy az Általánosított Modus Ponens szabályt alkalmazzuk rájuk:
- (a) A lovak, a tehenek és a malacok emlősök.
  - (b) Egy ló leszármazottja is ló.
  - (c) Kékszakáll egy ló.
  - (d) Kékszakáll Charlie szülője.
  - (e) A leszármazott és a szülő inverz relációk.
  - (f) minden emlősnek van szülője.
- 9.10.** Ebben a problémában a 9.4. feladatban bemutatott mondatokat fogjuk használni. Válaszolja meg a következő kérdéseket hátrafelé láncolást alkalmazva!

- (a) Rajzolja fel egy kimerítő hátrafelé láncolás algoritmus bizonyítási fáját a következő mondat igazolásához:  $\exists h \ Ló(h)$
- (b) Mi figyelhető meg erről a tényterületről?
- (c) Hány megoldás származtatható le  $h$ -ra a mondatokból?
- (d) Tudna-e olyan módszert mondani, amellyel minden megkaphatjuk? (Segítség: érdemes megnézni (Smith és társai, 1986).)
- 9.11.** Egy népszerű találós kérdés gyerekeknek a következő: „Nincs se bátyám, se nővérem, mégis annak az embernek az apja az én apám fia.” Használja fel a családtárgykör szabályait (lásd 7. fejezet), és mutassa meg, hogy ki is az említett ember. Használhatja bármelyik, a fejezetben bemutatott következtetési módszert.
- 9.12.** Kövesse nyomon a 9.6. ábrán látható hátrafelé láncolási algoritmus végrehajtását, amikor azt a bűntény probléma megoldására alkalmazzuk. Mutassa be azt az érték-szekvenciát, amelyet a célok változó vesz fel, és rendezze egy fa formába.
- 9.13.** A következő Prolog kód egy P predikátumot határoz meg:
- ```
P(X, [X|Y])
P(X, [Y|Z]) :- P(X, Z)
```
- (a) Mutasson be bizonyítási fákat és megoldásokat ezekre a lekérdezésekre:  
 $P(A[1, 2, 3])$  és  $P(2, [1, A, 3])$
- (b) Milyen standard operációs listát reprezentál a P?
- 9.14.** Ebben a feladatban megvizsgáljuk egy sorbarendezés megvalósítását a Prologban.
- (a) Írjon Prolog-klózokat, amelyek definiálják a `rendezés(L)` predikátumot, amely akkor és csak akkor igaz, ha az L lista emelkedő sorrendben van rendezve.
- (b) Írjon egy Prolog-definíciót erre a predikátumra: `perm(L, M)`, amely akkor és csak akkor igaz, ha az L a permutációja az M-nek.
- (c) Definiálja a `rendez(L, M)`-et (az M egy válogatott verziója az L-nek) a `perm` és a `rendezés` használatával.
- (d) Futtassa a `rendez` predikátumot minél hosszabb listákon, amíg el nem veszíti a türelmét. Mekkora a programjának az időkomplexitása?
- (e) Írjon egy gyorsabb rendező algoritmust Prologban, mint amilyen például a beszúrásos válogatás vagy a gyorsválogatás (quicksort).
- 9.15.** Ebben a feladatban megvizsgáljuk az újraíró szabályok rekurzív alkalmazását a logikai programozás felhasználásával. Egy újraíró szabály (vagy **demodulátor** az OTTER terminológiában) egy egyenlet egy megadott irányával. Például az  $x + 0 \rightarrow x$  újraíró szabály azt sugallja, hogy minden, az  $x + 0$ -hoz illeszkedő ki-fejezést fel kell cserélni  $x$ -re. Az újraíró szabályok alkalmazása központi része az egyenletkövetkeztető rendszereknek. Ezt a predikátumot fogjuk használni: `újraír(X, Y)` az újraíró szabályok reprezentálására. Például a korábbi újraíró szabályt így írtuk: `újraír(X+0, X)`. Néhány term `primitív`, és nem lehet tovább

egyszerűsíteni; így tehát ezt fogjuk írni:  $\text{primitív}(0)$ , ami azt jelenti, hogy a 0 egy primitív term.

- (a) Írjon egy definíciót az egyszerűsít(X, Y) predikátumra, amely akkor igaz, amikor az Y az egyszerűsített változata X-nek – tehát amikor már több újraíró szabály nem alkalmazható az Y egyik részkifejezésére sem.
- (b) Írjon egy szabálygyűjteményt az aritmetikai operátorokat tartalmazó kifejezések egyszerűsítésére, és alkalmazza az egyszerűsítési algoritmusát néhány minta kifejezésre.
- (c) Írjon egy újraíró szabálygyűjteményt a szimbolikus differenciálásra, és használja az egyszerűsítési szabályaival együtt, hogy differenciáljon, és egyszerűsítse aritmetikai kifejezéseket tartalmazó kifejezéseket, beleértve a hatványozást.

- 9.16.** Ebben a feladatban megvizsgáljuk a keresési algoritmusok alkalmazását a Prologban. Tételezzük fel, hogy a következő(X, Y) akkor igaz, ha az Y állapot az X állapotot követi; és ha a cél(X) akkor igaz, amikor az X a célállapot. Írjon egy definíciót a meghold(X, P)-re, ami azt jelenti, hogy a P az útvonal (az állapotok listája), amely az X-szel kezdődik, a célállapotban végződik, és szabályos lépésekkel áll, amelyeket a következő határoz meg. Valószínűleg a mélységi keresés a legkönnyebb módja annak, hogy ezt végrehajtsa. Mennyire lenne könnyű egy heurisztikus keresési vezérlés hozzáadása?
- 9.17.** Hogyan alkalmazható a rezolúció annak megmutatásra, hogy egy mondat
- (a) érvényes-e?
  - (b) kielégíthetetlen-e?
- 9.18.** Abból a mondatból, hogy „a lovak állatok” következik-e, hogy „egy ló feje egy állat feje”. Demonstrálja, hogy ez a következetés érvényes, ha végrehajtjuk a következő lépéseket:
- (a) Fordítsa le a premisszát és a következményt az elsőrendű logika nyelvére. Használjon három predikátumot:  $\text{Feje}(h, x)$ ,  $\text{Ló}(x)$  és  $\text{Állat}(x)$ !
  - (b) Negálja a következményt, és konvertálja a premisszát és a negált következményt konjunktív normál formára!
  - (c) Mutassa meg a rezolúció felhasználásával, hogy a következmény valóban következik a premisszából!
- 9.19.** Tekintsük a következő két elsőrendű logikai mondatot:
- (A):  $\forall x \exists y (x \geq y)$
  - (B):  $\exists y \forall x (x \geq y)$
  - (a) Legyenek a változók lehetséges értékei a természetes számok 0, 1, 2, ...,  $\infty$ , és jelentse a „ $\geq$ ” predikátum azt, hogy „nagyobb vagy egyenlő”. Fordítsa le ezeket a mondatokat magyarra ebben az interpretációban!
  - (b) Igaz-e (A) ebben az interpretációban?
  - (c) Igaz-e (B) ebben az interpretációban?
  - (d) Maga után vonja-e az (A) mondat a (B) mondatot?
  - (e) Maga után vonja-e a (B) mondat az (A) mondatot?

- (f) Próbálja meg a rezolúció felhasználásával bizonyítani, hogy (A) következik (B)-ból! Tegye meg ezt akkor is, ha azt gondolja, hogy (B) nem vonja maga után (A)-t; folytassa az eljárást addig, amíg a bizonyítás már nem folytatható. Tüntesse fel az egyesítő helyettesítéseket minden rezolúciós lépésnél! Ha a bizonyítás sikertelen, magyarázza meg, hogy hol, hogyan és miért akad meg!
- (g) Most próbálja meg igazolni azt, hogy (B) következik (A)-ból!
- 9.20. A rezolúció változókkal való lekérdezésekre létrehozhat nem konstruktív bizonyításokat, ezért egy speciális eljárást kell bevezetnünk a definit válaszok kinyerésére. Magyarázza meg, hogy ez a probléma miért nem merül fel csak definit klózokat tartalmazó tudásbázisokkal!
- 9.21. Ebben a fejezetben azt mondottuk, hogy a rezolúció nem használható egy mondat-halmaz összes logikai következményének a generálására. Van olyan algoritmus, ami képes erre?

# 10. TUDÁSBÁZIS REPREZENTÁCIÓ

Ebben a fejezetben megmutatjuk, hogy az elsőrendű logikát hogyan kell használni a valódi világ olyan legfontosabb aspektusainak ábrázolására, mint a cselekvés, a tér, az idő, a mentális események és a bevásárlás.

Az utolsó három fejezet a tudásalapú ágens technológiáját mutatta be, avagy az ítélet- és az elsőrendű logika szintaxisát, szemantikáját, bizonyításelméletét és az ilyen logikákat használó ágens implementációját. Ebben a fejezetben azzal a kérdéssel foglalkozunk, hogy milyen *tartalmat* tegyünk az ágens tudásbázisába, azaz, hogy a vilagról szóló tényeket hogyan reprezentáljuk.

A 10.1. alfejezet bevezeti az általános ontológia gondolatát, amely világban minden a kategóriák hierarchiájába szervez. A 10.2. alfejezet az objektumok és a szubsztan-ciák alapvető kategóriáival foglalkozik. A 10.3. alfejezet a cselekvések reprezentációját elemzi, amelyek a tudásalapú ágens megtervezése szempontjából központi fontosságúak, továbbá bemutatja a tér-idő darabkák, az **események (events)** általánosabb fogalmát. A 10.4. alfejezet a hiedelmekkel foglalkozik, a 10.5. alfejezet pedig az egészet egybefogja az internetes bevásárlás kontextusában. A 10.6. és a 10.7. alfejezet a bizonytalan és a változó ismeretekkel dolgozó specializált következtető rendszerekkel foglalkozik.

## 10.1. ONTOLOGIASZERVEZÉS

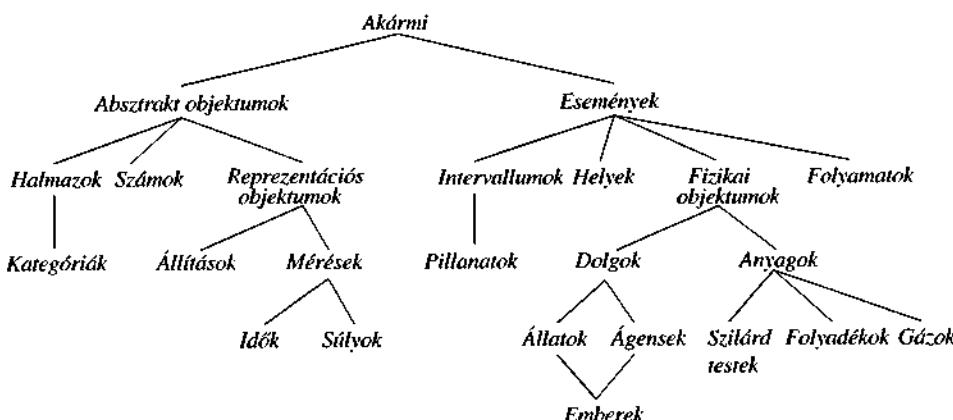
„Játék” tárgyterületeken a reprezentáció megválasztása nem annyira lényeges. Könnyű egy konzisztens fogalomkészletet kialakítani. Az olyan komplex területeken azonban, mint például az interneten történő bevásárlás vagy egy robot vezérlése változó fizikai környezetben, általánosabb és rugalmasabb reprezentációra van szükség. Ebben a fejezetben megmutatjuk, hogy ilyen reprezentációkat hogyan lehet létesíteni, olyan általános fogalmakra fókuszával, mint a *Cselekvés*, az *Idő*, a *FizikaiObjektum* és a *Hiedelem*, amelyek számos tárgyterületen fordulnak elő. Ezen absztrakt fogalmak reprezentálását néha **ontológiaszervezésnek (ontological engineering)** nevezik. Az ontológiaszervezés kapcsolatban áll a 8.4. alfejezetben leírt tudásszervezés folyamatával, de a hatása szélesebb körű.

A vilagról *mindent* reprezentálni ijesztő perspektíva. Valójában természetesen nem fogjuk mindennek a komplex leírását megadni – ez még egy 1000 oldalas könyvnek is sok lenne –, de megjelöljük világosan annak a helyét, ahova egy tetszőleges területről szóló új tudás beilleszthető. Így például definálni fogjuk, hogy mit is jelent fizikai objektumnak lenni, és a különböző objektumok részletei – robotok, tv-készülékek, könyvek vagy bármi más – később illeszthetők ebbe a keretbe. A fogalmak egy általános

keretét **felső ontológiának** (*upper ontology*) nevezük, mert az az általános konvenció, hogy a gráfszerű ábrázolásnál az általánosabb fogalmak felül helyezkednek el, a konkrétabb fogalmak pedig alul (lásd 10.1. ábra).

Mielőtt az ontológiát tovább tekintenénk, egy fontos figyelmeztetést kell tennünk. A tudás tartalmának és szervezésének a megvitatására elsőrendű logikát választottunk. A valódi világ bizonyos aspektusait ezen a nyelven nehéz lesz kifejezni. Egy alapvető tulajdonságot kényetlenek leszünk kihagyni: azt, hogy minden általánosítás alól léteznek kivételek, vagy hogy a részletesebb információ hiányában csupán az alapértelmezést tekinthetjük, vagy pedig hogy minden általánosítás csak egy bizonyos mértékig érvényes. Bár a „paradicsom piros” hasznos szabály, van zöld, sárga és narancssárga paradicsom is. Hasonló kivételeket ebben a részfejezetben majdnem minden egyik állítás esetén lehet találni. A kivételek és a bizonytalanság kezelése nagyon fontos képességek, azonban egy általános ontológia megértéséhez viszonyítva ortognálisak. Ezért a kivételek és az alapértelmezés megvitatását a 10.6. alfejezetre, a bizonytalan információ sokkal általánosabb témaját pedig a 13. fejezetre hagyjuk.

Milyen haszn van egy felső ontológiának? Gondolunk vissza a 8.4. alfejezet áramköri ontológiájára. Megkonstruálásánál igen sok egyszerűsítő feltételezéssel éltünk. Így például az időt teljes egészében ki is hagytuk. A jelek rögzítettek voltak, és a jeltovábbterjedésről nem is esett szó. Az áramkörök struktúrája változatlan maradt. Az általánosság felé úgy tudnánk lépni, hogy a jeleket konkrét időpillanatokban definíálnánk és foglalkoznánk a vezetékek hosszával, valamint a vezetékekben és a berendezésekben fellépő jelterjedési késleltetésekkel. Ezzel szimulálni tudnánk az áramkör időzítési tulajdonságait; és valóban az áramkörtervezők gyakran folyamodnak is az ilyen elemzéshez. A kapuk érdekesebb osztályait is be lehetne vezetni, például leírva a technológiát (TTL, MOS, CMOS stb.) és a bemeneti/kimeneti specifikációkat is. Ha a megbízhatósággal vagy diagnózissal szeretnék foglalkozni, engedélyezni kellene, hogy az áramkör struktúrája spontán módon meg változhasson. A szót kapacitások figyelembevételéhez a tisztán topológiai reprezentációval fel kellene hagyni, és át kellene téni a geometriai tulajdonságok valósághűbb leírására.



**10.1. ábra.** A világ ontológiájának felsőbb szintjei, amelyek a fejezetben később tárgyalott témaikat mutatják. minden él azt jelzi, hogy az alsó fogalom a felső fogalom egy specializálódása.

Hasonló megfontolások érvényesek, ha a wumpus világát nézzük. Bár foglalkozunk az idővel, ennek struktúrája igen egyszerű. A dolgok csak az ágens cselekvésével egy időben történnek, és minden változás azonnali jellegű. Egy általánosabb, a valódi világhoz jobban illeszkedő ontológia engedélyezné például, hogy az egyidejű változásoknak legyen időbeli kiterjedésük. Arra, hogy egy konkrét négyzetben csapda van, a *Csapda* konstanst használtuk, mert minden csapda azonos volt. Engedélyezni lehetne a csapdák több fajtáját is, a csapdák osztályához tartozó, eltérő tulajdonságú példányokat használva. Hasonló módon több állatfajtát is be tudnánk vezetni a wumpuson kívül. Elképzelhető, hogy a rendelkezésre álló érzékelésekkel az állat konkrét fajtáját nehéz lenne megállapítani. Az ágens megsegítésére a wumpus világ biológiai taxonómiáját fel lehetne állítani, hogy gyenge nyomravezető jelekből képes legyen megjósolni az állat viselkedését.

Az ilyen módosítások révén minden speciális rendeltetésű ontológiában lehetséges egy nagyobb általánosság felé elmozdulni. A nyilvánvaló kérdés ilyenkor az, hogy konvergálnak-e ezek a módosítások egy általános célú ontológiához? Évszázadokon át folyó filozófiai és számítási kutatások után a válasz az, hogy „lehetséges”. Ebben a fejezetben egy lehetséges változatot mutatunk be, amely az évszázadok során született gondolatok szintetizálását képviseli. Az általános célú ontológiának két olyan fő jellemzője van, ami azt a speciális rendeltetésű ontológiák sokaságától megkülönbözteti:

- Az általános ontológiát (a tárgytartományra vonatkozó axiomák hozzáadásával) többé-kevésbé minden speciális rendeltetésű tárgytartományban kell tudnunk alkalmazni. Lehetőség szerint tehát egyetlen reprezentációs problémával sem lehet ravaszkodni vagy azt a szönyeg alá söpörni.
- minden kellően igényes tárgytartományban a tudás egyes részeit *egyesíteni* kell, hiszen a következetés és a problémamegoldás egyszerre több területet is igényelhet. Egy robot áramkörjavító rendszer esetén például az áramkörökön a villamos összeköttetések és a fizikai elrendezés szempontjából kell tudnunk következtetni, de az idő az áramkör-időzítési analízis és a munkaköltségek szempontjából egyaránt lényeges. Az időt leíró állításokat tehát össze kell tudnunk kombinálni a fizikai elrendezést leíró állításokkal, és ezeknek az állításoknak egyformán jól kell működniük nanomásodpercekre és percekre is, valamint nanometerekre és métrekre is.

Miután az általános ontológiát megalkottuk, felhasználjuk az internetes bevásárlás tárgytartományra. Ez a tárgytartomány több mint alkalmas arra, hogy az ontológiánkkal kísérletezzünk. Sok helyet hagy az olvasó részére is, hogy a reprezentációba a saját kreativitását is bevhesse. Gondoljunk például arra, hogy egy interneten bevásárló ágensnek rengeteg témát és szerzőt kell ismernie, hogy az Amazon.com-tól könyveket vásároljon, élelmiszerek egész választékáról kell tudnia, hogy a Peapod.com-nál élelmiszer vásároljon, és mindenről, amit egy bolhapiacon találni lehet ismeretekkel kell rendelkeznie, hogy az Ebay.com<sup>1</sup>-nál az alkalmi jó üzletekre vadásszon.

<sup>1</sup> Az olvasó elnélkül, ha rajtunk kívül eső okok miatt ezen online boltok valamelyike a könyv olvasásának pillanatában már nem működik.

## 10.2. KATEGÓRIÁK ÉS OBJEKTUMOK

 Az objektumok **kategóriákba** (*categories*) való szervezése a tudásreprezentáció szempontjából létfontosságú. Bár a világgal való kölcsönhatás az egyedi objektumok szintjén történik, a következetések zöme a kategóriák szintjén valósul meg. A bevásárló célja egy kosárlabda megvásárlása és nem egy konkrét kosárlabdapéldánynak, mondjuk  $KL_9$ -nek a megvétele. Ha az objektumok osztályozását elvégeztük, kategóriák segítségével megjósolhatjuk az objektumok tulajdonságait. Érzékelő jelek alapján következtetünk bizonyos objektumok jelenlétére, az érzékelő tulajdonságokból következtetünk, hogy az objektumok milyen kategóriához tartoznak, majd a kategóriára vonatkozó ismereteket felhasználhatjuk, hogy az objektumokra vonatkozóan előrejelzéseket hozzunk. Így például a zöld, foltos felület, a tekintélyes nagyság és az ovális vagy gömb szerű alak alapján görög dinnyére következtethetünk; a görög dinnyére vonatkozó ismereteink alapján pedig arra, hogy az felhasználható lenne gyümölcsesséhez.

Az elsőrendű logikában a kategóriák reprezentálására két alapvető választásunk lehet: a predikátumok és az objektumok. Használhatunk egy *KosárLabda(l)* predikátumszimbólumot, vagy a kategóriát *KosárLabda* objektumként **reifikálhatjuk** (**reification**). Mondhatjuk akkor, hogy *Eleme(l, KosárLabda)* (rövidítve  $l \in KosárLabda$ ) annak a kifejezésére, hogy  $l$  a kosárlabda-kategória eleme. Azt mondjuk, hogy *Részalmaza(KosárLabda, Labda)* (rövidítve  $KosárLabda \in Labda$ ) annak a kifejezésére, hogy a *KosárLabda* a *Labda* egy alkategóriája vagy részhalmaza. Egy kategóriát tekinthetünk az elemeiből álló halmaznak, de egy sokkal bonyolultabb objektumnak is képzelhetjük, olyannak, amire az *Eleme* és a *Részalmaza* relációk definiáltak.

A kategóriák **örökölődés** (*inheritance*) révén szolgálják a tudásbázis szervezését és egyszerűsítését. Ha kijelentjük, hogy az *Élelem* kategória minden egyes példánya ehető, és feltételezzük, hogy a *Gyümölcs Élelem*, az *Alma* viszont a *Gyümölcs* kategória egy alosztálya, akkor tudjuk, hogy minden alma ehető. Azt mondjuk, hogy az egyes almák az ehetőségi tulajdonságukat örökölték (*inherit*), jelen esetben az *Élelem* kategóriához való tartozás révén.

Az alosztály-relációk a kategóriákat **taxonómia**ba vagy **taxonomikus hierarchiába** (**taxonomy, taxonomic hierarchy**) szervezik. A taxonómikákat tudományos területeken évszázadok óta alkalmazták. Így például a rendszerező biológia az összes élő és kihalt faj taxonómiaját igyekszik megadni, a könyvtártudomány az összes tudományos területet átfogó taxonómiaját – a Dewey Decimális rendszert – alakította ki, az adóhatóságok és más kormányzati szervek a foglalkozások és termékek kiterjedt taxonómiaját hozták létre. A taxonómikák – ahogy a további elemzésekben látni fogjuk – fontos aspektusai a józan ész tudásnak is.

A kategóriákról az elsőrendű logikában könnyű állításokat megfogalmazni az objektumok és a kategóriák egymáshoz való rendelésével vagy a kategóriához tartozó egyedeik szerinti kvantifikálás révén:

- Egy objektum egy kategória egyede, például:  
 $KL_9 \in KosárLabda$
- Egy kategória egy másik kategória alosztálya, például:  
 $KosárLabda \subset Labda$

- Egy kategória összes egyede egy bizonyos tulajdonsággal rendelkezik, például:  
 $\forall x \in \text{KosárLabda} \Rightarrow \text{Gömbölű}(x)$
- Egy kategória egyedeit bizonyos tulajdonságaik alapján fel lehet ismerni, például:  
 $\text{Narancsszinű}(x) \wedge \text{Gömbölű}(x) \wedge \text{Átmérő}(x) = 24 \wedge x \in \text{Labda} \Rightarrow x \in \text{KosárLabda}$
- Egy kategóriának önmagában bizonyos tulajdonságai vannak, például:  
 $\text{Kutya} \in \text{HonosítottFajta}$

Jegyezzük meg, hogy mivel a *Kutya* egy kategória és a *HonosítottFajta* kategóriának egy egyede, így a *HonosítottFajta* a kategóriák kategóriája. A kategóriák kategóriáinak a kategóriáiról is lehetne beszélni, de ennek kevés a haszna.

Bár az alosztály- és az egyedrelációk a kategóriák szempontjából a legfontosabbak, olyan kategóriák közötti relációkat is ki szeretnénk fejezni, amelyek egymásnak nem alosztályai. Ha például azt mondjuk, hogy a *Hím* és a *Nőstény* az *Állat* alosztályai, ezzel nem mondduk azt, hogy egy hím nem lehet nőstény. Azt mondjuk, hogy két vagy több kategória **diszjunkt** (*disjoint*), ha közös egyedei nincsenek. Ha tudjuk azt is, hogy a hímek és a nőstények kölcsönösen kizártják egymást, attól még nem tudjuk, hogy egy állatnak, amely nem hím, nősténynek kell lennie, hacsak nem mondjuk ki, hogy a hímek és a nőstények az állatok **kimerítő felosztását** (*exhaustive decomposition*) képezik. A diszjunkt kimerítő felosztás a **partíció** (*partition*). Ezt a három fogalmat az alábbi példák illusztrálják:

*Diszjunkt({Állat, Zöldség})*

*KimerítőFelosztás({Amerikai, Kanadai, Mexikói}, ÉszakAmerikai)*

*Partíció({Hím, Nőstény}, Állat)*

(Jegyezzük meg, hogy az *ÉszakAmerikai* *KimerítőFelosztás*-a nem *Partíció*, mert vannak kettős állampolgárságú személyek is.) E három predikátum definíciója az alábbi:

*Diszjunkt(s)  $\Leftrightarrow (\forall c_1, c_2 : c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow \text{Metszet}(c_1, c_2) = \{\})$*

*KimerítőFelosztás(s, c)  $\Leftrightarrow (\forall i : i \in c \Leftrightarrow \exists c_1, c_2 : c \in s \wedge i \in c_1 \wedge i \in c_2)$*

*Partíció(s, c)  $\Leftrightarrow \text{Diszjunkt}(s) \wedge \text{KimerítőFelosztás}(s, c)$*

Kategóriákat úgy is *definiálhatunk*, hogy megadjuk a tagságuk elégsges és szükséges feltételeit. Például egy agglegény egy felnőtt, nem házas férfi:

$x \in \text{Agglegény} \Leftrightarrow \text{NemHázas}(x) \wedge x \in \text{Felnőtt} \wedge x \in \text{Férfi}$

Ahogy erről a természetes fajtákról szóló rövid kitérőben szó lesz, a kategóriák szigorú logikai definíciója nem minden lehetséges, és nem is minden szükséges.

## Fizikai összetétel

Az a gondolat, hogy egy objektum része lehet egy másik objektumnak, nem újkeletű. Valakinek az orra része a fejének, Románia Európa része, ez a fejezet a könyünk egy része. Hogy megmondhassuk, hogy egy dolog része egy másiknak, egy általános *Rész*

relációt fogunk használni. Az objektumokat tehát *Része* hierarchiába lehet szervezni, ami hasonlít a *Részjelmező* hierarchiára:

*Része(Bukarest, Románia)*  
*Része(Románia, KeletEurópa)*  
*Része(KeletEurópa, Európa)*  
*Része(Európa, Föld)*

A *Része* reláció tranzitív és reflexív, azaz:

$$\begin{aligned} \text{Része}(x, y) \wedge \text{Része}(y, z) &\Rightarrow \text{Része}(x, z) \\ \text{Része}(x, x) \end{aligned}$$

Ebből adódóan kikövetkeztethető, hogy *Része(Bukarest, Föld)*

Az összetett objektumok (*composite object*) kategóriáit gyakran ezen objektumok részei között értelmezett strukturális relációkkal jellemzzük. Például egy kétlábúnak pontosan két lába van, ami egy testhez van rögzítve:

$$\begin{aligned} \text{KétLábú}(a) &\Rightarrow \exists l_1, l_2, b \ Láb(l_1) \wedge Láb(l_2) \wedge \text{Test}(b) \wedge \text{Része}(l_1, a) \wedge \text{Része}(l_2, a) \\ &\quad \wedge \text{Része}(b, a) \wedge \text{Rögzített}(l_1, b) \wedge \text{Rögzített}(l_2, b) \wedge l_1 \neq l_2 \\ &\quad \wedge [\forall l_3 \ Láb(l_3) \wedge \text{Része}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)] \end{aligned}$$

A „pontosan kettő” jelölés egy kicsit fura. Kénytelenek vagyunk kijelenteni, hogy két láb van, és ezek nem azonosak, és ha valaki egy harmadik lábbal jön elő, annak azonosnak kell lennie a kettő valamelyikével. A 10.6. alfejezetben látni fogjuk, hogy a lefró logikának nevezett formalizmus a „pontosan kettő” típusú korlátozásokat egyszerűbben fejezi ki.

A kategóriáakra vonatkozó *Partíció* reláció mintájára egy *RészPartíció* relációt definiálhatunk (lásd 10.6. feladat). Egy objektum a *RészPartíció*-jában felsorolt részeiből áll, és bizonyos tulajdonságaira ezekből a részekből lehet következtetni. Így például egy összetett objektum tömege a részeihez tartozó tömegek összege. Jegyezzük meg, hogy ez a kategóriáakra nem vonatkozik: egy kategóriának nincs tömege annak ellenére, hogy az elemeinek lehet tömege.

Hasznos olyan összetett objektumokat is definiálni, amelyeknek meghatározott részei vannak, de konkrét struktúrájuk nincsen. Előfordulhat, hogy azt szeretnénk mondani, hogy: „A zacskóban 3 kg alma van.” Kísértést érezhetnénk, hogy a zacskóban-alma *halmazhoz* súlyt rendeljük, ez azonban hiba lenne, mert egy halmaz absztrakt matematikai fogalom, aminek vannak elemei, de súlya nincs. Ahelyett egy új fogalomra van szükségünk, amit **kötegnek** (*bunch*) fogunk nevezni. Ha az almák például az *Alma*<sub>1</sub>, az *Alma*<sub>2</sub> és az *Alma*<sub>3</sub>, akkor a:

$$\text{Köteg}(\{\text{Alma}_1, \text{Alma}_2, \text{Alma}_3\})$$

a három almából (mint részből, de nem mint elemből) álló összetett objektumot jelöli. A köteget egy közönséges, bár nem strukturált objektumként használhatjuk. Jegyezzük meg, hogy  $\text{Köteg}(\{x\}) = x$ . Továbbá, hogy a  $\text{Köteg}(Alma)$  az összes almából álló összetett objektum, amit az *Alma* kategóriával összetéveszteni nem szabad.

A *Köteg*-et a *Része* relációval tudjuk definíálni. Az  $s$  minden eleme, eleme a *Köteg*( $\{s\}$ )-nek is:

$$\forall x \ x \in s \Rightarrow \text{Része}(x, \text{Köteg}(s))$$

Továbbá, a *Köteg(s)* a legkisebb objektum, amely ezt a feltételt teljesíti. Más szóval a *Köteg(s)* részének kell lennie minden olyan objektumnak, amely az *s* összes elemét részeként tartalmazza:

$$\forall y \ [\forall x \ x \in s \Rightarrow Része(x, y)] \Rightarrow Része(Köteg(s), y)$$

Ezek az axiómák a **logikai minimalizálásnak (logical minimization)** nevezett általános módszer egy példáját jelentik. A logikai minimalizálás azt jelenti, hogy egy objektumot bizonyos feltételeket kielégítő legkisebb objektumnak definiálunk.

## Mérések

A világnak mind a tudományos, mind a józan ész elméleteiben az objektumoknak magassága, tömege, ára van és így tovább. Az ezekhez a tulajdonságokhoz előírt értékek a **mértékek (measures)**. Közönséges, kvantitatív mértékeket könnyű reprezentálni. Képzeljük el, hogy az univerzum absztrakt „mértékobjektumokat” tartalmaz, mint például a *hossz*, ami az itt látható vonalszegmens hossza: ——————. Nevezhetjük ezt a hosszt 1,5 hüvelyknek vagy 3,81 cm-nek. Ugyanannak a hossznak tehát a nyelvünkben több, különböző neve is lehet. Logikailag ez úgy lehetséges, hogy egy egységfüggvényt (*unit function*) egy számmal kombinálunk. (Egy alternatív sémaival a 10.8. feladatban foglalkozunk.) Ha  $L_1$  a vonalszegmens neve, akkor azt írhatjuk, hogy:

$$Hossz(L_1) = Hüvelyk(1,5) = Centiméter(3,81)$$

Az egységek közötti konverziót olyan állításokkal lehet megoldani, amelyek az egyik egység többszörösét a másik egységgel teszik azonossá:

$$Centiméter(2,54 \times l) = Hüvelyk(l)$$

Hasonló axiómákat a fontokra és kilogrammokra, a másodpercekre és a napokra, a dollárakra és a centekre is felírhatunk. A mértékekkel az objektumokat az alábbi módon tudjuk leírni:

$$\text{Átmérő}(KosárLabda_{12}) = Hüvelyk(9,5)$$

$$\text{Ár}(KosárLabda_{12}) = \$19)$$

$$d \in \text{Napok} \Rightarrow \text{Tartam}(d) = \text{Óra}(24)$$

Jegyezzük meg, hogy a  $\$(1)$  nem egy egypénti bankó! Az egypénti bankóból lehet kettő, de a  $\$(1)$  nevű objektumból csak egy van. Jegyezzük meg azt is, hogy míg a *Hüvelyk(0)* és *Centiméter(0)* ugyanarra a zérushosszra hivatkozik, más zérusmértékkel, mint például a *Másodperc(0)* nem azonosak.

Egyszerű, kvantitatív mértékeket reprezentálni könnyű. Más mértékek több problémát okoznak, mert nincs hozzájuk elfogadott értékskála. A gyakorlat nehéz, a desszert finom és a vers szép, azonban e kvalitásokhoz nehéz számokat rendelni. Valaki megkísérelhetné az ilyen tulajdonságokat teljesen elutasítani, mint olyanokat, melyeknek a logikai következetet szempontjából nincs hasznuk, vagy – ami még rosszabb – megkísérelhetne a szépségre egy numerikus skálát rákényszeríteni. Ez súlyos hiba lenne, mert ilyen lépésre nincs is szükség. A mértékek legfontosabb tulajdonsága nem az,

hogy valamilyen konkrét numerikus értékkal rendelkeznek, hanem, hogy *rendezetek*.

Annak ellenére, hogy a mértékek nem számok, összehasonlításukra a  $>$  rendező szimbólumot fogjuk használni. Így például hihetjük azt, hogy a Norvig írta feladatok nehezebbek, mint azok, amiket Russell írt, és hogy egy nehezebb feladatnál kevesebb pontot lehet elérni:

$$e_1 \in \text{Feladatok} \wedge e_2 \in \text{Feladatok} \wedge \text{Írta}(Norvig, e_1) \wedge \text{Írta}(Russell, e_1) \Rightarrow$$

$$\text{Nehézség}(e_1) > \text{Nehézség}(e_2)$$

$$e_1 \in \text{Feladatok} \wedge e_2 \in \text{Feladatok} \wedge \text{Nehézség}(e_1) \wedge \text{Nehézség}(e_2) \Rightarrow$$

$$\text{VárhatóEredmény}(e_1) < \text{VárhatóEredmény}(e_2)$$

Ez elég is ahhoz, hogy valaki eldönthesse, melyik feladattal érdemes foglalkoznia, bár mennyire nem használtunk semmiféle numerikus értéket a nehézség kifejezésére. (Azért azt meg kell tudnunk állapítani, hogy melyik feladatot ki írta.) A mértékek közötti efféle monoton reláció az alapja a **kvalitatív fizika** (**qualitative physics**) területének. Ez az MI egy részterülete, amely azt vizsgálja, hogy hogyan lehetne a fizikai rendszerekre következtetni anélkül, hogy a részletes egyenletekbe és a numerikus szimulációkba belebonyolódnánk. A kvalitatív fizikával a történeti megjegyzésekben foglalkozunk.

## Természetes fajták

Egyes kategóriáknak szigorú definíciói vannak. Egy objektum akkor és csak akkor háromszög, ha egy háromoldalú sokszög. A valódi világban viszont a kategóriák többsége **természetes fajtájú** (**natural kind**) anélkül, hogy bármilyen letisztult definíciója lenne. Tudjuk például, hogy a paradicsomok mélyvörös színűek és nagyból gömbösrűek szoktak lenni, a tetejükön, ahol a szárú csatlakozott, egy kis bemélyedéssel rendelkeznek, nagyból 5–8 cm átmérőjük, és vékony, de erős bőrrel, belül héossal, magvakkal és lével rendelkeznek. Azt is tudjuk azonban, hogy eltérések is vannak. Egyes paradicsomok narancsszínűek, az érettel paradicsom zöld, egyesek kisebbek, illetve nagyobbak, mint az átlag, a miniparadicsomok egységesen kisméretűek. Ahelyett hogy a paradicsomok tökéletes definíciójára törekednénk, inkább egy tulajdonsághalmazzal rendelkezünk, amely arra szolgál, hogy azonosítunk bizonyos objektumokat, amelyek nyilvánvalónan tipikus paradicsomok, más objektumok esetén azonban lehet, hogy a felismerés csödöt mond. (Lehet egy paradicsom szörös, mint egy ószibarack?)

Egy logikai ágens részére ez problémát jelent. Az ágens nem lehet biztos abban, hogy az általa érzékelt objektum valóban egy paradicsom, és ha mégis biztos lenne benne, nem tudná eldönteni, hogy a tipikus paradicsom tulajdonságai közül az érzékelt objektum melyekkel rendelkezik. Ez a probléma egyenes következménye annak, hogy az ágens egy hozzáérhetetlen környezetben működik.

Egy hasznos megközelítés szétválasztani minden, ami a kategória minden egyedére igaz, attól, ami csupán a kategória tipikus egyedeire igaz. A *Paradicsom* kategória mellett szükség van még a *Tipikus(Paradicsom)* kategóriára. A *Tipikus* függvény egy kategóriát egy olyan alosztályra képezi le, amely csak tipikus egyedekből áll:

$$\text{Tipikus}(c) \subseteq c$$

A természetes fajtára vonatkozó tudás többsége tulajdonképpen tipikus egyedekről szól

$$x \in \text{Tipikus}(\text{Paradicsom}) \Rightarrow \text{Piros}(x) \wedge \text{Gömbölyű}(x)$$

Ily módon képesek vagyunk a kategóriára vonatkozó fontos tényeket feljegyezni anélkül, hogy pontos definíciókat kellene megadnunk.

Azzal, hogy a természetes kategóriák többsége esetén az egzakt definíció megalkotása nehézségekbe ütközik, részletesen Wittgenstein foglalkozott a *Philosophical Investigations* c. művében

(Wittgenstein, 1953). A játékok példáját használta fel, hogy kimutassa, a kategória egyedei inkább egyfajta „családi hasonlóságban” és nem valami szükséges és elégsges jellegzetességekben osztoznak.

Quine szintén támadt a szigorú definíció hasznosságát (Quine, 1953). Azt mutatta ki, hogy az „aggregény” definíciója, mint egy nem házas felnőtt férfi is gyanús. Valaki megkerdőjelezhetné például, hogy „A Pápa aggregény” értelmes állítás-e. Bár nem kifejezetten *hamis*, ez a fogalomhasználat mégis szerencsétlen, mert az olvasónak nem szándékolt következtetések levonását teszi lehetővé. A feszültséget fel lehetne oldani megkülönböztetést téve a tudásreprezentációban való belső használatra alkalmas logikai definíció és a helyes nyelvészeti használat jobban árnyalt kritériumai között. A másik megkapható az elsőből a levont következtetések szűrésével. Az is lehetséges, hogy a nyelvészeti használat kudarcai visszacsatolásként hatnak a belső definíció módosítása érdekében, fölöslegessé téve a szűrést.

## Szubsztanciák és objektumok

A világot lehetne úgy szemlélni, hogy primitív objektumokból (rézecskékből) és az azokból felépülő összetett objektumokból áll. Az olyan nagy objektumok, mint például az almák vagy az autók szintén végzett következtetéssel meg tudunk birkózni a primitív objektumok óriási számából eredő bonyolultsággal. A valóság tekintélyes része azonban az **egyedesítés (individuation)** – az elkülönülő objektumokra való felbontásnak – látszólag ellenáll. A valóságnak ezt a részét az **anyag (stuff)** általános névvel fogjuk illetni. Példaképpen tételezzük fel, hogy van előttem egy malac és egy kevés vaj.<sup>2</sup> Azt mondhatom, hogy malacból egy van, de a „vaj objektum” nem megszámlálható, hiszen akármilyen része a vaj objektumnak szintén vaj objektum, legalábbis amíg az igazán kicsi részekhez el nem jutunk. Ez az anyag és a dolgok közötti fő különbség. Sajnos nem lesz két malacunk, ha a malacot kettésszeljük.

Vegyük észre, hogy a magyar nyelv az anyag és a dolgok között különbséget tesz.<sup>3</sup> Azt mondjuk „egy malac”, de bizonyos éttermi vagy bolti gyakorlattól eltekintve, nem mondhatjuk azt, hogy „egy vaj”. A nyelvészek különbséget tesznek a **megszámlálható főnevek (count nouns)**, mint például malacok, gödrök vagy tételek, és a **nem megszámlálható főnevek (mass nouns)**, mint például vaj, víz és energia között. Néhány versengő ontológiáról azt állítják, hogy e különbségeket képes kezelni. Az egyiket itt írjuk le, a többiről a történeti feljegyzésekben lehet olvasni.

Hogy az anyagot jól reprezentálhassuk, egy nyilvánvaló dologgal kezdünk. Ontológiánkban az anyag nagyobb kötegeit kell tudnunk objektumként kezelni. Például a vajban felismerjük azt a vajat, amit tegnap este az asztalon hagyunk, esetleg felemelhetjük, megmérhetjük, eladhatjuk vagy akármi másra csinálhatunk vele. Ilyen értelemben ez egy objektum, pontosan olyan, mint egy malac. Nevezzük *Vaj<sub>3</sub>*-nak. Definiálni fogjuk a *Vaj* kategóriát is. Ennek elemei, informálisan, mindenkor az objektumok, amikre azt lehet mondani: „Ez vaj”, a *Vaj<sub>3</sub>*-at is beleérte. A nagyon kicsi, de most elhanyagolt alkotórészek szerepére vonatkozó intéről nem megfeledkezve, a vaj objektum minden része szintén vaj objektum:

$$x \in Vaj \wedge Része(y, x) \Rightarrow y \in Vaj$$

<sup>2</sup> A szövegben eredetileg „aardvark” (földimalac) szerepel, ami egy malacszerű dél-afrikai állatfajta. (A ford.)

<sup>3</sup> A szövegben eredetileg természetesen angol példa szerepel – „an aardvark” és „a butter/butter”. (A ford.)

Most már mondhatjuk, hogy vaj kb. 30 foknál olvad meg:

$$\text{Vaj}(x) \Rightarrow \text{Ol vadásiPont}(x, \text{Celsius}(30))$$

A vaj sárga, kevésbé sűrű, mint a víz, szobahőmérsékleten puha, magas zsírtartalmú stb. Másrészt a vajnak nincs konkrét nagysága, alakja vagy súlya. Specializáltabb kategóriákat is definiálhatunk, mint például *NemSózottVaj*, ami szintén egy anyag, mert a nem sózott vaj objektum akármilyen része szintén nem sózott vaj objektum. Ha azonban az *EgyKilóVaj* kategóriát definiáljuk, amely az összes, egy kiló súlyú vaj objektumot tartalmazza, nincs többé anyagunk! Ha egy kiló vajat kettévágunk, nem kapunk kétszer egy kiló vajat – ami a valódi világ azon bosszantó dolgainak egyike, amivel sajnos együtt kell elni.

Valójában arról van szó, hogy vannak ún. **belső (intrinsic) tulajdonságok**: ezek inkább magához az objektum szubsztanciájához tartoznak, mint az objektum egészéhez. Ha valamit kettévágunk, a részei a belső tulajdonságukat megtartják – legyen ez sűrűség, forráspont, íz, szín, a tulajdonos stb. A **külső (extrinsic) tulajdonságok** éppen az ellenkezőt jelentik: olyan tulajdonságokat, mint a súlyt, hosszat, alakot, funkciót stb., amelyeket a részekre bontásnál megtartani nem lehet.

Az objektumok olyan osztálya, amelyek definíciójában csakis *belső* tulajdonságok szerepelnek, a szubsztanciák, illetve a nem megszámlálható főnevek osztálya. Az az osztály, amelynek definíciójában *bármilyen* külső tulajdonság is szerepel, a megszámlálható főnevek osztálya. Az *anyag* kategória a legáltalánosabb szubsztanciakategória, egyetlen belső tulajdonsága sincs. A *dolog* a diszkrét objektumok legáltalánosabb kategóriája, egyetlen külső tulajdonsága sincs. minden fizikai objektum minden kategóriába tartozik, így a kategóriák együtt léteznek – ugyanazokra az entitásokra utalnak.

## 10.3. CSELEKVÉSEK, SZITUÁCIÓK ÉS ESEMÉNYEK

A cselekvések eredményeiről következtetni kulcsfontosságú egy tudásalapú ágens számára. A 7. fejezetben példákat láttuk olyan ítéletállításokra, amelyek leírják, hogy a cselekvések hogyan hatnak a wumpus világra. A 283. oldalon lévő (7.3) egyenlet azt állítja például, hogy az ágens lokációja hogyan változik az ágens mozgásával. Az ítéletkalkulus egyik hátránya, hogy cselekvés leírásáról külön másolatokkal kell rendelkeznünk azokra az időpontokra, amikor a cselekvést végre fogják hajtani. Ebben az alfejezetben leírt, az elsőrendű logikára alapozó reprezentációs módszer ezt a problémát elkerüli.

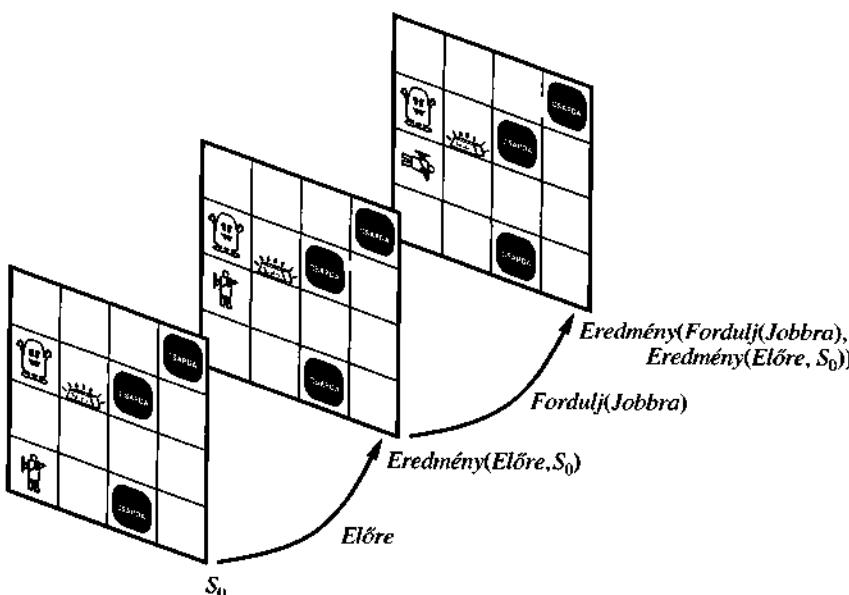
### A szituációkalkulus ontológiája

Az axiómák többszörös másolatainak az elkerülésére nyilvánvaló módszer az idő szerinti kvantifikálás – azaz azt mondani, hogy „ $\forall t$ -re a  $t$ -beli cselekvésnek ez a  $t + 1$ -beli eredménye”. A  $t + 1$  típusú explicit időpillanatok helyett ebben az alfejezetben szituációkkal dolgozunk, amelyek a cselekvések végrehajtásából adódó állapotok jelölései. E megközelítés neve **szituációkalkulus (situation calculus)** és az alábbi ontológiára támaszkodik:

- A 8. fejezethez hasonlóan a cselekvések logikai termek, például az *Előre*, a *Fordulj(Jobbra)*. Egyelőre feltételezzük, hogy a környezetben csak egy ágens tartózkodik. (Ha több lenne, egy további argumentummal megjelölhetjük, hogy melyik ágens hajtja végre a cselekvést).
- A **situációk** (*situations*) logikai termek, amelyekhez tartozik egy (általában  $S_0$ -nak jelölt) kezdeti szituáció és minden más szituáció, amely a szituációra vonatkozó cselekvés végrehajtásából adódott. Ha az a cselekvést az  $s$  szituációban hajtották végre, az eredményezett szituációt *Eredmény(a, s)* (néha *Csinálás*-nak nevezett) függvény nevezi meg. A gondolatot a 10.2. ábra illusztrálja.
- A **folyó események** (*fluents*) olyan függvények és predikátumok, amelyek szituációról szituációra változnak. Ilyen például az ágens lokációja vagy a wumpus jóléte. A szótár szerint a folyó valamelyen folyadékszerű viselkedésre utal. Használhatunkban a szituációk menti lefolyását vagy változását jelenti.  $\neg \text{Tart}(G_1, S_0)$  például azt mondja, hogy az ágens a kezdeti  $S_0$  szituációban a  $G_1$  aranyat nem tartja a kezében. *Kor(Wumpus, S<sub>0</sub>)* a wumpus  $S_0$ -beli életkorára vonatkozik.
- Engedélyezzük az **időtlen**, **örök** (*atemporal, eternal*) predikátumokat és függvényeket is. Példaként tekinthetjük az *Arany(G<sub>1</sub>)* vagy a *BalLába(Wumpus)* predikátumokat.

Az egyedi cselekvéseken túlmenően hasznos, ha a cselekvések sorozatairól is tudunk következtetni. A sorozat eredményét az egyedi cselekvések eredménye alapján definiálhatjuk. Először azt fogjuk mondani, hogy egy üres sorozat végrehajtása a szituációt változatlanul meghagyja:

$$\text{Eredmény}([], s) = s$$



10.2. ábra. Szituációkalkulusban minden szituáció (az  $S_0$ -t kivéve) valamilyen cselekvés eredménye

Egy nem üres sorozat végrehajtása nem más, mint az első cselekvés végrehajtása, majd az eredményül adódó szituációban a maradó sorozatnak a végrehajtása:

$$\text{Eredmény}([a]\text{sorozat}, s) = \text{Eredmény}(\text{sorozat}, \text{Eredmény}(a, s))$$

Egy szituációkalkulus ágensnek tudnia kellene egy adott cselekvéssorozat eredményét kikövetkeztetni, ez az ún. **előrevetítő feladat** (*projection task*). A megfelelő konstruktív következtetési mechanizmus birtokában képesnek kellene lennie arra is, hogy egy kívánatos eredményt biztosító sorozatot *megtaláljon*, ez az ún. **tervkészítési feladat** (*planning task*).

Egy módosított wumpus világ ról fogunk példát venni, ahol az ágens orientációjával nem törődünk, és ahol az ágens egy helyről egy szomszédos helyre *Megy*. Tegyük fel, hogy ágens az [1, 1]-nél és arany az [1, 2]-nél van. A cél az aranyat az [1, 1] helyen birtokoljuk. A folyó esemény predikátumok a *Nála(o, x, s)* és *Tart(o, s)*. A kezdeti tudásbázis az alábbi leírást tartalmazhatná:

$$\text{Nála}(\text{Ágens}, [1, 1], S_0) \wedge \text{Nála}(G_1, [1, 2], S_0)$$

Ez azonban még nem elég, mert nem mondta, hogy az  $S_0$ -ban mi nem igaz (a probléma további elemzését lásd a 425. oldalon). A teljes leírás az alábbi:

$$\begin{aligned} \text{Nála}(o, x, S_0) &\Leftrightarrow \wedge [(o = \text{Ágens} \wedge x = [1, 1]) \vee (o = G_1 \wedge x = [1, 2])] \\ &\neg \text{Tart}(o, S_0) \end{aligned}$$

Arra is szükség van, hogy kijelentsük,  $G_1$  egy arany, és [1, 1] és [1, 2] szomszédosak:

$$\text{Arany}(G_1) \wedge \text{Szomszédos}([1, 1], [1, 2]) \wedge \text{Szomszédos}([1, 2], [1, 1])$$

Valaki bizonyára azt szeretné bebizonyítani, hogy ágens a célját eléri, ha az [1, 2]-re átmegy, ott megfogja az aranyat és az [1, 1]-re visszatér. azaz:

$$\text{Nála}(G_1, [1, 1], \text{Eredmény}([\text{Megy}([1, 1], [1, 2]), \text{Megfog}(G_1), \text{Megy}([1, 2], [1, 1])], S_0))$$

Annál érdekesebb lehetőség az arany birtoklását biztosító tervet készíteni, amit a „mi-lyen cselekvéssorozat eredménye az arany az [1, 1]-en?” kérdés megválaszolásával lehet megtenni.

$$\exists \text{sorozat } \text{Nála}(G_1, [1, 1], \text{Eredmény}(\text{sorozat}, S_0))$$

Nézzük, hogy e kérdések megválaszolásához mivel kellene a tudásbázist kiegészíteni.

## Cselekvések leírása a szituációkalkulusban

A szituációkalkulus legegyszerűbb változatában minden cselekvést két axiómával lehet leírni. A **lehetőségi axióma** (*possibility axiom*) megmondja, hogy a cselekvést mikor lehet elvégezni, a **hatásaxióma** (*effect axiom*) pedig azt, hogy a cselekvés végrehajtásával mi is fog történni. Annak jelölésére, hogy az  $s$  szituációban az  $a$  cselekvés végrehajtása lehetséges, a *Lehet(a, s)* predikátumot fogjuk használni. Az axiómák formája az alábbi:

**LEHETŐSÉGI AXIÓMA:**  $\text{Előfeltételek} \Rightarrow \text{Lehet}(a, s)$

**HATÁSAXIÓMA:**  $\text{Lehet}(a, s) \Rightarrow A \text{ cselekvés végrehajtásából adódó változások}$

Ezeket az axiómákat a módosított wumpus világra fogjuk felírni. A mondatok rövidsége érdekében el fogjuk hagyni az egész mondatra vonatkozó univerzális kvantorokat. Feltételezzük, hogy az  $s$  változó a szituációkra, az  $a$  változó a cselekvésekre, az  $o$  változó az objektumokra (az ágenseket beleértve), a  $g$  változó az aranya. az  $x$  és az  $y$  változók pedig a helyre vonatkoznak.

Az erre a világra vonatkozó lehetőségi axióma azt mondja, hogy egy ágens a szomszédos helyek között mozoghat, az aktuális helyen megfoghatja az aranyat és elengedheti az aranyat, amelyet tartott:

$$\begin{array}{ll} \text{Nála(Ágens, } x, s) \wedge \text{Szomszédos}(x, y) & \Rightarrow \text{Lehet(Megy}(x, y), s) \\ \text{Arany}(g) \wedge \text{Nála(Ágens, } x, s) \wedge \text{Nála}(g, x, s) & \Rightarrow \text{Lehet(Megfog}(g), s) \\ \text{Tart}(g, s) & \Rightarrow \text{Lehet(Elenged}(g), s) \end{array}$$

A hatásaxióma azt állítja, hogy ha egy cselekvés lehetséges, akkor a cselekvés végrehajtásából adódó szituációban bizonyos tulajdonságok (folyó események) érvényesek lesznek. Az  $x$ -ről az  $y$ -ra menni cselekvés eredménye az  $y$ -ban tartózkodni, az aranyat megfogni eredménye az aranyat tartani, végül az aranyat elengedni eredménye az aranyat nem tartani:

$$\begin{array}{ll} \text{Lehet(Megy}(x, y), s) & \Rightarrow \text{Nála(Ágens, } y, \text{Eredmény}(Megy}(x, y), s)) \\ \text{Lehet(Megfog}(g), s) & \Rightarrow \text{Tart}(g, \text{Eredmény}(Megfog}(g), s)) \\ \text{Lehet(Elenged}(g), s) & \Rightarrow \neg \text{Tart}(g, \text{Eredmény}(Elenged}(g), s)) \end{array}$$

Az axiómák megállapításával tudjuk-e bizonyítani, hogy a tervünk biztosítja a cél elérését? Sajnos még nem! Először minden jól működik: a  $\text{Megy}([1, 1], [1, 2])$  valójában lehetséges az  $S_0$ -ban, és a  $\text{Megy}$  hatásaxiomája meggyőzhet minket, hogy az ágens tényleg eléri az  $[1, 2]$ -t:

$$\text{Nála(Ágens, } [1, 2], \text{Eredmény}(Megy}([1, 1], [1, 2]), S_0))$$

Most tekintsük a  $\text{Megfog}(G_1)$  cselekvést. Ki kell mutatni, hogy ez lehetséges az új szituációban, azaz:

$$\text{Nála}(G_1, [1, 2], \text{Eredmény}(Megy}([1, 1], [1, 2]), S_0))$$

Sajnos a tudásbázisunkban az ilyen konklúziót semmi sem támasztja alá. Intuitíve persze megérjük, hogy az ágens  $\text{Megy}$  cselekvése az arany helyzetére nincs hatással, így az még mindig az  $[1, 2]$ -ben van, ahogy ott volt az  $S_0$  szituációban is. A probléma az, hogy a hatásaxióma kijelenti, hogy mi változik, de nem mond semmit arról, hogy mi nem változik.

A változatlanul megmaradó dolgok reprezentálása az ún. **keretprobléma (frame problem)**. A keretproblémára hatékony megoldást kell találnunk, mert a valódi világban az idő többségében majdnem minden változatlan marad. minden egyes cselekvés a folyó eseményeknek csak kis töredékét befolyásolja.

Az egyik megközelítés az **explicit keretaxiomák (frame axioms)** felírása, amelyek azt mondják, hogy mi marad változatlan. Az ágens mozgásai például más objektumok helyzetét változatlanul hagyják, hacsak azokat az ágens nem tartja magánál:

$$\text{Nála}(o, x, s) \wedge (o = \text{Ágens}) \wedge \neg \text{Tart}(o, s) \Rightarrow \text{Nála}(o, x, \text{Eredmény}(Megy}(y, z), s))$$

Ha összesen  $F$  folyó esemény és  $A$  cselekvés lenne, akkor  $O(AF)$  keretaxiomára lenne szükségünk. Másfelől, ha minden cselekvésnek legfeljebb  $E$  hatása lenne, ahol az  $E$  tipikusan jóval kisebb, mint  $F$ , akkor a történeteket sokkal kisebb,  $O(AE)$  nagyságú



tudásbázissal is ki tudnánk fejezni. Ez a **reprezentációs keretprobléma** (representational frame problem). A vele szorosan kapcsolódó **következtetési keretprobléma** (inferential frame problem) egy *t*-lépéses cselekvéssorozat eredményeinek  $O(Et)$  időben, és nem  $O(Ft)$  vagy  $O(AEt)$  időben való kivétítése. Sorban egymás után az összes problémával fogunk foglalkozni. Azonban egy probléma mégis marad – biztosítani, hogy egy cselekvés sikeres elvégzéséhez minden feltételet sikerült megfogalmazni. Így például a *Megy* kudarccal fenyeget, ha az ágens *útközben* meghal. Ez az ún. **kvalifikációs probléma** (qualification problem), melynek teljes megoldása nincs.

## A reprezentációs probléma megoldása

A reprezentációs probléma megoldása csak egy kis változást igényel abban, hogy az axiómákat hogyan írjuk fel. Ahelyett hogy felírnánk minden cselekvés hatását, azal foglalkozunk inkább, hogy egy folyó esemény időben hogyan evolvál.<sup>4</sup> Az általunk használt axiómákat **követő állapot** axiómáknak (successor-state axioms) hívják. Alakjuk a következő:

**KÖVETŐ-ÁLLAPOT AXIÓMA:**

*Cselekvés lehetséges*  $\Rightarrow$

*(Folyó esemény igaz az eredmény állapotban  $\Leftrightarrow$  A cselekvés hatása igazzá tette  
 $\vee$  Igaz volt a cselekvés előtt  
 és a cselekvés nem változtatta)*

Feltételezve, hogy lehetetlen cselekvésekkel nem foglalkozunk, vegyük észre, hogy a definícióban  $\Leftrightarrow$  szerepel  $\Rightarrow$  helyett. Ez azt jelenti, hogy a folyó esemény *akkor* és *csak akkor* lesz igaz, ha a jobb oldali rész igaz. Más szóval minden folyó eseménynek a következő állapotban érvényes igazságértékét a cselekvésnek és az aktuális állapotban érvényes értékének a függvényeként határozzuk meg. Ez azt jelenti, hogy a következő állapot teljesen meghatározott az aktuális állapottól kiindulva, és így pótlagos keretaxió-mákra nincs szükség.

Az ágens helyére érvényes követő állapot axióma azt mondja, hogy ágens egy cselekvés után az y-ban lesz, ha a cselekvés lehetséges, és az az y-ba való mozgásból áll, vagy ha az ágens már az y-ban tartózkodott, és a cselekvésnek a mozgáshoz nem volt köze:

*Lehet(a, s)  $\Rightarrow$*

*(Nála(Agens, y, Eredmény(a, s))  $\Leftrightarrow$  (a = Megy(x, y))  
 $\vee$  (Nála(Agens, y, s)  $\wedge$  (a  $\neq$  Megy(y, z)))*

A *Tart* axiómája azt fejezi ki, hogy ágens tarja a g-t, ha a cselekvés a g megfogását jelentette, és a cselekvés lehetséges volt, vagy ha az ágens már tartotta a g-t és a cselekvés nem annak elengedése volt:

*Lehet(a, s)  $\Rightarrow$*

*(Tart(g, Eredmény(a, s))  $\Leftrightarrow$  (a = Megfog(g))  
 $\vee$  (Tart(g, s)  $\wedge$  (a  $\neq$  Elenged(g))))*

<sup>4</sup> Ez lényegében az a megközelítés, amit a 7. fejezetben a logikai áramkör-ágens tervezésénél alkalmaztunk. Az olyan axiómákat, mint a (7.4) és a (7.5), valóban követő állapot axiómáknak lehet nézni.

A követő állapot axiómák a reprezentációs keretproblémát megoldják, mert az axiómák összszáma  $O(AE)$  literál. Az  $E$  hatások és az  $A$  cselekvések mindegyike pontosan egyszer kerül említésre. A literálok  $F$  különböző axióma között vannak szétosztva, így egy axióma átlagos nagysága  $AE/F$ .

Az éles szemű olvasó észreveszi, hogy ezek az axiómák a *Nála* folyó eseményt kezelik az ágens számára, de nem így az arany esetében. Még mindig nem tudjuk bebizonyítani, hogy ez a háromlépéses terv eljut a célhöz, az arany eléréséhez az [1, 1]-ben. Amire szükségünk lenne még, az annak a kijelentése, hogy az ágens  $x$ -től  $y$ -ig való mozgásának **implicit hatása (implicit effect)**, hogy minden, általa megfogott arany is vele fog mozogni (ahogy az aranyon a hangya, a hangyán a bacillus stb.). Az implicit hatások kezelését **ramifikációs problémának (ramification problem)** nevezzük. A problémát általánosságban később vitatjuk meg. A jelenlegi speciális tárgyterületen e probléma a *Nála* számára kissé általánosabb követő állapot axiómák felírásával megoldható. Az új axiómák, amelyek a régieket magukban foglalják, azt fejezik ki, hogy minden  $o$  objektum  $y$ -ban tartózkodik, ha az ágens az  $y$ -ba ment, és az  $o$  maga az ágens, vagy bármi, amit az ágens tart; vagy ha az  $o$  már az  $y$ -ban volt, és az ágens nem ment sehol, miközben az  $o$  maga az ágens, vagy bármi, amit az ágens tart.

$\text{Lehet}(a, s) \Rightarrow$

$$\begin{aligned} \text{Nála}(o, y, \text{Eredmény}(a, s)) \Leftrightarrow & (a = \text{Megy}(x, y) \wedge (o = \text{Ágens} \vee \text{Tart}(o, s))) \\ & \vee (\text{Nála}(o, y, s) \wedge \neg(\exists z \ y \neq z \wedge a = \text{Megy}(y, z) \wedge \\ & (o = \text{Ágens} \vee \text{Tart}(o, s)))) \end{aligned}$$

Egy további technikai nehézséggel számolunk kell. Az ilyen axiómákat felhasználó következetési eljárásnak képesnek kell lennie az azonosság hiányát kimutatni. A legegyszerűbb eset a két konstans esete, például  $\text{Ágens} = G_1$ . Az elsőrendű logika általános szemantikája lehetővé teszi, hogy a különböző konstansok ugyanazt az objektumot jelentsék, a tudásbázisnak tartalmaznia kell tehát az ilyen eseteket kitiltó axiómát. Az **egyedi elnevezések axiómája (unique names axiom)** a konstansok azonossághiányát fejezi ki, a tudásbázisban létező minden konstanspárra. Ha a tételebizonyító rendszer az azonossághiányt feltételezi, ahelyett hogy explicit módon a tudásbázisban ez le lenne írva, az **egyedi elnevezések feltételezésről (unique names assumption)** beszélünk. Az azonossághiányt ki kell jelentenünk a cselekvésekre is: a  $\text{Megy}([1, 1], [1, 2])$  cselekvés és a  $\text{Megy}([1, 2], [1, 1])$ , vagy a  $\text{Megfog}(G_1)$  cselekvés nem ugyanaz. Először azt fogjuk mondani, hogy a cselekvések minden típusa különböző – semmilyen *Megy* cselekvés nem lehet egy *Megfog* cselekvés. A cselekvésnevek minden pájról:

$$A(x_1, \dots, x_m) \neq B(y_1, \dots, y_n)$$

Most kijelentjük, hogy két, ugyanahhoz a cselekvéshez tartozó cselekvést term ugyanazt a cselekvést jelenti, ha a benne szereplő objektumok minden azonosak:

$$A(x_1, \dots, x_m) = B(y_1, \dots, y_m) \Leftrightarrow x_1 = y_1 \wedge \dots \wedge x_m = y_m$$

Ezeket az állításokat együttesen **egyedi cselekvés axiómáknak (unique action axioms)** nevezzük. A kezdeti állapotleírás, a követő állapot axiómák, az egyedi elnevezések axiómák és az egyedi cselekvésaxiómák együttese elegendő annak bizonyításához, hogy a javasolt terv megvalósítja a célt.

## A következtetési keretprobléma megoldása

Követő állapot axiómák a reprezentációs keretproblémát ugyan megoldják, de a következtetési keretproblémát nem. Tekintsünk egy olyan  $t$ -lépésű  $p$  tervet, hogy  $S_t = \text{Eredmény}(p, S_0)$ . Hogy eldönthessük, mely folyó esemény igaz  $S_t$ -ben, szükséges minden egyes  $F$  keretaxiómáktól minden  $t$  időlépésben kiértékelni. Mivel az axiómáknak átlagosan  $AE/F$  a nagyságuk,  $O(AE)$  következtetéssel kell számolnunk. A munka zöme rá fog menni arra, hogy a folyó eseményeket az egyik szituációról a másikra változatlanul másoljuk át.

A következtetési keretprobléma megoldására két lehetőségünk van. Először is, eldobhatjuk a szituációkalkulust, és kitalálhatjuk az axiómák egészen új felírási módját. Ez történt még az olyan formalizmusokban, mint a **folyó esemény kalkulus (fluent calculus)**. Másodszor, megváltottathatjuk magát a következtetési eljárást, hogy a keretaxiómákat hatékonyabban dolgozza fel. Erre utalást ad magának az egyszerű megközelítésnek az  $O(AE)$  jellege. De miért függ ez a cselekvések  $A$  számától, amikor pontosan tudjuk, hogy mindegyik időlépésben melyik cselekvés kerül végrehajtásra? Hogy belássuk, hogyan tudnánk a dolgokon javítani, nézzük meg először a keretaxiómák formátumát:

$\text{Lehet}(a, s) \Rightarrow$

$$F_i(\text{Eredmény}(a, s)) \Leftrightarrow (a = A_1 \vee a = A_2 \dots) \\ \vee F_i(s) \wedge (a \neq A_3) \wedge (a \neq A_4) \dots$$

Vagyis mindegyik axióma említ tehát néhány cselekvést, amelyek a folyó eseményt igazzá, és néhányat, amelyek azt hamissá teszik. Ezt a körülményt formalizálhatjuk a  $\text{PozHatás}(a, F_i)$  és a  $\text{NegHatás}(a, F_i)$  predikátumok bevezetésével. A  $\text{PozHatás}(a, F_i)$  azt jelenti, hogy az  $a$  cselekvés az  $F_i$ -t igazzá, a  $\text{NegHatás}(a, F_i)$  pedig azt, hogy hamissá teszi. A fenti axióma ezzel a következőképpen írható át:

$\text{Lehet}(a, s) \Rightarrow$

$$F_i(\text{Eredmény}(a, s)) \Leftrightarrow \text{PozHatás}(a, F_i) \vee [F_i(s) \wedge \neg \text{NegHatás}(a, F_i)]$$

$\text{PozHatás}(A_1, F_i)$

$\text{PozHatás}(A_2, F_i)$

$\text{NegHatás}(A_3, F_i)$

$\text{NegHatás}(A_4, F_i)$

Az, hogy ez mennyire automatizálható, a keretaxiómák pontos formátumán múlik. Hogy az ilyen axiómákra alapozva hatékony következtetési eljárást dolgozhassunk ki, három dolgot kell még megtenni:

1. Indexeljük a  $\text{PozHatás}$  és a  $\text{NegHatás}$  predikátumokat az első argumentumaik szerint, hogy amikor adott egy  $t$  időpontban bekövetkező cselekvés, a hatásait  $O(1)$  időben meg lehessen keresni.
2. Indexeljük az axiómákat úgy, hogy amikor világos már, hogy egy cselekvés hatása  $F_i$ ,  $O(1)$  idő alatt ki lehessen keresni az  $F_i$ -re vonatkozó axiómákat. Ekkor a cselekvés hatásai között nem szereplő folyó esemény axiómáival nem is kell törödni.
3. minden szituációt reprezentálunk egy megelőző szituáció plusz egy növekményként. Így, amikor lépésről lépésre nem változik semmi, nem kell semmiféle munkát végeznünk. A régebbi megközelítésben  $O(F)$  munka ment volna rá, hogy minden

folyó esemény esetén a megelőző  $F_i(s)$  feltételezésből  $F_i(Eredmény(a, s))$  feltételezést generálunk.

Minden időlépésnél tehát megnézzük az aktuális cselekvést, megkeressük a hatásait és az igaz folyó események halmazát felfrissítjük. minden időlépében átlagosan  $E$  ilyen frissítéssel kell számolnunk, ami az eredő komplexitásra  $O(Et)$ -t ad. Ez megadja a következetési keretprobléma megoldását.

## Idő- és eseménykalkulus

A szituációkalkulus akkor működik jól, ha egy ágens diszkrét, azonnal lezajló cselekvések hajt végre. Ha cselekvéseknek időtartamuk is van, és egymással át is lapolódhatnak, a szituációkalkulus nehézkessé kezd válni. Emiatt a kérdésekkel egy más megközelítésen belül próbálkozunk, amit eseménykalkulusnak (**event calculus**) fogunk nevezni és amely inkább az időpontokon, mint a szituációkon alapul. (Az „esemény” és a „cselekvés” fogalmakat felváltva is használhatjuk. Egy „esemény” informálisan a cselekvések tágabb osztályát jelenti, az explicit ágenst nélkülöző cselekvéset beleértve. Az eseményeket az eseménykalkulusban könnyebben kezelhetjük, mint a szituációkalkulusban.)

Az eseménykalkulusban a folyó események időpillanatokra és nem szituációkra vonatkoznak, és a kalkulust úgy tervezték, hogy az időintervallumokról is lehessen következtetni. Az eseménykalkulus axióma azt mondja ki, hogy egy folyó esemény igaz egy időpontban, ha a folyó eseményt valamelyen múltbeli esemény kezdeményezte, és időközben a folyó eseményt semmilyen közbenső esemény nem állította le. Az *Incializál* és a *Leállít* relációk a szituációkalkulus-beli *Eredmény* relációhoz hasonló szerepet töltönenek be. Az *Incializál(e, f, t)* azt jelenti, hogy az  $e$  esemény a  $t$  időpontban történő bekövetkezése az  $f$  folyó eseményt igazzá teszi, míg a *Leállít(w, f, t)* jelentése, hogy  $f$  igaz értéke megszűnt. A *Történik(e, t)* azt fogja jelenteni, hogy az  $e$  esemény a  $t$  időpontban történik, a *Levág(f, t, t<sub>2</sub>)*-t pedig arra fogjuk használni, hogy leírhassuk, hogy az  $f$ -et valamelyen esemény a  $t$  és a  $t_2$  időpontok között leállította. Formálisan az axióma a következő:

### ESEMÉNYKALKULUS AXIÓMA:

$$T(f, t_2) \Leftrightarrow \exists e, t \ Történik(e, t) \wedge Incializál(e, f, t) \wedge (t < t_2) \wedge \neg Levág(f, t, t_2)$$

$$Levág(f, t, t_2) \Leftrightarrow \exists e, t_1 \ Történik(e, t_1) \wedge Leállít(e, f, t_1) \wedge (t < t_1) \wedge (t_1 < t_2)$$

A kapott funkcionálitás hasonlít a szituációkalkulusra, azonban az időpontokról és az időintervallumokról is nyilatkozhatunk. Így képesek vagyunk a *Történik(Kikapcs(LámpaKapcs<sub>1</sub>), 1:00)*-val azt mondani, hogy a lámpakapesolót pontosan 1:00-kor kapcsolták ki.

Az eseménykalkulus számos továbbfejlesztését dolgozták ki, hogy a közvetett hatásokkal, a nem zérus időtartalmú eseményekkel, a folyamatosan változó eseményekkel, a nemdeterminisztikus hatásokkal, a kauzális korlátozásokkal és más bonyodalmakkal is tudjanak dolgozni. Egyes kérdésekkel a következő alfejezetben fogunk találkozni. A tiszteség kedvéért megjegyezzük, hogy teljesen elfogadható megoldások máig sem születtek, azonban legyőzhetetlen akadályok sem merültek fel.

## Általánosított események

Egyelőre két fő fogalommal – a cselekvésekkel és az objektumokkal – foglalkoztunk. Ideje most utána nézni annak, hogyan illeszkednek ezek a fogalmak egy olyan befogadó ontológiába, ahol mind a cselekvések, mind az objektumok a fizikai univerzum aspektusának foghatók fel. Egy konkrét univerzumról feltesszük, hogy térbeli és időbeli dimenziója is van. A wumpus világ a kétdimenziós rács által definiált térbeli dimenzióval és diszkrét idővel rendelkezett. A világ térben háromdimenziós és időben egydimenziós,<sup>5</sup> mely dimenziók mindegyike folytonos. Egy általánosított esemény (*generalized event*) a többdimenziós univerzum egy részének – a „ter-idő darabkának” – aspektusaiból áll össze. Ez az absztrakció az eddig látott fogalmak többségét általánosítja, a cselekvéseket, a lokációkat, az időket, a folyó eseményeket és a fizikai objektumokat beleértve. Az általános ötletet a 10.3. ábra szemlélteti. Mostantól kezdve az „esemény” egyszerű fogalommal az általánosított eseményeket fogjuk nevezni.

A második világháború például egy olyan esemény, amely a tér-idő különböző pontjaiban történt meg, ezt a szabálytalan szürke folt jelzi. Az eseményt a részeseményekre (*subevents*)<sup>6</sup> bonthatjuk szét:

*RészEsemény(AngliaiCsata, MásodikVilágHáború)*

Hasonlóképpen a második világháború a 20. század részeseménye:

*RészEsemény(MásodikVilágHáború, HuszadikSzázad)*

A 20. század az időnek egy *intervalluma*. Intervallumok a tér-idő olyan darabkái, amelyek két időpont között az egész teret tartalmazzák. A *Periodus(e)* függvény az *e* eseményt bezáró legkisebb intervallumot jelöli. A *Tartam(i)* egy intervallum által foglalt idő hossza, mondhatjuk tehát, hogy *Tartam(Periodus(MásodikVilágHáború)) > Év(5)*.

Ausztrália egy *hely*, egy darabka rögzített térbeli határokkal. A határok időben változnak, geológiai vagy politikai okoknál fogva. A *Benne* predikátumot fogjuk használni az olyan részesemény reláció megjelölésére, amely akkor áll fenn, ha egy esemény térbeli vetülete *Része* egy másik esemény vetületének:

*Benne(Sydney, Ausztrália)*

A *Hely(e)* függvény az *e* eseményt befogadó legkisebb helyet jelenti.

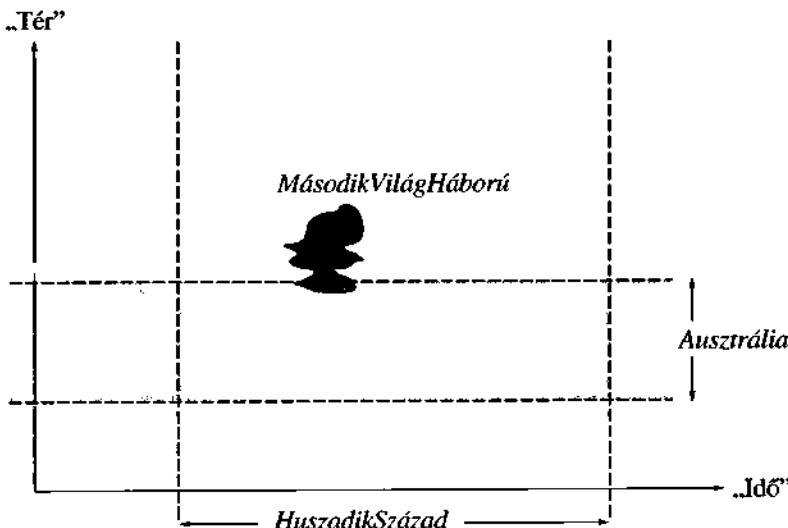
Mint minden más objektumot, az eseményeket szintén lehet kategóriákba szervezni. A *MásodikVilágHáború* például a *Háború* kategóriába tartozik. Azt, hogy Angliában 1640-ben polgárháború zajlott, a következőképpen mondhatjuk:

$\exists w \ w \in \text{PolgárHáború} \wedge \text{RészEsemény}(w, 1640) \wedge \text{Benne}(\text{Lokáció}(w), \text{Anglia})$

Az eseménykategória fogalma segít megválaszolni azt a kérdést, amit igyekeztünk elkerülni, amikor a 10.3. alfejezetben a cselekvések hatásairól beszélünk. Mire hivatkoznak pontosan az olyan logikai termek, mint a *Megy([1, 1], [1, 2])*? Események ezek?

<sup>5</sup> A füzérelméletet tanulmányozó néhány fizikus a 10 vagy több dimenzió mellett foglai állás, mások diszkrét világról beszélnek, a 4-D folytonos tér-idő azonban a józan ész következetési céloknak megfelelő reprezentáció.

<sup>6</sup> Jegyezzük meg, hogy a *RészEsemény* a *Része* reláció speciális esete, és ugyanúgy tranzitív és reflexív.



**10.3. ábra.** Általánosított események. Az univerzumnak térfogati és időbeli dimenziója van. Ezen az ábrán csak egy térfogati dimenziót mutatunk. minden esemény az univerzum Részé. Az olyan esemény, mint a MásodikVilágHáború a tér-idő egy tartományában történik, melynek határai kissé szabadon választhatók, és időben változók. Egy *Intervalum*, mint amilyen a HuszadikSzázad, rögzített és korlátos időbeli, valamint maximális térfogati kiterjedéssel rendelkezik. A *Hely*, amilyen például Ausztrália, nagyjából rögzített térfogati és maximális időbeli kiterjedéssel rendelkezik.

A válasz, talán meglepő, de az, hogy *nem*. Ennek megértéséhez nézzünk meg két „azonos” cselekvést tartalmazó tervet, mint például:

[*Megy*([1, 1], [1, 2]), *Megy*([1, 2], [1, 1]), *Megy*([1, 1], [1, 2])]

Ebben a tervben a *Megy*([1, 1], [1, 2]) nem lehet egy esemény neve, mert két különböző esemény van, amelyek különböző időpontban történnek meg. helyette a *Megy*([1, 1], [1, 2]) egy eseménykategória neve – azoké az eseményeké, amikor az ágens az [1, 1]-ről az [1, 2]-re lép át. A háromlépéses terv azt mondja ki, hogy e három eseménykategória példányai fognak előfordulni.

Jegyezzük meg, hogy ez az első alkalom, amikor a kategóriák megnevezésére komplex termeket vetettünk be, egyszerű konstansszimbólumok helyett. Ez nem jelent új nehézséget. Az argumentumstruktúrát valójában a hasznunkra is fordíthatjuk. Az argumentumok eliminálása az általánosabb kategóriák bevezetését teszi lehetővé:

*Megy*(*x*, *y*) ⊆ *MegyHová*(*y*)

*Megy*(*x*, *y*) ⊆ *MegyHonnan*(*x*)

Hasonlóan, argumentumok hozzáadásával még specifikusabb kategóriákat hozhatunk létre. más ágensek cselekvéseinek lefrásához például hozzáadhatjuk az ágenseket jelölő argumentumot. így ahhoz, hogy azt mondassuk, hogy „tegnap Shankar New York-ból Újdelhibe utazott”, azt írhatnánk, hogy:

$\exists e \in \text{Repül}(\text{Shankar}, \text{New York}, \text{Újdelhi}) \wedge \text{RészEsemény}(e, \text{Tegnap})$

Az állítás e formája annyira gyakori, hogy külön rövidítést fogunk rá alkalmazni:  $E(c, i)$ , aminek az a jelentése, hogy a  $c$  eseménykategória egy eleme egy  $i$  intervallum, vagy az esemény részeseménye:

$$E(c, i) \Leftrightarrow \exists e \in c \wedge \text{RészEseménye}(e, i)$$

Igy:

$$E(\text{Repül}(\text{Shankar}, \text{NewYork}, \text{Újdelli}), \text{Tegnap})$$

## Folyamatok

Az eddig látott események meghatározott struktúrával rendelkező **diszkrét események** (*discrete events*) voltak. Shankar utazásának van kezdete, közepe és vége. Ha félbeszakítjuk, az esemény megváltozik – nem lesz többé egy New York és Újdelli közötti utazás, hanem helyette egy New York és egy Európában valahol fekvő pont közötti utazás. A *Repül(Shankar)* által jelölt eseménykategóriának ezzel szemben más a jellege. Ha Shankar repülésének egy rövid intervallumát vesszük, mondjuk a harmadik 20 perces időszakaszát (amikor is türelmetlenül várja már a második csomag mézben pörkölt földimogyorót), ez az esemény még mindig a *Repül(Shankar)* egy része. Valójában ez akármilyen részintervallum esetében is igaz.

Az ilyen tulajdonságú események kategóriáit **folyamat-** (*process*) kategóriáknak, illetve **folytonos esemény** (*liquid event*) kategóriáknak nevezzük. Egy folyamat akármilyen részintervalluma ugyanannak a folyamatkategóriának a tagja. A diszkrét eseményekre ki-alakított jelölést alkalmazva mondhatjuk például, hogy Shankar valamikor tegnap repült:

$$E(\text{Repül}(\text{Shankar}), \text{Tegnap})$$

Gyakran ki szeretnénk jelenteni, hogy valamelyen folyamat egy teljes intervallumon keresztül tartott, és nem csak annak valamelyen részintervallumában. Erre  $T$  predikátumot fogjuk használni:

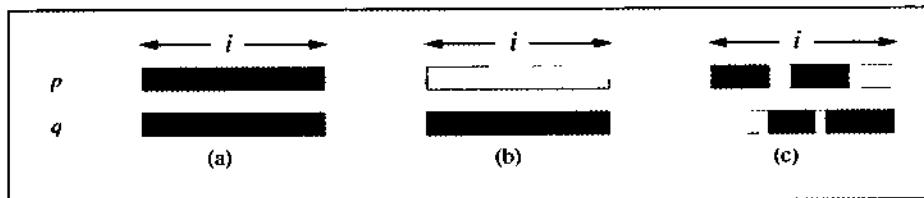
$$T(\text{Dolgozik}(\text{István}), \text{MaiEbéldő})$$

$T(c, i)$ -nek az a jelentése, hogy egy  $c$  típusú esemény az egész  $i$  intervallum ideje alatt tartott – azaz az esemény pontosan akkor kezdődik és fejeződik be, amikor az intervallum.

Folyékony és nem folyékony események közötti megkülönböztetés pontosan analóg a szubsztanciák, avagy az *anyag* és az *egyedi objektumok* közötti különbséggel. Tény, hogy egyes kutatók folyékony eseményeket **temporális szubsztanciáknak** (*temporal substances*) neveztek el, míg a vajszerű dolgok **terebeli szubsztanciák** (*spatial substances*).

A folytonos események a folyamatosan változó folyamatokon túlmenően képesek a folyamatosan nem változó folyamatok leírására is. Ezeket gyakran **állapotoknak** (*states*) nevezzük. A „Shankar New Yorkban tartózkodik” például egy olyan állapotkategória, amit *Benne(Shankar, NewYork)*-kal lehetne jelölni. Azt, hogy Sankar az egész mai napot New Yorkban töltötte, úgy írhatnánk, hogy:

$$T(\text{Benne}(\text{Shankar}, \text{NewYork}), \text{Ma})$$



**10.4. ábra.** Összetett események ábrázolása. (a)  $T(\text{Mindkettő}(p, q), i)$ , másnéven  $T(p \circ q, i)$ , (b)  $T(\text{Egyik}(p, q), i)$ , (c)  $T(\text{VagyVagy}(p, q), i)$ .

Primitív állapotok vagy események kombinálásával bonyolultabb képződményeket is ki-alakíthatunk. Ez a megközelítés a **folyó esemény kalkulus** (*fluuent calculus*). A folyó esemény kalkulus nem egyedi folyó eseményeket, hanem folyó események kombinációt reifikálja. Láttuk már annak a módját, hogy az egyszerre történő két dolog eseményét hogyan reprezentáljuk, azaz láttuk a  $\text{Mindkettő}(e_1, e_2)$  függvényt. A folyó kalkulusban ezt általában az  $e_1 \circ e_2$  infix jelöléssel rövidíti. Hogy megmondhassuk, hogy például „valaki sétált és közben rágógumit rágott”, azt írhatjuk, hogy:

$$\exists p, i \ (p \in \text{Ember}) \wedge T(\text{Sétál}(p) \circ \text{RágótRág}(p), i)$$

A „ $\circ$ ” függvény kommutatív és asszociatív, pontosan úgy, mint a logikai konjunkció. Definiálhatjuk a diszjunkció és a negálás analóját is, azzal viszont óvatosabban kell el-járunk, mivel a diszjunkciót kétféle módon lehet értelmesen interpretálni. Amikor azt mondjuk, hogy „az ágens az utóbbi két percben vagy sétált, vagy rágógumit rágott”, gondolhatunk arra, hogy az ágens az egész idő alatt az egyik cselekvést végezte, vagy pedig, hogy a két cselekvést felváltva tette. E két lehetőség megjelölésére az *Egyik* és a *VagyVagy* függvényeket fogjuk használni. A komplex eseményeket a 10.4. ábra szemlélteti.

## Intervallumok

Az idő fontos minden cselekvő ágens számára, és az időintervallumok reprezentálására számos erőfeszítés történt. Az időintervallumok két fajtájával foglalkozunk: az időpillanatokkal és a kiterjesztett intervallumokkal. A különbség köztük az, hogy csak az időpillanatnak lehet zérus időtartama:

$$\begin{aligned} & \text{Partíció}\{\text{IdőPillanatok}, \text{KiterjesztettIntervallumok}\}, \text{IdőIntervallumok} \\ & i \in \text{IdőIntervallumok} \Leftrightarrow (\text{Tartam}(i) = \text{Másodperc}(0)) \end{aligned}$$

Most egy időskálát konstruálunk, amelynek pontjait időpillanatokhoz rendeljük hozzá: ezzel abszolút időpontokat kapunk. Az időskála tetszőleges lehet. Az időt másodpercekben fogjuk mérni, és kimondjuk, hogy a 0. idő az 1900. január 1-jei (GMT) éjfél. A *Kezdet* és a *Vég* függvények az időintervallum legkorábbi és a legkésőbbi időpillanatát adják vissza, az *Idő* függvény egy időpillanathoz tartozó pontot keres ki az időskálán. Az *IdőTartam* függvény megadja a kezdeti idő és a végidő közötti különbséget.

$$\begin{aligned} & \text{IdőIntervallum}(i) \rightarrow \text{IdőTartam}(i) = (\text{Idő}(\text{Vége}(i)) - \text{Idő}(\text{Kezdete}(i))) \\ & \text{Idő}(\text{Kezdete}(AD1900)) = \text{Másodperc}(0) \end{aligned}$$

$Idő(Kezdete(AD2001)) = Másodperc(3187324800)$

$Idő(Vége(AD2001)) = Másodperc(3218860800)$

$IdőTartam(AD2001) = Másodperc(31536000)$

Hogy a számokat könnyebb legyen olvasni, vezessünk be egy határgumentumú *Dátum* függvényt (óra, percek, másodpercek, hónap, nap és év), melynek visszaadott értéke egy pont az időskálán:

$Idő(Kezdete(AD2001)) = Dátum(0, 0, 0, 1, Január, 2001)$

$Dátum(0, 20, 21, 24, 1, 1995) = Másodperc(3000000000)$

Két intervallum *Találkozik*, ha az egyiknek a végideje megegyezik a másiknak a kezdeti idejével. Már egyedül a *Találkozik* segítségével is lehetséges olyan további predikátumokat definiálni, mint az *Előtte*, az *Utána*, a *Közben* és az *Átlapolódik*. Azonban sokkal intuitívebb, ha a definíálásuk az időskála pontjainak segítségével történik (lásd 10.5. ábrán a grafikus reprezentációt).

$Találkozik(i, j) \Leftrightarrow Idő(Vége(i)) = Idő(Kezdete(j))$

$Előtte(i, j) \Leftrightarrow Idő(Vége(i)) < Idő(Kezdete(j))$

$Utána(j, i) \Leftrightarrow Előtte(i, j)$

$Közben(i, j) \Leftrightarrow Idő(Kezdete(j)) \leq Idő(Kezdete(i)) \wedge Idő(Vége(i)) \leq Idő(Vége(j))$

$Átlapolódik(i, j) \Leftrightarrow \exists k \ Közben(k, i) \wedge Közben(k, j)$

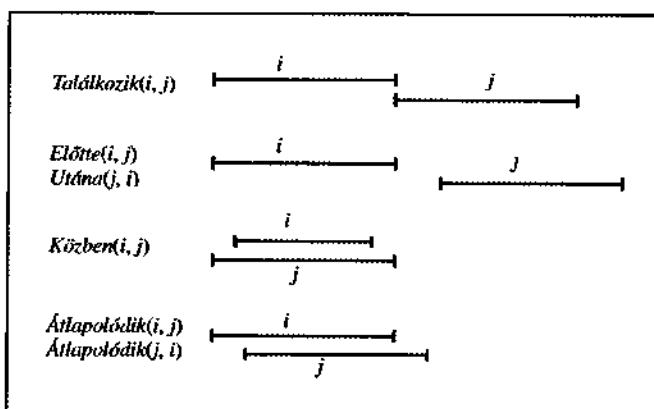
Annak megadására, hogy II. Erzsébet uralkodása VI. György uralkodását követte, Elvis uralkodása viszont az 1950-es évekkel átlapolódott, a következő írhatjuk:

$Utána(Uralkodása(II.Erzsébet), Uralkodása(VI.György))$

$Átlapolódik(ÖtvenesÉvek, Uralkodása(Elvis))$

$Kezdete(ÖtvenesÉvek) = Kezdete(AD1950)$

$Vége(ÖtvenesÉvek) = Vége(AD1950)$



10.5. ábra. Az időintervallumok predikátumai

## Folyó események és objektumok

Említettük, hogy fizikai objektumokat általánosított objektumoknak lehet tekinteni, abban az értelemben, hogy egy objektum a tér-idő egy darabkája. Az USA például egy olyan eseménynek képzelhető el, amely mondjuk 1776-ban, 13 állam uniójaként kezdődött és ma, 50 állam uniójaként, még folyamatban van. Az USA változó aspektusait állapot folyó eseményekkel írhatjuk le. Mondhatjuk például, hogy valamikor, az 1999. évben a lakossága 271 millió fő volt:

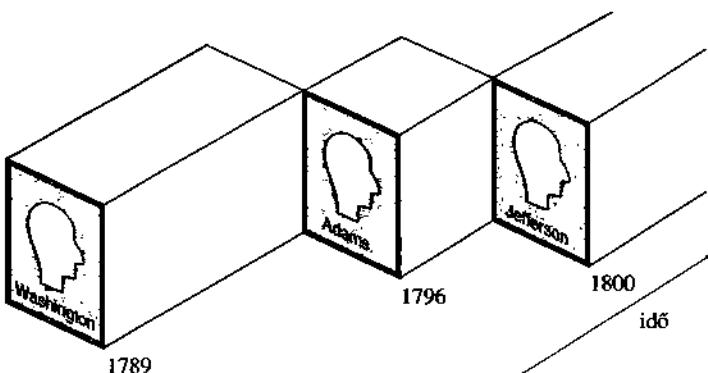
*E(Lakosság(USA), (271000000), AD1999)*

Egy másik ilyen aspektus, amely a szerencsétlenségektől eltekintve, négy- vagy nyolcévenként változik, az ország elnöke. Felvehető, hogy az *Elnök(USA)* egy olyan logikai term, amely különböző időkben különböző objektumokat jelent. Ez sajnos nem lehetőséges, mert egy term egy adott modellstruktúrában pontosan egy objektumot jelöl meg. (Az *Elnök(USA, t)* jelölhetne különböző objektumokat / értékének megfelelően, az ontológiánk azonban az időpontokat és a folyó eseményeket szétválasztja.) Egyetlen lehetőség, hogy az *Elnök(USA)* egy olyan egyedi objektumot jelöl, amely különböző időkben különböző emberekből áll. Az *Elnök(USA)*, mint objektum, George Washington volt 1789-től 1796-ig, John Adams volt 1796-tól 1800-ig stb. (lásd 10.6. ábra). Azt, hogy George Washington 1790-ben elnök volt, a következőképpen írhatjuk le:

*T(Elnök(USA) = GeorgeWashington, AD1790)*

Azonban óvatosnak kell lenni. Ebben a mondatban az „=” inkább egy függvény, és nem egy standard logikai operátor. Az interpretáció nem az, hogy *GeorgeWashington* és az *Elnök(USA)* 1790-ben logikailag azonosak. A logikai azonosság nem olyan dolog, ami időben változhat. A logikai azonosság az egyes objektumok azon részeseményei között áll fenn, amelyeket az 1790-es elnöki periódus definiált.

Ne tévessze össze a *GeorgeWashington* fizikai objektumot az atomok egy gyűjteményével. George Washington logikailag nem azonos semmilyen konkrét atomgyűjteménnyel, mert az az atomhalmaz, amiből áll, időben lényegesen változik. Neki rövid élettartama van, minden atomnak saját hosszú élettartama van. Bizonyos periódusban metszik egymást, amikor is az atom temporális szelete a *RészEseménye* George-nak, majd külön útjukat járják.



10.6. ábra. Az *Elnök(USA)* objektum sémaszerű ábrázolása létezésének első 15 évében

## 10.4. MENTÁLIS ESEMÉNYEK ÉS MENTÁLIS OBJEKTUMOK

Az eddig megkonstruált ágenseknek hiedelmeik vannak, továbbá az ágensek új hiedelmeik kikövetkeztetésére is képesek. Ennek ellenére *hiedelmekre* és *következtetésre vonatkozó* tudással egy ágens sem rendelkezik. Az egyágenses problématerületeken egy ágensnek a saját tudásáról és következtető eljárásáról meglévő tudása hasznos a következtetés irányításában. Ha például valaki tudatában van annak, hogy Románia földrajzárol semmit sem tud, nem kell óriási erőfeszítést tennie, hogy az Arad és Bukarest közötti legrövidebb utat kiszámítsa. Következtethetünk a saját tudás állapotunkról annak érdekében, hogy terveket készítsük annak megváltoztatására – például azáltal, hogy Románia térképét megvásároljuk. Többágenses problématerületeken fontos, hogy egy ágens más ágensek mentális folyamataira tudjon következtetni. Egy román rendőr nyilván jól tudja, hogy melyik a legjobb út Bukarest felé, így az ágens kérhet tőle segítséget.

Végeredményben tehát olyan mentális objektumok és az azokat manipuláló mentális folyamatok modelljére van szükségünk, amelyek másvalaki fejében (vagy másvalami tudásbázisában) találhatók meg. A modellnek valóságban kell lennie, azonban nem kell a részletekre kitérnie. Nem kell, hogy képesek legyünk megjósolni, hogy egy konkrét ágens a következtetéseit hány milliszekundum alatt állítja elő, vagy hogy egy konkrét vizuális ingerrel szembesítve, mely neuronok tüzelnek az állat agyában. Boldogok leszünk, ha kikövetkezhetjük, hogy román rendőr eligazít minket Bukarest irányába, ha tudja az utat, és azt hiszi, eltévedtünk.

### A hiedelmek formális elmélete

Az ágensek és a „mentális objektumok” közötti olyan kapcsolatokkal kezdünk, mint a *Hiszi*, a *Tudja* és az *Akarja*. Az ilyen relációkat **ítéletlogikai attitűdöknek (propositional attitudes)** nevezzük, mert egy ágens attitűdjét írják le egy ítéletlogikai állítással szemben. Tegyük fel, hogy Lujza hisz valamit, azaz *Hiszi(Lujza, x)*. Vajon milyenfajta dolog az *x*? Először is világos, hogy *x* logikai állítás nem lehet. Ha *Repül(Superman)* egy logikai állítás, akkor nem mondhatjuk azt, hogy *Hiszi(Lujza, Repül(Superman))*, mert predikátumok argumentumaként csak termeket (és nem állításokat) fogadunk el. Ha azonban a *Repül* egy függvény, akkor a *Repül(Superman)* már jó jelölt egy mentális objektumra, és a *Hiszi* lehet az a reláció, amely egy ágenst összekapcsol ezzel az ítélet jellegű folyó eseménnyel. Egy állítás átalakítása objektummá nem más, mint **reifikálás**<sup>7</sup> (*reification*).

Úgy tűnik meg is kaptuk, amit akartunk: egy ágensnek azt a képességet, hogy következtetni tudjon más ágensek hiedelmeiről. Sajnos e megközelítésnek vannak nehézségei. Ha Clark és Superman egy és ugyanaz a személy (azaz *Clark = Superman*), akkor Clark repül és Superman repül egy és ugyanaz az eseménykategória, azaz *Repül(Clark) = Repül(Superman)*.

<sup>7</sup> A „reifikálás” szó a latin *res*, avagy dolog szóból származik. John McCarthy a „dolgozítás” (*thingification*) elnevezést javasolta, de ez nem terjedt el.

*pil(Superman)*. Azaz el kell fogadnunk, hogy ha Lujza azt hiszi, hogy Superman tud repülni, el kell hinnie, hogy Clark tud repülni, akkor is, ha nem hiszi, hogy Clark Superman. Azaz:

$$(Superman = Clark) \models (Hiszi(Lujza, Repül(Superman)) \leftrightarrow Hiszi(Lujza, Repül(Clark)))$$

Bizonyos értelemben ez igaz is. Egy bizonyos személyről, akit néha Clarknak hívnak, Lujza azt hiszi, hogy tud repülni. Más értelemben azonban ez nincs rendjén. Ha megkérdezzük Lujzát, hogy „Clark tud-e repülni?”, minden bizonnal nemmel válaszolna. A reifikált objektumok és események jól működnek a *Hiszi* első értelmezése esetén, a másodikhoz viszont az szükséges, hogy reifikáljuk ezen objektumok és események leírását, így *Clark* és *Superman* különböző leírások legyenek, annak ellenére, hogy ugyanarra az objektumra vonatkoznak.

Azt a tulajdonságot, hogy egy termet egy vele ekvivalens termmel szabadon helyettesíthetünk, formálisan **referenciális átláthatóság**nak (*referential transparency*) nevezzük. Elsőrendű logikában minden reláció referenciálisan átlátható. Számunkra az lenne a jó, ha a *Hiszi* (és más ítéletlogikai attitűdök is) olyan relációkként lennének definíálhatók, melynek második argumentuma **elmosódott (opaque)** – azaz, ahol ekvivalens termeket egymással nem lehet helyettesíteni az értelmezés megváltoztatása nélkül.

Ennek két módja van. Az első a logika egy új változatának – az ún. **modális logikának (modal logic)** a használata, amelyben az olyan ítéletlogikai attitűdök, mint a *Hiszi* és a *Tudja* referenciálisan elmosódott **modális operátorok (modal operators)** lesznek. Ezzel a megközelítéssel az irodalmi és történeti megjegyzések keretében foglalkozunk. A második, a továbbiakban kifejtett megközelítés a hatékony elmosódottság elérése a különben referenciálisan átlátható nyelvben a mentális objektumok ún. **szintaktikai elmélete (syntactic theory)** révén. Ez azt jelenti, hogy mentális objektumokat **füzérekkel (strings)** reprezentálunk. Az eredmény az ágens tudásbázisának egy olyan durva modellje, ahol a tudásbázis az ágens által elhitt állításokat reprezentáló füzérből áll. Egy füzér nem más, mint egy szimbólumlistát megjelölő komplex term, így a *Repül(Clark)* eseményt a  $[R, e, p, ü, l, (, C, l, a, r, k, )]$  karakterfüzérrel reprezentáljuk, amit „*Repül(Clark)*”-ként fogunk rövidíteni. A szintaktikai elmélet eleme az **egyedi füzér axióma (unique string axiom)**, amely azt mondja ki, hogy két füzér akkor és csak akkor azonos, ha azonos karakterekből áll. Ebben a megfogalmazásban a *Clark = Superman* ellenére „*Clark*”  $\neq$  „*Superman*”.

Most meg kell adnunk a füzér reprezentációs nyelv részére a szintaxist, a szemantikát és a bizonyítási elméletet, pontosan úgy, ahogyan ezt a 7. fejezetben tettük. A különbség az, hogy ezeket most minden elsőrendű logikában kell definiálnunk. A *Jelent* függvényel kezdünk, amely a füzért arra az objektumra képezi le, amit jelöl. A *Neve* egy olyan függvényel definíálható, amely egy objektumot az azt megnevező füzérre képezi le. A „*Clark*” és a „*Superman*” jelölés jelentése egy objektum, amire a *VasEmber* konstansszimbólummal fogunk hivatkozni, és aminek a neve a tudásbázisban lehet „*Clark*”, „*Superman*” vagy valamilyen más konstans, mondjuk „*X<sub>11</sub>*”.

$$Jelent(„Clark”) = VasEmber \wedge Jelent(„Superman”) = VasEmber$$

$$Neve(VasEmber) = „X_{11}”$$

A következő lépés a következetési szabályok definíálása egy logikai ágens számára. Kívánhatnánk például, hogy egy logikai ágens képes legyen Modus Ponent elvégezni:

ha elhiszi a  $p$ -t, és elhiszi, hogy  $p \Rightarrow q$ , akkor elhiszi a  $q$ -t is. Az axióma első verziója valahogy így nézhetne ki:

$$\text{LogikaiÁgens}(a) \wedge \text{Hiszi}(a, p) \wedge \text{Hiszi}(a, „p \Rightarrow q”)} \Rightarrow \text{Hiszi}(a, q)$$

Ez azonban helytelen, mert a „ $p \Rightarrow q$ ” füzér tartalmazza ugyan a „ $p$ ” és „ $q$ ” betűket, de semmi köze az olyan füzérekhez, amelyekben a  $p$  és  $q$  változók értékei szerepelnek. A helyes megfogalmazás az alábbi:

$$\text{LogikaiÁgens}(a) \wedge \text{Hiszi}(a, p) \wedge \text{Hiszi}(a, \text{Concat}(p, „\Rightarrow”, q))} \Rightarrow \text{Hiszi}(a, q)$$

ahol a *Concat* egy füzérfüggvény, amely az elemeit konkátenálja. A *Concat*( $p$ , „ $\Rightarrow$ ”,  $q$ )-t le fogjuk rövidíteni „ $p \Rightarrow q$ ”-ra. Azaz  $x$  előfordulása egy füzéren belül azt jelenti, hogy az  $x$  változó értékét be kell helyettesíteni. A Lisp-programozók ebben felismerik a back-quote operátort, a Pearl-programozók pedig a \$-változó interpolációt.

Ha Modus Ponensen kívül más következtetési szabályokat is hozzáveszünk, olyan kérdések megválaszolására leszünk képesek, mint a „Feltéve, hogy a logikai ágens ezeket a premisszákát ismeri, képes-e azt a konklúziót kikövetkeztetni?”. A közönséges következtetési szabályokon túlmenően szükségünk van hiedelem-specifikus szabályokra is. Így például az alábbi szabály azt mondja ki, hogy ha egy logikai ágens elhisz valamit, akkor azt is elhiszi, hogy elhiszi azt.

$$\text{LogikaiÁgens}(a) \wedge \text{Hiszi}(a, p) \Rightarrow \text{Hiszi}(a, „\text{Hiszi}(\underline{\text{Neve}}(a), p)“)$$

Az ilyen ágens, axiomáinkból adódóan, minden érvényes konklúziót képes azonnal következtetni. Ezt a jelenséget **logikai mindentudásnak** (*logical omniscience*) hívják. Számos kísérlet történt arra, hogy korlátos racionalitású ágenseket definiálunk, amelyek véges idő alatt korlátozott számú következetesre képesek. Ezek egyike sem teljesen kielégítő, azonban e megfogalmazások lehetővé teszik, hogy korlátos ágensekről igen korlátozott terjedelmű jóslásokat végezzünk.

## Tudás és hiedelem

A filozófusok évszázadok óta tanulmányozták a hinni és a tudni közötti kapcsolatot. Általánosan elfogadott, hogy tudás a bizonyított, igaz hiedelem. Azaz, ha „megtámadhatatlanul jó oknál fogva” hiszünk valamiben, és ez a valami igaz is, akkor ezt a valamit tudjuk is. A „megtámadhatatlanul jó oknál fogva” lényeges, hogy ne mondassunk olyat, hogy: „Tudom, hogy ez a feldobott érme fejjel felfelé fog leesni”, és ne legyen igazunk az esetek felében.

Legyen a *Tudja*( $a, p$ ) jelentése az, hogy egy  $a$  ágens *tudja*, hogy a  $p$  állítás igaz. A tudás más fajtáját is lehet definiálni. Nézzük például a „*tudja, hogy vagy..., vagy*” definícióját:

$$\text{TudjaVagyVagy}(a, p) \Leftrightarrow \text{Tudja}(a, p) \vee \text{Tudja}(a, „\neg p“)$$

Példánkat folytatva, Lujza tudja, hogy Clark vagy tud repülni, vagy nem, ha tudja, hogy Clark tud repülni, vagy ha tudja, hogy Clark nem tud repülni.

A „*tudja, mi*” koncepció valamivel bonyolultabb. Kísértést érezhetünk arra, hogy azt állítsuk, egy ágens tudja, mi Béla telefonszáma, ha létezik egy  $x$ , amiről az ágens azt tudja, hogy  $x = \text{TelefonSzáma(Béla)}$ . De ez így nem jó, mert például az ágens tudhatja, hogy Aliz és Béla telefonszáma ugyanaz (azaz  $\text{TelefonSzáma(Aliz)} = \text{TelefonSzáma}$

(Béla)), de ez sokat nem segít, ha Aliz telefonszámát sem tudja. A „tudja, mi” jobb definíciója azt mondja ki, hogy az ágensnek valami olyan  $x$ -et kell ismernie, amely egy számjegykből álló fűzér és amely Béla telefonszáma:

$TudjaMi(a, „TelefonSzáma(b)”) \Leftrightarrow$

$\exists x \ Tudja(a, ..x = TelefonSzáma(b)) \wedge x \in SzámjegyFüzér$

Természetesen más kérdések esetén az elfogadható válasznak mások lesznek a kritériumai. Arra a kérdésre, hogy „Mi New York állam fővárosa?”, a jó válasz „Albany”, és nem az, hogy „az a város, ahol a városháza áll”. Hogy ez lehetséges legyen, a *TudjaMi*-t háromargumentumos relációt alakítjuk át: lesz benne egy ágens, egy term és egy predikátum, amelynek igaznak kell lennie a válaszra alkalmazva. Például:

$TudjaMi(\text{Ágens}, „Főváros(NewYork”), SajátNév)$

$TudjaMi(\text{Ágens}, „TelefonSzáma(Béla”), SzámjegyFüzér)$

## Tudás, idő és cselekvés

A valódi esetek többségében az ágensnek az időben változó – saját vagy más ágensek – hiedelmeivel kell foglalkoznia. Az ágensnek terveket kell szónie, amelyek a saját hiedelmeinek változásaival járnak együtt – mint amilyen például egy térkép vásárlása, hogy Bukarestbe találjon. Más predikátumokhoz hasonlóan lehetséges a *Hiszi* reifikálása, és egy időperiódus alatt történő hiedelmek megtárgyalása. Így mondhatjuk például, hogy Lujza ma hiszi, hogy Superman tud repülni:

$T(\text{Hiszi}(Lujza, „Repül(Superman)”), Ma)$

Ha a hiedelem objektuma egy olyan állítás, amely időben változhat, ezt a *T* operátorral a füzéren belül írhatjuk le. Ekkor mondhatnánk azt, hogy Lujza ma hiszi, hogy Superman tegnap tudott repülni:

$T(\text{Hiszi}(Lujza, „T(Repül(Superman), Tegnap)”), Ma)$

Ha már az időben evolváló hiedelmeket le tudjuk írni, felhasználhatjuk az eseménykalkulus mechanizmusát, hogy hiedelmeket tartalmazó terveket készítsünk. A cselekvéseknek lehetnek **tudás-előfeltételei** (*knowledge preconditions*) és **tudáshatásai** (*knowledge effects*). Annak a cselekvésnek például, hogy valakinek a telefonszámát tárcsázuk, az az előfeltétele, hogy tudjuk a számot, és annak a cselekvésnek, hogy a telefonszámot a könyvből kikeressük, az a hatása, hogy a számot tudni fogjuk. Ezt az utolsó cselekvést az eseménykalkulus mechanizmusával az alábbi módon írhatjuk le:

$Iniciálz(Kikeres(a, „TelefonSzáma(b)”),$

$TudjaMi(a, „TelefonSzáma(b)”, SzámjegyFüzér), t)$

Az információgyűjtő és az azt felhasználó terveket sokszor röviden **futási idejű változóknak** (*runtime variables*) nevezzük, ami a korábban leírt értékbe helyettesítő változó konvenciójához szorosan kapcsolódik. Béla telefonszámának a kikeresésére, majd a tárcsázására vonatkozó tervet az alábbi alakban fel lehet írni:

$[Kikeres(\text{Ágens}, „TelefonSzáma(Béla)”, n), Tárcsáz(n)]$

Itt n egy futási idejű változó, aminek értékét a *Kikeres* cselekvés fogja rögzíteni, és amit majd a *Tárcsáz* cselekvés felhasználhat. Az ilyen tervezek sűrűn előfordulnak a részben megfigyelhető tárgyterületeken. Látni fogunk erre példákat a következő alfejezetben és a 12. fejezetben.

## 10.5. AZ INTERNETES BEVÁSÁRLÁS VILÁGA

Ebben az alfejezetben az internetes bevásárlással kapcsolatos tudás egy részét fogjuk kódolni. Egy olyan vásárló-kutató ágenst fogunk tervezni, amely segíti a vásárlónak az interneten megtalálni a termékek ajánlatait. A vásárló ágens a vásárlótól a termék leírását kapja és feladata a termék eladását hirdető weblapok listáját előállítani. Egyes esetekben a vásárló leírása pontos lesz, mint amilyen például a *Coolpix 995 digitális fényképezőgép* – a feladat ekkor a legjobb ajánlatot tevő boltot megtalálni. Más esetekben a leírás csak részben lesz specifikált, mint például a *300 dollárnál olcsóbb digitális fényképezőgép*, és az ágens kénytelen lesz különböző termékeket összehasonlítani.

A vásárló ágens környezete az egész világháló – ami semmiképpen sem egy játék-környezet. Ugyanaz a komplex, folyamatosan evolváló környezet, amit az emberek milliói minden nap használnak. Az ágens érzetei weblapok, de amíg egy emberi web-felhasználó a lapokat a képernyőre kivetített pixelek formájában látja, a vásárló ágens a lapokat karakterfüzérek alakjában fogja érzékelni, ahol közönséges szavak keverednek a HTML jelölőnyelv formattáló utasításaival. A 10.7. ábra egy weblapot mutat és a hozzá tartozó HTML-karakterfüzért. A vásárló ágens érzékelési problémájának lényege a hasznos információk az ilyen fajtájú érzelből való kiemelése.

Világos, hogy weblapot érzékelni egyszerűbb, mint mondjuk, Kairóban taxi vezetése közben érzékeléseket szerezni. Az internetes érzékelés mégsem mentes a bonyodalmaktól. A 10.7. ábrán látható weblap igen egyszerű a valódi vásárlási lapokhoz képest, ahol találkozni lehet olyan elemekkel, mint sütit, Java, Javascript, Flash, robot kizároló protokollok, elégpelt HTML, hangfájlok, filmfelvételek és egy JPEG kép részeként megjelenő szövegek. Egy ágens, amely képes az internet *egészével* elbánni, majdnem olyan bonyolult, mint egy robot, amely a valódi világban képes mozogni. Mi egy egyszerű ágensre összpontosítunk, amely e bonyodalmak zömét nem fogja figyelembe venni.

Az ágens első feladata, hogy releváns termékajánlatokat találjon (majd később megláttuk, hogy a releváns ajánlatok közül hogyan kell kiválasztani a legjobbat). Legyen a *lekérdezés* a terméknak a felhasználó által begépelt leírása (például „laptop”), akkor egy weblap a *lekérdezés* szempontjából releváns ajánlat, ha a lap releváns és tényleg egy ajánlat. A lappal kapcsolatos URL-t is nyomon fogjuk követni:

$$\text{RelevánsAjánlat}(lap, url, \text{lekérdezés}) \Leftrightarrow \text{Releváns}(lap, url, \text{lekérdezés}) \wedge \text{Ajánlat}(lap)$$

A legmodernebb laptopot bemutató lap releváns lenne, ha azonban a vásárlásra nem ad lehetőséget, nem ajánlat. Egyelőre azt fogjuk mondani, hogy egy lap ajánlat, ha a lapon egy HTML-hivatkozásban, vagy űrlapon belül a „vásárol” vagy az „ára” szavakat tartalmazza. Más szóval, ha a lap tartalmaz egy „<a ... vásárol ... </a>” alakú füzért, akkor ez egy ajánlat. A „vásárol” helyett lehetne az „ára”, illetve az „a” helyett a „form”. Felírhatjuk ennek axiomatikus alakját:

## Egy absztrakt online bolt

Válasszon a termékeink listájából:

- Számítógépek
- Fényképezőgépek
- Könyvek
- Videók
- Zene

```
<h1>Egy absztrakt online bolt</h1>
<i>Válasszon</i> a termékeink listájából:
<ul>
<li> <a href="http://absz-bolt.com/szmg">Számítógépek</a>
<li> <a href="http://absz-bolt.com/feny">Fényképezőgépek</a>
<li> <a href="http://absz-bolt.com/konyv">Könyvek</a>
<li> <a href="http://absz-bolt.com/video">Videók</a>
<li> <a href="http://absz-bolt.com/zene">Zene</a>
</ul>
```

**10.7 ábra.** Egy absztrakt online bolt weblapja, ahogy egy böngésző használó ember látja (felül), és a hozzá tartozó HTML-füzér, ahogy azt a böngésző vagy a vásárló ágens látja (alul). A HTML-ben a < és a > közötti karakterek jelölő direktívák, amelyek meghatározzák, hogy a lapot hogyan kell kijelezní. Az <i>Válasszon</i> füzér például jelzi, hogy dölt betűre kell átkapcsolni, a Válasszon szót kijelezni és a dölt betű használatát befejezni. Egy lapazonosító, mint amilyen a http://absz-bolt.com/zene az ún. egységes erőforrás azonosító (uniform resource locator, URL). Az <a href="url">hivatkozás</a> bejelölés jelzi, hogy az url felé létesíteni kell egy hipertextkapcsolatot a hivatkozó szöveg (anchor text) hivatkozással.

$$\begin{array}{ll}
 \text{Ajánlat}(lap) & \Leftrightarrow (\text{Címkében}(„a”, füzér, lap) \vee \\
 & \quad \text{Címkében}(„form”, füzér, lap)) \\
 & \quad \wedge (\text{Benne}(„vásárol”, füzér)) \vee \text{Benne}(„ára”, füzér)) \\
 \text{Címkében}(„címke”, füzér, lap) & \Leftrightarrow \text{Benne}(“<” + \text{címke} + \text{füzér} + “</” + \text{címke}, lap) \\
 \text{Benne}(\text{alfüzér}, füzér) & \Leftrightarrow \exists i \text{ füzér}[i : i + \text{Hossza}(\text{alfüzér})] = \text{alfüzér}
 \end{array}$$

Most szükséges releváns lapokat találni. A stratégia az online bolt honlapjától kezdeni, és azokkal a lapokkal foglalkozni, amelyet a releváns linkek mentén el lehet érni.<sup>8</sup> Ágenünk számos boltról fog tudni, például:

Amazon ∈ OnlineBolt ∧ HonLap(Amazon, „amazon.com”)  
 Ebay ∈ OnlineBolt ∧ HonLap(Ebay, „ebay.com”)  
 ÁltBolt ∈ OnlineBolt ∧ HonLap(ÁltBolt, „ált-bolt.com”)

E boltok termékeit termékkategóriákba sorolják, és a fő kategóriákhoz linkekkel biztosítanak a honlaptól kiindulva. Kisebb kategóriákat a releváns linkek láncának követésével lehet elérni, és végül az ajánlatokhoz is eljutunk. Más szóval, egy lap a lekérdezés szempontjából releváns, ha a bolt honlapjától a releváns kategóriakapcsolat láncán át elérhető, és egy további link követésével a termékajánlathoz is eljutunk:

<sup>8</sup> A hivatkozáskövető stratégia alternatívája egy internetkereső gépnek a használata. Az internetkeresésről, az információk nyerés mögött húzódó technológiáról a 23.2. alfejezetben lesz szó.

*Releváns(lap, url, lekérdezés)  $\Leftrightarrow$*

$$\begin{aligned} & \exists bolt, hon \in OnLineBolt \wedge Honlap(bolt, hon) \\ & \wedge \exists url_2 \text{ RelevánsLánc}(hon, url_2, \text{lekérdezés}) \wedge \text{Link}(url_2, url) \\ & \wedge \text{lap} = \text{VeddElőLapot}(url) \end{aligned}$$

A *Link(tól, ig)* predikátum jelentése, hogy a *tól* URL-től az *ig* URL-ig létezik egy hiperhivatkozás (lásd 10.13. feladat). Hogy a *RelevánsLánc*-ot definiálhassuk, nem valamilyen régi hiperhivatkozást kell követnünk, hanem azokat, amelyeknél a csatolt hivatkozó szöveg jelzi, hogy a hivatkozás a lekérdezés szempontjából releváns. Erre a *LinkSzöveg(tól, ig, szöveg)*-et fogjuk használni, melynek jelentése, hogy létezik egy hivatkozás *tól*-tól *ig*-ig, szöveg-gel mint hivatkozó szöveggel. A *start* és a *vége* URL-ek közötti hivatkozáslánc a *d* leírás szempontjából releváns, ha minden hivatkozás hivatkozó szövege a *d* szempontjából releváns kategóriánév. A lánc létezését egy rekurzív definíció biztosítja, az üres láncnal (*start = vége*) mint kiinduló állappal:

*RelevánsLánc(start, vége, lekérdezés)  $\Leftrightarrow$  (start = vége)*

$$\begin{aligned} & \vee (\exists u, szöveg \text{ LinkSzöveg}(start, u, szöveg)) \\ & \wedge \text{RelevánsKategóriaNév(lekérdezés, szöveg)} \\ & \wedge \text{RelevánsLánc}(u, vége, lekérdezés)) \end{aligned}$$

Most definiálnunk kell, hogy egy *lekérdezés* szempontjából egy *szöveg* mikor lesz egy *RelevánsKategóriaNév*. Először tudunk kell a füzereket és az általuk megnevezett kategóriákat kapcsolatba hozni. Jó lesz erre a *Név(s, c)*, amely azt állítja, hogy az *s* füzer a *c* kategória neve – mondhatjuk például hogy *Név(laptop”, LaptopSzámítógép)*. A *Név* predikátumra további példák a 10.8. ábrán találhatók. A következő a relevancia definíálása. Tegyük fel, hogy a lekérdezés a „laptop”. *RelevánsKategóriaNév(lekérdezés, szöveg)* igaz lesz, feltéve, hogy az alábbiak közül egy teljesül:

- A *szöveg* és a *lekérdezés* ugyanazt a kategóriát nevezi meg – például „laptop számítógép” és „laptop”.
- A *szöveg* egy szuperkategóriát nevez meg, mint például „számítógép”.
- A *szöveg* egy alkategóriát nevez meg, mint például „ultrakönnyű laptop”.

A *RelevánsKategóriaNév* logikai definíciója az alábbi:

*RelevánsKategóriaNév(lekérdezés, szöveg)  $\Leftrightarrow$*

$$\exists c_1, c_2 \ Név(lekérdezés, c_1) \wedge Név(szöveg, c_2) \wedge (c_1 \subseteq c_2) \vee (c_2 \subseteq c_1) \quad (10.1)$$

Máskülönben a hivatkozó szöveg irreleváns, mert egy, ezen a vonulaton kívüli kategóriát nevez meg, mint például a „mainframe számítógép” vagy „pázsit és kert”.

Hogy képesek legyünk releváns kapcsolatokat követni, lényeges, hogy rendelkeznünk kell a termékek gazdag kategóriahierarchiával. E hierarchia legfelső része hasonlíthat például a 10.8. ábrán látható hierarchiára. Nem lenne jó ötlet az összes lehetséges bevásárlási kategóriát listázni, mert a bevásárlónak minden lehetnek új kíváncsai, és a termékgyártók minden új termékekkel fognak előrukkolni, hogy a vásárlókat kielégítsék (elektromos térdmelegítő?). Egy, mondjuk ezer kategóriából álló ontológia azonban a vásárlók többsége számára hasznos eszköznek fog bizonyulni.

A termékhierarchián túlmenően rendelkeznünk kell a kategórianevek gazdag szótárával. Az élet sokkal egyszerűbb lenne, ha a kategóriák és az azokat megnevező füzerek

<i>Könyv</i> ⊂ <i>Termék</i>	<i>Név</i> („ <i>könyv</i> ”, <i>Könyv</i> )
<i>ZeneiFelvétel</i> ⊂ <i>Termék</i>	<i>Név</i> („ <i>zene</i> ”, <i>ZeneiFelvétel</i> )
<i>ZeneiCD</i> ⊂ <i>ZeneiFelvétel</i>	<i>Név</i> („ <i>CD</i> ”, <i>ZeneiCD</i> )
<i>ZeneiSzalag</i> ⊂ <i>ZeneiFelvétel</i>	<i>Név</i> („ <i>szalag</i> ”, <i>ZeneiSzalag</i> )
<i>Elektronika</i> ⊂ <i>Termék</i>	<i>Név</i> („ <i>elektronika</i> ”, <i>Elektronika</i> )
<i>DigitálisFényképezőgép</i> ⊂ <i>Elektronika</i>	<i>Név</i> („ <i>digikamera</i> ”, <i>DigitálisFényképezőgép</i> )
<i>SztereóBerendezés</i> ⊂ <i>Elektronika</i>	<i>Név</i> („ <i>sztereó</i> ”, <i>SztereóBerendezés</i> )
<i>Számítógép</i> ⊂ <i>Elektronika</i>	<i>Név</i> („ <i>szo&amp;szacute;mítogep</i> ”, <i>Számítógép</i> )
<i>LaptopSzámítógép</i> ⊂ <i>Számítógép</i>	<i>Név</i> („ <i>laptop</i> ”, <i>LaptopSzámítógép</i> )
<i>AsztaliSzámítógép</i> ⊂ <i>Számítógép</i>	<i>Név</i> („ <i>PC</i> ”, <i>AsztaliSzámítógép</i> )
...	...
(a)	(b)

10.8. ábra. (a) A termékkategóriák taxonómiaja. (b) Ezen kategóriák számára a hivatkozó szövegek.

között egy-egyértelmű kapcsolat állna fenn. A **szinonima (synonymy)** problémáját – két név, például „laptop számítógép” és „laptop” ugyanazon kategória számára – már láttuk. Van még továbbá az **egyértelműsítés (ambiguity)** problémája – egy név két vagy több különböző kategória számára. Ha például a 10.8. (b) ábrán látható tudásbázishoz a:

*Név*(„*CD*”, *DiplomataJármű*)

mondatot is hozzáadjuk, akkor a „*CD*” két különböző kategóriát fog megnevezni.

A szinonimák és az egyértelműsítés lényegesen megnövelheti az ágens által követendő utak számát, és néha megnehezítheti annak eldöntését, hogy egy adott lap tényleg releváns-e. Sokkal komolyabb probléma, hogy a felhasználó által begépelhető leírások vagy a bolt által használt kategórianevek választéka igen széles lehet. A link például a „laptop”-ról szólhat, holott a tudásbázis csak „laptopok”-ról tud, vagy pedig a felhasználó „egy számítógépet” keres, „amely elférne a Boeing 737 turistaosztályán az étkező-asztalkán”. Lehetetlen előre végig felsorolni egy kategória megnevezéseinek minden változatát, így az ágensnek bizonyos esetekben tovább kell következtetnie, hogy kiderítse, érvényes-e a *Név* reláció. A legrosszabb esetben ez a természetes nyelv teljes megértését igényelné, mely téma tárgyalását a 22. fejezetig készletetjük. A gyakorlatban néhány egyszerű szabály, mint például engedélyezni, hogy a „laptop” illeszkedjen a „laptopok” kategóriára, igen jól szuperál. A 10.5. feladatban megkérjük majd az olvasót, hogy egy ilyen szabálykészletet fejlesszen ki, azt követően, hogy az online boltokban egy kicsit körbenézett.

Az előbbi bekezdésben megadott logikai definíció, valamint a termékkategóriákat és a megnevezési konvenciókat tartalmazó megfelelő tudásbázisok birtokában készek vagyunk-e már, hogy a lekérdezésünkre vonatkozó releváns ajánlatok halmazát kikövetkeztető algoritmust alkalmazzuk? Még nem! A hiányzó elem a *VeddElőLapot(url)* függvény, amely egy adott *url* címen lévő HTML lapra hivatkozik. Az ágens nem rendelkezik a tudásbázisában minden URL lap tartalmával, és az ilyen tartalom kitalálására vonatkozó explicit szabályokkal sem rendelkezik. Ehelyett elintézhetjük, hogy amikor egy részcél a *VeddElőLapot* függvényre hivatkozik, a megfelelő HTTP eljárás fog meg-

hívódni. A következtető gép számára ily módon mintha az egész világháló jelenne meg a tudásbázisában. Ez az ún. **procedurális kibővítésként (procedural attachment)** ismert általános módszer egy példája, ahol az egyes predikátumokat és függvényeket speciális rendeltetésű eljárásokkal kezeljük.

## Ajánlatok összehasonlítása

Tegyük fel, hogy az előbbi részfejezet következtetési eljárása a „laptop” lekérdezésünkre több ajánlatlapot is adott vissza. Az ajánlatok összehasonlításához az ágensnek a lapokból releváns információt – ár, sebesség, diszkkapacitás, súly stb. – ki kell nyerni. Az előbb említett okoknál fogva, valódi weblapok esetén, ez igen nehéz feladatnak bizonyulhat. E probléma kezelésének megszokott módja a lapról információt kinyerő ún. **csomagolóprogramok (wrappers)** használata. Az információkinyerés technológiájával a 23.3. alfejezetben foglalkozunk. Egyelőre feltételezzük, hogy csomagolóprogram létezik, és egy lap és a tudásbázis birtokában a tudásbázishoz tényeket ad hozzá. A lapra tipikusan csomagolóprogramok egész hierarchiáját lehetne alkalmazni. Egy általánosat a dátumok és az árak kinyeréséhez, egy specifikusabbat, hogy a számítógépes termékek attribútumait találja meg, végül, ha szükséges, egy weboldal-specifikust, amely az adott bolt weboldalformátumát is tudja. Ha az ált-bolt.com egy weboldalt tartalmaz:

YVM ThinkBook 970. Ára: 300.000,- Ft

szöveggel, amit mindenféle technikai specifikáció követ, a csomagolótól elvárnánk, hogy az alábbi információt nyerje ki:

$$\exists lc, \text{ajánlat} \text{ } lc \in \text{LaptopSzámítógép} \wedge \text{ajánlat} \in \text{TermékAjánlat} \wedge \\ \text{KépernyőNagyság}(lc, Cm(32)) \wedge \text{KépernyőTipus}(lc, \text{SzínLCD}) \wedge \\ \text{MemóriaNagyság}(lc, Mbyte(512)) \wedge \text{CPUSebesség}(lc, GHz(2,4)) \wedge \\ \text{AjánlottTermék}(\text{ajánlat}, lc) \wedge \text{Bolt}(\text{ajánlat}, \text{AbsztBolt}) \wedge \\ \text{URL}(\text{ajánlat}, \text{„abszt-bolt.com/szg/34356.html”}) \wedge \\ \text{Ára}(\text{ajánlat}, \text{Fr}(300000)) \wedge \text{Dátum}(\text{ajánlat}, Ma)$$

Ez a példa néhány olyan problémát hivatott megmutatni, amelyek akkor jelentkeznek, ha a kereskedelmi tranzakcióhoz szükséges tudásszervezést komolyan vesszük. Vegyük észre például, hogy az ár az *ajánlatnak*, és nem magának a terméknek az attribútuma. Ez fontos, mert egy adott bolt ajánlata napról napra változhat, akár ugyanarra az egyedi laptopra nézve. Egyes kategóriák esetén – mint például a házak és a festmények esetén – ugyanazt az egyedi objektumot különböző közvetítők egy időben különböző áron is ajánlhatják. Több bonyodalom is elképzelhető, amelyek kezelésével nem foglalkoztunk. Ilyen például annak a lehetősége, hogy az ár függhet a fizetés módjától, vagy hogy a vásárlóra bizonyos kedvezmények vonatkozhatnak. Összegezve, sok érdekes munka még hátramaradt.

Az utolsó feladat a kiemelt ajánlatok összehasonlítása. Tekintsük például az alábbi három ajánlatot:

A: 2,4 GHz CPU, 512 Mbyte RAM, 80 Gbyte diszk, DVD, CDRW, 350000 Ft

B: 2,0 GHz CPU, 1 Gbyte RAM, 120 Gbyte diszk, DVD, CDRW, 400000 Ft

C: 2,2 GHz CPU, 512 Mbyte RAM, 80 Gbyte diszk, DVD, CDRW, 400000 Ft

A C-t az A **dominálja**. Az A olcsóbb és gyorsabb, különben ugyanolyan. Általánosságban azt mondjuk, hogy X dominálja Y-t, ha X legalább egy attribútum esetében kedvezőbb értékű, és semmilyen más attribútum esetében sem rosszabb. Az A és a B közül egyik sem dominálja a másikat. Hogy megállapítsuk, melyik a jobb, tudnunk kell, hogy a vásárló hogyan mérlegeli a CPU sebességét és árát a memóriához és a diszkkapacitáshoz képest. Többszörös attribútum esetében a preferenciák általános tárgyalását a 16.4. alfejezet tartalmazza. Most az ágensünk egyszerűen ki fogja jelezni a vásárló lefrásával megegyező, nem dominált ajánlatok listáját. Ebben a példában sem az A, sem a B nem dominált. Jegyezzük meg, hogy ez az eredmény azon a feltevésen alapul, hogy mindenki jobban kedveli az olcsóbb árakat, a gyorsabb processzorokat és a nagyobb tárat. Bizonyos attribútumok, mint például a notebook képernyőmérete, a vásárló partikuláris preferenciáitól függenek (a hordozhatóság a láthatósággal szemben). Az ilyenekről a vásárló ágensnek a felhasználót kell kérdeznie.

A leírt vásárló ágens egy egyszerű rendszer, és számos finomítása lehetséges. Mégis elegendő a képessége arra, hogy a megfelelő területspecifikus tudással a vásárlót ténylegesen tudja segíteni. Deklaratív konstrukciója miatt könnyen felskálázható bonyolultabb alkalmazásokhoz. Ennek az alfejezetnek a fő célja az volt, hogy kimutassuk, hogy bizonyos tudásreprezentáció – különösképpen a termék hierarchiája – szükséges az ilyen típusú ágensek számára, és ha már az ilyen formájú tudás rendelkezésre áll, a többi már nem is olyan nehéz egy tudásalapú ágens számára.

## 10.6. KÖVETKEZTETŐ RENDSZEREK KATEGÓRIÁK SZÁMÁRA

Láttuk, hogy a kategóriák alapvető építőkockái bármely nagyobb méretű reprezentációs sémának. Ebben az alfejezetben a kategóriák szervezéséhez és a kategóriákkal való következtetéshez célzottan megtervezett rendszereket mutatjuk be. Két szoros kapcsolatban lévő rendszerssaládról beszélhetünk. A **szemantikus hálók** (*semantic network*) grafikus segítséget nyújtanak a tudásbázis vizualizálásában, és hatékony algoritmusokat biztosítanak, hogy egy objektum tulajdonságait a kategóriához való tartozásából kikövetkeztessük. A **leíró logikák** (*description logics*) formális nyelvet adnak a kategóriadefiníciók konstruálásához és kombinálásához, valamint hatékony algoritmusokat annak előtöntéséhez, hogy a kategóriák között fennáll-e a részhalmaz- és a szuperhalmaz-reláció.

### Szemantikus hálók

1909-ben Charles Peirce egy **egzisztenciális gráfoknak** (*existential graphs*) elnevezett grafikus jelölésrendszert javasolt, ami az ő elnevezése szerint a „jövő logikája”. Ezzel kezdetét vette a „logika” és a „szemantikus hálók” szószólónak hosszú vitája. Sajnos a vita elfedte azt a tényt, hogy a szemantikus hálók – legalább a jól definiált szemantikával ellátottak – a logika *egy formáját* jelentik. Bizonyosfajta állítások számára a szemantikus hálók nyújtotta jelölés gyakran kényelmesebb, azonban ha az „emberi interfész” problémáktól eltekintünk, a mögötte felsorakozó fogalmak – az objektumok, a relációk, a kvantifikálás stb. – mind ugyanazok.

A szemantikus hálóknak több változata van, mindegyik képes azonban az egyedi objektumokat, az objektumok kategóriáit és az objektumok közötti relációkat reprezentálni. A tipikus grafikus jelölés objektumok, illetve kategóriák neveit ovális keretekben vagy dobozokban mutatja, címkézett élekkel összekapcsolva. A 10.9. ábrán például a *Mária* és a *NőneműSzemély* között *Tagja* kapcsolat van, ami annak a logikai állításnak felel meg, hogy *Mária* ∈ *NőneműSzemély*. Hasonlóan a *Mária* és a *János* közötti *Húga* kapcsolat annak az állításnak felel meg, hogy *Húga*(*Mária, János*). A kategóriákat a *Részalmaza* éssel kapcsolhatjuk össze. A krumplik és a nyilak rajzolása annyira szórakoztató, hogy könnyű kísérésbe esni. Tudjuk például, hogy minden személy anyja egy nőnemű személy, szabad-e akkor a *Személy*-től a *NőneműSzemély*-ig egy *Anyja* élt húzni? A válasz nem, mert hogyan az *Anyja* egy reláció egy személy és az anyja között, a kategóriáknak viszont nincsen anyuk.<sup>9</sup> Emiatt a kapcsolatra a 10.9. ábrán egy külön jelölést használtunk – egy kettős vonallal történő bekeretezést. Ez a kapcsolat azt állítja, hogy:

$$\forall x \ x \in \text{Személy} \Rightarrow [\forall y \text{ Anyja}(x, y) \Rightarrow y \in \text{NőneműSzemély}]$$

Esetleg azt is állíthatnánk, hogy egy személynek két lába van, azaz:

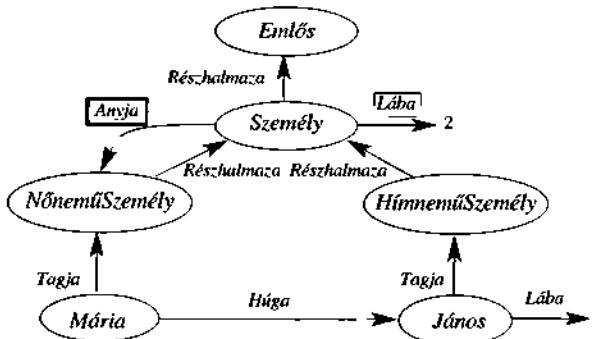
$$\forall x \ x \in \text{Személy} \Rightarrow \text{Lába}(x, 2)$$

Mint korábban, most is óvatosnak kell lennünk, nehogy azt állítsuk, hogy kategóriának van két lába. A 10.9. ábra egyszeres vonallal végzett keretezése jelzi az egy kategória minden tagjának tulajdonságaira vonatkozó állítást.

A szemantikus háló jelölésrendszere igen alkalmas a 10.2. alfejezetben bevezetett típusú öröklődéses (*inheritance*) következtetés végrehajtására. Így személy révén, Mária örökli azt a tulajdonságot, hogy két lába van. Ahhoz, hogy kitaláljuk, hány lába van Márának, az öröklődéses algoritmus követi a *Tagja* élt a *Máriá*-tól az őt tartalmazó kategóriáig, majd a *Részalmaza* élt a hierarchiába felfelé, amíg egy olyan kategóriát nem talál meg, amelyhez létezik egy bekeretezett *Lába* él – ebben ez esetben ez *Személy* kategória lesz. Ennek az öröklődéses algoritmusnak az egyszerűsége és hatékonysága a logikai tételelbizonyítóhoz képest, a szemantikus hálók egyik fő vonzereje volt.

Az öröklődés komplikálódik, amikor egy objektum egyszerre több kategóriához tartozhat, vagy pedig amikor egy kategória több kategória részhalmaza. Ezt az esetet többszörös öröklődésnek (*multiple inheritance*) nevezzük. Az ilyen helyzetben a öröklődéses algoritmus esetleg két vagy több, egymással ellentétes választ talál a felkérésre. Ennél fogva a többszörös öröklődés bizonyos objektumorientált programozási (*object-oriented programming, OOP*) nyelvekből, mint amilyen például a Java, ami az osztályhierarchiában öröklődést használ, ki van zárva. Szemantikus hálókban a többszörös öröklődést általában megengedjük, de ennek részletes tárgyalását a 10.7. alfejezetre hagyjuk.

<sup>9</sup> Bizonyos rendszerek nem tudtak különbséget tenni egy kategória tagjának tulajdonságai és kategória mint egész tulajdonságai között. Ez egyenes úton vezethet inkonzisztenciákhoz, ahogy ezt Drew McDermott kiemelte az „Artificial Intelligence Meets Natural Stupidity” című cikkében (McDermott, 1976). Más gyakori probléma volt a részhalmaz és a tagság kapcsolatoknál alkalmazott *IsA* választással, az angol használatnak megfelelően: „a cat is a mammal” (a macska egy emlős) és „Fifi is a cat” (Fifi egy macska). E témaról többet lásd 10.25. feladat.



**10.9. ábra.** Egy szemantikus háló négy objektummal (János, Mária, 1 és 2) és négy kategóriával. A relációkat a címkezett élek jelölik

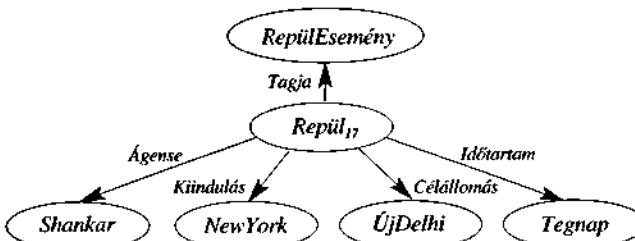
A következtetés másik közönséges formája az **inverz kapcsolatok (inverse links)** használata. Például a *VanHúga* a *Húga* inverze, azaz:

$$\forall p, s \ VanHúga(p, s) \leftrightarrow Húga(s, p)$$

Ezt az állítást megfogalmazhatjuk szemantikus hálóval, feltéve hogy a kapcsolatok **reifikáltak (reified)**, azaz önálló objektumnak tekinthetők. Így például *VanHúga* objektumot az *Inverze* éssel lehetne összekapcsolni a *Húga* objektummal. Ha egy lekérdezés azt firtatná, hogy ki Jánosnak a *Húga*, az öröklődéses algoritmus felfedezheti, hogy a *VanHúga* a *Húga* inverze, és a lekérdezést úgy megválaszolhatja, hogy a *VanHúga* kapcsolatot János-tól Máriá-ig követi végig. Az inverz információ nélkül esetleg szükséges lehetne az összes nőnemű személyt megvizsgálni, van-e netán *Húga* kapcsolata Jánossal. Ez amiatt van így, mert a szemantikus háló közvetlen indexelést csak az objektumok, a kategóriák és a belük induló kapcsolatok esetén biztosít. Az elsőrendű logika nyelvében ez annak felel meg, mintha a tudásbázist a predikátumok csak első argumentumai szerint indexelnénk.

Az olvasó talán felfedezte már a szemantikus háló jelölésnek egy nyilvánvaló hátrányát az elsőrendű logikához képest. A krumplik közötti élek csak *bináris* relációkat reprezentálnak. A *Repiiil*(Shankar, New York, Újdelhi, Tegnap) állítást a hálóban közvetlenül megfogalmazni nem tudjuk. Az  $n$  értékű állítások hatását megkaphatjuk azonban, ha az állítást magát egy, az események megfelelő kategóriájához tartozó eseményként reifikáljuk (lásd 10.3. alfejezetet). A 10.10. ábra ehhez a konkrét eseményhez tartozó szemantikus hálóstruktúrát mutat. Jegyezzük meg, hogy a bináris relációkra vonatkozó korlátozás a reifikált fogalmak gazdag ontológiáját teszi szükségeséssé. Az ebben a fejezetben kifejlesztett ontológia zöme ténylegesen a szemantikus hálós rendszerekben gyökerezik.

Állítások reifikálása lehetővé teszi, hogy az elsőrendű logika minden rögzített, függvénymentes atomi állítása reprezentálható szemantikus hálós jelöléssel. Univerzálisan kvantifikált állítások bizonyos típusai reprezentálhatók a kategóriákra alkalmazott inverz kapcsolatok, valamint egyszeresen és kétszeresen bekeretezett élek alkalmazásával, ettől még persze az elsőrendű logikától messze vagyunk. A negálás, a diszjunkció, a beágazott függvényszimbólumok és az egzisztenciális kvantifikálás mind hiányzik. Manapság lehetséges a jelölés kiterjesztése, hogy az az elsőrendű logikával ekvivalens legyen –



**10.10. ábra.** A szemantikus háló egy részlete, amely a *Repül(Shankar, NewYork, Újdelhi, Tegnap)* logikai állítás reprezentációját valósítja meg

ilyenek például a Pierce-féle egzisztenciális gráfok vagy Hendrix-féle particionált szemantikus hálók (Hendrix, 1975) – ez viszont a szemantikus hálók fő előnyét – az öröklődéses folyamat egyszerűségét és átláthatóságát – semmisíti meg. A tervezők nagy hálókat építhetnek, és még minden lehetnek jó ötleteik, hogy mely lekérdezések lesznek hatékonyak, mert (a) az öröklődéses eljárás lépésein könnyű vizualizálni, (b) egyes esetekben a lekérdezés nyelve olyan egyszerű, hogy bonyolult lekérdezéseket feltenni nem is lehet. Azokban az esetekben, amikor a kifejezőről túlságosan korlátozott, számos szemantikus hálórendszer a procedurális kiegészítéshez (*procedural attachment*) folyamodik, hogy a hiányokat kitöltsse. A procedurális kiegészítés egy olyan módszer, amikor egy bizonyos relációra vonatkozó lekérdezés (néha állítás) egy, a relációhoz megtervezett speciális eljárás, és nem egy általános öröklődéses algoritmus meghívását eredményezi.

A szemantikus hálók egyik legfontosabb aspektusa, hogy képesek a kategóriák számára az alapértelmezett értékeket (*default values*) reprezentálni. A 10.9. ábra gondos végignézésével észrevehetjük, hogy Jánosnak egy lába van annak ellenére, hogy ő egy személy, és minden személynek két lába van. Egy tisztán logikai tudásbázisban ez egy ellentmondás lenne, de a szemantikus hálóban az az állítás, hogy minden személynek két lába van, egy alapeseti állítás, azaz egy személyről feltételezzük, hogy két lába van, ha nincs ezzel ellentmondó specifikusabb információ. Az alapeseti szemantikát természetesen az öröklődéses következtetés kényszeríti ki, mert a kapcsolatokat magától az objektumtól (itt Jánostól) felfelé követi, és megáll, ha csak egy értéket meg nem talál. Azt mondjuk, hogy az alapeseti értéket egy specifikusabb érték *feltürlírhatja* (*overridden*). Vegyük észre, hogy lábak számát felül tudnánk írni egy *EgyLábúSzemély* kategória létesítésével, ami a *Személy Réshalmaza* lenne, és amelynek János lenne a tagja.

A háló szigorúan logikai szemantikáját megtarthatjuk, ha azt mondjuk, hogy a *Lába* állítás a *Jánosra* vonatkozó kivételt tartalmazza:

$$\forall x \ x \in \text{Személy} \wedge x \neq \text{János} \Rightarrow \text{Lába}(x, 2)$$

Egy rögzített háló esetén ez szemantikusan megfelelő, azonban ha sok kivételünk lesz, kevésbé tömör lesz a felírásunk, mint a hálós jelölés maga. Több állítással frissített háló esetén az ilyen megközelítés sajnos kudarcra van ítélezve – amit valójában mondani szeretnénk az az, hogy minden, egyelőre még ismeretlen egylábú személy szintén kivételek számít. Ezzel a témaival és az alapeseti következtetéssel általában bővebben a 10.7. alfejezet foglalkozik.

## Leíró logikák

Az elsőrendű logika szintaxisát úgy tervezték, hogy egyszerű legyen objektumokról kijelentéseket tenni. A leíró logikák (*description logics*) olyan jelölések, amelyeket a kategóriák definícióinak és tulajdonságainak könnyebb leírására terveztek. A leíró logikák a szemantikus hálókból alakultak ki, válaszul arra a nyomásra, hogy a hálók jelentését formalizáljuk, hangsúlyozva emellett a taxonómia szervezési elvként való használatát.

A leíró logika alapvető következtetési feladata a **részvizsgálat** (*subsumption*) – annak ellenőrzése, hogy egy kategória része-e egy másiknak a definíciói alapján – és az **osztályozás** (*classification*) – annak ellenőrzése, hogy egy objektum egy kategóriába tartozik-e. Bizonyos rendszerek egy kategóriadefiníció **konziszenciavizsgálatát** (*consistency*) is tartalmazzák arra, hogy a tagsági kritérium logikailag kielégíthető-e.

<i>Fogalom</i>	$\rightarrow$	Dolog   Fogalomnév És( <i>Fogalom</i> , ...) Mind( <i>SzerepNév</i> , <i>Fogalom</i> ) Legalább( <i>Egész</i> , <i>SzerepNév</i> ) Legfeljebb( <i>Egész</i> , <i>SzerepNév</i> ) Betölt( <i>SzerepNév</i> , <i>EgyediNév</i> , ...) UgyanazMint( <i>Ut</i> , <i>Ut</i> ) EgyBelőle( <i>EgyediNév</i> , ...) [ <i>SzerepNév</i> , ...]
<i>Ut</i>	$\rightarrow$	

10.11. ábra. A leírások szintaxisa a CLASSIC nyelv egy részhalmazában

A CLASSIC nyelv (Borgida és társai, 1989) egy tipikus leíró logika. A 10.11. ábra mutatja a CLASSIC leírások szintaxisát (az érthetőség kedvéért a nyelv kulcsszavait magyar fordításban adjuk meg – *a ford.*).<sup>10</sup> Azt az állítást például, hogy az agglegények nőtlen, felnőtt férfiak, következőképpen írhatnánk le:

$$\text{Agglegény} = \text{És}(\text{Nőtlen}, \text{Felnőtt}, \text{Férfi})$$

Ennek elsőrendű logikai ekvivalense a következő lenne:

$$\text{Agglegény}(x) \Leftrightarrow \text{Nőtlen}(x) \wedge \text{Felnőtt}(x) \wedge \text{Férfi}(x)$$

Vegyük észre, hogy a leíró logika hatékonyan teszi lehetővé azt, hogy a predikátumokon direkt logikai műveleteket hajtsunk végre anélkül, hogy először mondatokat alkotnánk, amiket azután összekapsolunk kötőszavakkal. Bármilyen leírás a CLASSIC-ban kifejezhető elsőrendű logikával, de bizonyos leírások a CLASSIC-ban sokkal világosabbak. Például az emberek egy olyan halmazában leírásához, aiknek legalább három fiuk van, és a fiúk mind munkanélküliek, és a feleségeik orvosok, valamint maximum

<sup>10</sup> Vegyük észre, hogy a nyelv *nem* engedi meg annak állítását, hogy az egyik fogalom vagy kategória egy másiknak a része. Ez egy átgondolt irányvonal: a kategóriák közötti részviszonyt a kategóriák bizonyos aspektusaiból kell tudni levezetni. Ha ez nem sikerül, akkor valami hiányzik a leírásból.

két lányuk van, akik vagy fizika, vagy matematika tanszéken professzorok, azt használnánk, hogy:

*És(Férfi, Legalább(3, Fiú), Legfeljebb(2, Lány),*

*Mind(Fiú, És(Munkanélküli, Házas, Mind(Feleség, Orvos))),*

*Mind(Lány, És(Professzor, Betölt(Tanszék, Fizika, Matematika))))*

Gyakorlatként meghagyjuk ennek lefordítását elsőrendű logikára.

A leíró logikák talán egyik legfontosabb aspektusa a következtetés nyomon követethetőségére tett hangsúly. Egy problémaeset megoldása a lefrásával kezdődik, majd annak megkérdezésével folytatódik, hogy része-e a lehetséges megoldás a kategóriák egyikének. Standard elsőrendű logikában a válasz idejének megjósolása gyakran lehetetlen. Gyakran a felhasználóra vár a reprezentáció olyan átalakítása, hogy kikerülje azokat a mondathalmazokat, amelyek miatt úgy tűnik, hogy a rendszernek több hétre telik a probléma megoldása. A leíró logikák viszont azt biztosítják, hogy a részvizsgálat megoldható legyen polinomiális idő alatt a probléma leírásától függően.<sup>11</sup>

Ez a gyakorlatban csodálatosan hangzik, míg az ember rá nem jön, hogy a következő két konzekvencia valamelyike lehetséges csak: a nehéz problémákat vagy nem lehet kifejezni, vagy exponenciálisan nagy leírást igényelnek! Mindenesetre a kezelhetőség eredményei vetnek némi fényt arra, hogy milyen konstrukciók okoznak bajt, és így segítik a felhasználót annak megértésében, hogyan viselkednek különböző reprezentációk. Például a leíró logikából általában hiányzik a *negálás* és a *diszjunkció*. Ezek mindenkorra kényszerítik az elsőrendű logikai rendszert, hogy lényegében egy exponenciális eset analízisen menjen keresztül a teljesség biztosítása érdekében. A negálás és a diszjunkció ugyanezen ok miatt van kizárvva a Prolog nyelvből is. A CLASSIC csak egy korlátozott formájú diszjunkciót engedélyez a *Kitölő* és az *EgyBelőle* szerkezetekben, ami lehetővé tesz diszjunkciót explicit módon megszámolt egyedek felett, de nem a leírások felett. Diszjunktív leírással az egymásba ágyazott definíciók könnyen exponenciális számú alternatív úthoz vezethetnek, amelyekben az egyik kategória könnyen része lehet egy másiknak.

## 10.7. KÖVETKEZTETÉS ALAPÉRTELMEZETT INFORMÁCIÓVAL

Az előbbi alfejezetben láttunk egy egyszerű példát egy alapértelmezett státussal rendelkező állításra: „az embereknek két lába van”. Ezt az alapértelmezett értéket specifikusabb információval, mint például „a Kékszakállúnak egy lába van”, felülírhatjuk. Láttuk, hogy a szemantikus háló öröklődési mechanizmusa az alapértelmezett értékek felülírását egyszerű és természetes módon oldja meg. Ebben a részben az alapértelmezett értékeket mélyebben tanulmányozzuk annak érdekében, hogy az alapértelmezett értékek *szemantikáját* alaposabban megértsük, és ne szorítkozzunk csupán a procedurális mechanizmus megadására.

<sup>11</sup> A CLASSIC nyelv a gyakorlatban hatékony részvizsgálatot biztosít, a legrosszabb eset azonban exponenciális futásidejű.

## Nyitott és zárt világok

Tegyük fel, hogy egyetemen a számítástudományi tanszék hirdetőtábláját nézzük, és azt az üzenetet látjuk, hogy: „Az alábbi tantárgyakat fel lehet venni: SZT 101, SZT 102, SZT 106 és VIM 101.” Próbáljuk most megválaszolni, vajon hány tárgyat lehet felvenni? Ha a válasza az, hogy „négy”, ez megegyezik egy tipikus adatbázis válaszával. Ha adott egy relációs adatbázisa:

$$\begin{aligned} & \text{Tantárgy(SZT, 101), Tantárgy(SZT, 102), Tantárgy(SZT, 106),} \\ & \quad \text{Tantárgy(VIM, 101)} \end{aligned} \tag{10.2}$$

négy állítás megfelelőjével, a count \* from Tantárgy SQL-felkérés 4-es válasszal tér vissza. Másfelől egy elsőrendű logikai rendszer válasza az lenne, hogy „egy és végtelen között valahány”, és nem „négy”. Ennek magyarázata, hogy a *Tantárgy* állítás nem zárja ki, hogy más, nem említett tantárgyakat is fel lehessen venni, és azt sem, hogy az említett tantárgyak minden különböző.

A példa mutatja, hogy az adatbázisok és az emberi kommunikáció konvenciója az előrendű logikától legalább két aspektusban különbözik. Először, az adatbázisok (és az emberek) feltételezik, hogy a megadott információ *teljes*, vagyis hogy az igazként ki nem jelentett rögzített atomi formulákról feltételezhetjük, hogy hamisak. Ez az ún. **zárt világ feltételezés** (*closed-world assumption*, CWA). Másodszor, elfogadjuk általában, hogy különböző nevek különböző objektumokat jelentenek. Ez az ún. **egyedi elnevezések feltételezés** (*unique names assumption*, UNA), amit a cselekvés-nevek kontextusában először a 10.3. alfejezetben vezettünk be.

Az előrendű logika e konvencióhoz nem folyamodik, így precízebbnek kell lennie. Hogy kijelentsük, csak négy különböző tantárgyat lehet választani, azt kellene írni, hogy:

$$\begin{aligned} \text{Tantárgy}(d, n) \Leftrightarrow [d, n] = [\text{SZT}, 101] \vee [d, n] = [\text{SZT}, 102] \\ \quad \vee [d, n] = [\text{SZT}, 106] \vee [d, n] = [\text{VIM}, 101] \end{aligned} \tag{10.3}$$

A (10.3) egyenletet a (10.2) **lezárásának**<sup>12</sup> (*completion*) nevezik. A lezárás általában minden predikátumhoz egy definíciót, egy „akkor és csak akkor” állítást rendel hozzá. minden definícióban mindegyik, a predikátumot a fejében tartalmazó definit klózhoz egy diszfunkció fog tartozni.<sup>13</sup> A lezárást általánosságban az alábbi módon szerkesztik meg:

1. Gyűjtsük ki az ugyanolyan ( $P$ ) predikátum névvel és ( $n$ ) aritással rendelkező klózokat.
2. minden klóz az ún. **Clark Normál Formára** (*Clark Normal Form*) transzformáljuk: cseréljük a:

$$P(t_1, \dots, t_n) \leftarrow \text{Törzs}$$

kifejezést, ahol a  $t_i$ -k termek, a:

$$P(t_1, \dots, t_n) \leftarrow \exists w_1 \dots w_m [v_1, \dots, v_n] = [t_1, \dots, t_n] \wedge \text{Törzs}$$

kifejezésre, ahol a  $v_i$ -k az újonnan bevezetett változók és a  $w_i$ -k az eredeti klóz változói. Használjuk minden klóz esetén a  $v_i$  változók ugyanazon halmozát. Eredményként a:

<sup>12</sup> A felfedezőjéről, Keith Clarkról néha Clark-lezárásnak is nevezik.

<sup>13</sup> Jegyezzük meg, hogy ez egyben a 10.3. alfejezelben megadott követő állapot axiómák alakja is.

$$P(v_1, \dots, v_n) \leftarrow B_1$$

⋮

$$P(v_1, \dots, v_n) \leftarrow B_k$$

klózok halmazát kapjuk.

3. Kombináljuk ezeket össze egy nagy diszjunktív klózzá:

$$P(v_1, \dots, v_n) \leftarrow B_1 \vee \dots \vee B_k$$

4. Zárjuk le azáltal, hogy a  $\leftarrow$ -at ekvivalenciára cseréljük:

$$P(v_1, \dots, v_n) \Leftrightarrow B_1 \vee \dots \vee B_k$$

A Clark-lezárás egy példáját – szabályokat és rögzített tényeket tartalmazó tudásbázis esetén – a 10.12. ábrán láthatjuk. Hogy hozzáadhatunk az egyedi elnevezések feltételezést is, egyszerűen adjuk meg az azonossági reláció Clark-lezárását, ahol csak azok az ismert tények, hogy  $SZT = SZT$  és  $101 = 101$  stb. Ennek megvalósítását gyakorlás közben meghagyjuk az olvasónak.

Horn-klózok	Clark-lezárás
$Tantárgy(SZT, 101)$	$Tantárgy(d, n) \Leftrightarrow [d, n] = [SZT, 101]$
$Tantárgy(SZT, 102)$	$\vee [d, n] = [SZT, 102]$
$Tantárgy(SZT, 106)$	$\vee [d, n] = [SZT, 106]$
$Tantárgy(VIM, 101)$	$\vee [d, n] = [VIM, 101]$
$Tantárgy(VIM, i) \leftarrow Egész(i)$	$\vee \exists i [d, n] = [VIM, 101] \wedge Egész(i)$
$\wedge 101 \leq i \wedge i \leq 130$	$\wedge 101 \leq i \wedge i \leq 130$
$Tantárgy(SZT, m + 100) \Leftarrow$	$\vee \exists m [d, n] = [SZT, m + 100]$
$Tantárgy(SZT, m) \wedge 100 \leq m$	$\wedge Tantárgy(SZT, m) \wedge 100 \leq m$
$\wedge m \leq 200$	$\wedge m \leq 200$

10.12. ábra. Horn-klózok egy halmazának Clark-lezárása. Az eredeti Horn-program (balra) négy tantárgyat említ explicit módon, és azt is állítja, hogy minden egészre a 101 és a 130 között létezik egy matematikai tárgy, meg azt is, hogy minden SZT tárgyhoz a 100-as (BSc) sorozatban létezik egy megfelelő tantárgy a 200-as (MSc) sorozatban. A Clark-lezárás (jobbra) azt mondja, hogy más tantárgy nincs is. A lezárással és az egyedi elnevezések feltételezéssel (valamint az *Egész* predikátum nyilvánvaló definíciójával) együtt eljutunk a kívánt konklúzióig, miszerint pontosan 36 tantárgy van: 30 matematikai és 6 SZT tárgy.

A zárt világ feltételezés lehetővé teszi, hogy megtaláljuk egy reláció **minimál-modelljét (minimal model)**. Ez azt jelenti, hogy a *Tantárgy* relációhoz a legkevesebb elemet tartalmazó modellt találhatjuk meg. A (10.2) egyenletben a *Tantárgy* minimál-modellje négyelemű, kevesebb már ellentmondáshoz vezet. Horn-klóz tudásbázisok esetén minden előforduló előtagban létezik egy *egyértelmű* minimálmodell. Jegyezzük meg, hogy az egyedi elnevezések feltételezés mellett ez az azonossági relációra is vonatkozik: minden term csakis saját magával azonos. Paradox módon ez azt jelenti, hogy a minimál modellek egyben maximálisak abban az értelemben, hogy annyi objektumot tartalmaznak, amennyi csak lehetséges.

Lehetséges egy Horn-program Clark-lezárását képezni, majd a következtetések levonása végett egy tételezéshez folyamodnak, óvatosan kell megválasztaniuk az általuk használt következtetést. Egy népszámlálási adatbázisban például, ésszerű a zárt világ feltételezést használni, ha a következtetés a városok populációjáról történik, de nyilván helytelen konklúzió lenne elfogadni, hogy a jövőben nem születnek gyerekek, csakis annak alapján, hogy az adatbázis a jövőbeli születési bejegyzéseket nem tartalmazza. A zárt világ feltételezés az adatbázist **teljessé (complete)** teszi abban az értelemben, hogy minden atomi lekérdezésre vagy pozitív, vagy negatív választ kapunk. Ha valamely tényállásról (mint például a jövőbeli születések) tényleg tudatlanok vagyunk, a zárt világ feltételezést használni nem szabad. Egy komplikáltabb tudásreprezentációs rendszerben a felhasználónak esetleg szabad lenne specifikálni a zárt világ feltételezés használatát leíró szabályokat.

## Negálás mint kudarc és stabil modell szemantika

A 7. és a 9. fejezetekben láttuk, hogy Horn-klóz formájú tudásbázisnak kedvező számítástechnikai tulajdonságai vannak. Sok alkalmazásban azonban nem kényelmes azt biztosítani, hogy klózok törzseiben csak pozitív literálok legyenek. Szeretnénk például azt mondani, hogy „Kimehetsz, ha nem esik”, anélkül hogy olyan predikátumokhoz kellene folyamodni, mint a *NemEsik*. Ebben a részben annak a lehetőségét kutatjuk, hogy a Horn-klózokhoz az explicit negálás egy formáját adjuk hozzá a **negálás mint kudarc (negation as failure)** ötletét felhasználva. Az ötlet az, hogy egy negatív „*not P*” literált igaznak „bizonyíthatunk”, hasonlóan, mint ahogy *P* bizonyítása kudarcba fulladhat. Ez az alapeseti következtetés egy formája, ami a zárt világ feltételezéshez szorosan kapcsolódik: tételezzük fel, hogy valami hamis, ha nem bizonyítható, hogy igaz. Hogy a negálás mint kudarc-ot a logikai „ $\neg$ ” operátorról megkülönböztethessük, megjelölésére a „*not*”-ot fogjuk használni.

A Prolog megengedi a *not* operátort egy klóz törzsében. Tekintsük például az alábbi Prolog programot:

```
IDEmeghajtó ← Meghajtó ∧ not SCSImeghajtó
SCSImeghajtó ← Meghajtó ∧ not IDEmeghajtó
SCSIvezérlő ← SCSImeghajtó
Meghajtó
```

Az első szabály azt mondja, hogy ha számítógépben merevlemez-meghajtónk van, és ez nem SCSI, akkor IDE-nek kell lennie. A másik szabály azt mondja, hogy ha ez nem IDE, akkor SCSI-nek kell lennie. A harmadik azt mondja, hogy az SCSI-meghajtó létezése egy SCSI-vezérlőt tételez fel, végül a negyedik kijelenti, hogy tényleg van egy meghajtó. Ennek a programnak két minimálmodellje van:

```
M1 = {Meghajtó, IDEmeghajtó}
M2 = {Meghajtó, SCSImeghajtó, SCSIvezérlő}
```

A minimálmodellek nem képesek a negálás mint kudarc-ot használó programok szándékolt szemantikáját kifejezni. Tekintstük a következő programot:

$P \leftarrow \text{not } Q$

Ennek két minimálmodellje van:  $\{P\}$  és  $\{Q\}$ . Az elsőrendű logika szemszögéből van ennek értelme, mivel a  $P \Leftarrow \neg Q$  a  $P \vee Q$ -val ekvivalens. A Prolog szemszögéből azonban aggódni kellene:  $Q$  soha nem jelenik meg a nyíl bal oldalán, hogyan lehet hát egy következmény?

Egy alternatíva a **stabil modell** (stable model), ami egy olyan minimálmodell, ahol minden atomnak igazolása (**justification**) van, azaz létezik hozzá olyan szabály, amelyben a fej az atom, és ahol a törzs minden literálja kielégített. Formálisan  $M$  egy  $H$  program egy stabil modellje, ha  $M$  a  $H$  az  $M$ -re vonatkozó redukáltjának (**reduct**) egy egyértelmű minimálmodellje. Egy  $H$  program redukáltját úgy definiáljuk, hogy  $H$ -ból először minden olyan szabályt törlünk, amely törzsében a  $\text{not } A$  literál szerepel, ahol  $A$  a modell része, majd a maradó szabályokban a negatív literálokat töröljük. Mivel  $H$  redukáltja most már egy Horn-kláz lista, egy egyértelmű minimálmodellel kell rendelkeznie.

A  $P \leftarrow \text{not } Q$  redukáltja a  $\{P\}$ -re nézve maga a  $\{P\}$ , aminek minimálmodellje  $\{P\}$ . Így  $\{P\}$  egy stabil modell. A  $\{Q\}$ -ra vonatkozó redukció egy üres program, aminek minimálmodellje  $\{\}$ .  $\{Q\}$  tehát nem stabil modell, mert  $Q$ -nak a (10.5) egyenletben nincs igazolás. Egy másik példa a (10.4) redukáltjára az  $M_1$ -re nézve:

```
IDEmeghajtó ← Meghajtó
SCSIvezérő ← SCSCImeghajtó
Meghajtó
```

Ennek minimálmodellje  $M_1$ , így  $M_1$  egy stabil modell. A **válaszhalmaz programozás** (answer set programming) a logikai programozásnak a negálás mint kudarc-cal bővített fajtája, amely úgy működik, hogy a logikai programot rögzített formába transzformálja, majd stabil modelleket (amelyeket válaszhalmazoknak – answer sets – is neveznek) keres, ítéletkalkulus modell-ellenőrzési technikákat felhasználva. A válaszhalmaz-programozás leszármazottja tehát mind a Prolognak, mind az olyan gyors ítéletkalkulus-beli kielégíthetőségi tételelbizonyítóknak, mint a WALKSAT. A válaszhalmaz programozást sikerrel alkalmazták tervkészítési problémákra, hasonlóan az ítéletkalkulus-beli kielégíthetőségi tételelbizonyítókhöz. A válaszhalmaz-programozás előnye más tervkészítőkkel szemben a rugalmassága: a tervkészítő operátorokat és a kényszereket logikai programokkal fejezi ki, és így azok nem kötődnek a konkrét tervkészítési formalizmus korlátozott formátumához. A válaszhalmaz-programozás hátránya ugyanaz, mint minden ítéletkalkulus szintű technikáé: ha az univerzumban túl sok objektum létezik, egy exponenciális lassulással kell számolnunk.

## Körülírás és alapeseti logika

Két példát láttunk, ahol látszólag természetes következtetési folyamatok megsértik a logika a 7. fejezetben bebizonyított **monotonitás** (monotonicity) tulajdonságát.<sup>14</sup> Az el-

<sup>14</sup> Emlékezzünk vissza, hogy a monotonitás megköveteli, hogy minden vonzatállítás igaz maradjon, amikor a tudásbázishoz új állításokat adunk hozzá. Azaz ha  $TB \models \alpha$ , akkor  $TB \wedge \beta \models \alpha$ .

ső példában egy szemantikus hálóban egy kategória összes tagja által megörökített tulajdonságot egy alkategóriára vonatkozó specifikusabb információ felülírhatja. A másik példában a zárt világ feltételezésből származtatott negált literálok pozitív literálok hozzáadása szintén felülbírálja.

Ha egy kicsit magunkba nézünk, rájövünk, hogy a monotonitás ilyen sérülései a józan ész következetetben igen elterjedtek. Úgy tűnik, az emberek gyakran „elhamarkodott következeteteket” vonnak le. Ha az utcán parkoló kocsit látunk, mindenki elhiszi, hogy a kocsinak négy kereke van, holott csak három kerék látszik (ha valaki a negyedik kerék létezésében kételkedik, akkor vegye fontolóra azt a kérdést, vajon a három látható kerék valódi-e, vagy csupán a valódinak egy papírmása.) A valószínűség-elmélet nyilván igazolhatja, hogy a negyedik kerék nagy valószínűséggel létezik, az emberek többségében az a lehetőség, hogy a gépkocsinak esetleg nincs négy kereke, egyáltalán fel sem merül, *hacsak valami új tényállás erre nem derít fényt*. A négy kerék konklúzióját tehát *alapesetként* kezeljük, a kételkedésre okot adó tények hiányában. Ha új tények érkeznek – észrevesszük például, hogy a tulajdonos a kereket elviszi, és a gépkocsi egy emelőn áll – akkor a konklúziót vissza kell vonni. Az ilyen következetetéről azt mondjuk, hogy **nemmonoton** (*nonmonoton*), mert a hiedelemek halmaza idővel, ahogy az új evideciák megjelennek, nem növekszik monoton módon. Az ilyen viselkedés leírására **nemmonoton logikákat** (*nonmonoton logics*) fejlesztettek ki, módosított igazságdefiniációval és vonzatrelációval. Két ilyen intenzíven tanulmányozott logikát fogunk megnézni: a körülírást és az alapeseti logikát.

A körülírást (*circumscription*) a zárt világ feltételezés erősebb és precízebb változatainak lehetne tekinteni. Az ötlet, hogy konkrét predikátumokat specifikálunk, amelyekről feltételezzük, hogy „amennyire csak lehetséges hamisak” – azaz minden objektumra hamisak, kivéve azokat, amelyekre tudjuk, hogy igazak. Tegyük fel például, hogy ki akarjuk jelenteni azt az alapételmezett szabályt, hogy a madarak tudnak repülni. Vezessük be az *Abnormális*<sub>1</sub>(*x*) predikátumot, és írjuk fel:

$$\text{Madár}(x) \wedge \neg \text{Abnormális}_1(x) \Rightarrow \text{Repül}(x)$$

Ha azt mondjuk, hogy az *Abnormális*<sub>1</sub>-et körül kell írni (*circumscribed*), a körülíró következető feltételezheti, hogy  $\neg \text{Abnormális}_1(x)$ , hacsak az *Abnormális*<sub>1</sub>(*x*)-et nem fogja igaznak találni. Ez lehetővé teszi, hogy a *Repül(Tweety)* konklúziót levonjuk a *Madár(Tweety)* premissza alapján, azonban a konklúzió csak addig igaz, amíg az *Abnormál(Tweety)*-t ki nem jelentjük.

A körülírást a **modellpreferencia-logika** (*model preference logic*) egy példájának is tekinthetjük. Az ilyen logikákban egy állítás vonzatrelációban van (alapeseti státuson), ha igaz a tudásbázis minden *preferált* modelljében, ellentétben a klasszikus logika igazságkövetelményével, hogy minden modellben legyen igaz. Körülírás esetében egy modell egy másiknál preferáltabb, ha kevesebb abnormális objektuma van.<sup>15</sup> Nézzük meg, hogy ez az ötlet hogyan működik szemantikus hálóban fellépő többszörös öröklődés esetén. A többszörös öröklődés standard példája az ún.

<sup>15</sup> A zárt világ feltételezésnél egy modell egy másiknál preferáltabb, ha kevesebb igaz atomja van – azaz a preferált modellek **minimálmodellek**. A CWA és a definit klöz tudásbázis között létezik természetes kapcsolat, mert az ilyen tudásbázisban az előrekesztő következetés révén elérő fix pont az egyértelmű minimálmodell (lásd 274. oldal).

„Nixon-gyémánt”.<sup>16</sup> Az eredete az a megfigyelés, hogy Richard Nixon egy kvéker vallási szektához tartozott (és így alapesetben pacifista) és republikánus is volt (és így alapesetben éppen nem pacifista). Ezt az alábbi módon írhatjuk fel:

$$\text{Republikánus}(\text{Nixon}) \wedge \text{Kvéker}(\text{Nixon})$$

$$\text{Republikánus}(x) \wedge \neg \text{Abnormális}_2(x) \Rightarrow \neg \text{Pacifista}(x)$$

$$\text{Kvéker}(x) \wedge \neg \text{Abnormális}_3(x) \Rightarrow \text{Pacifista}(x)$$

Ha az  $\text{Abnormális}_2$ -t és az  $\text{Abnormális}_3$ -at körülírjuk, két preferált modellt kapunk: az egyikben igaz az  $\text{Abnormális}_2(\text{Nixon})$  és a  $\text{Pacifista}(\text{Nixon})$ , a másikban igaz az  $\text{Abnormális}_3(\text{Nixon})$  és a  $\neg \text{Pacifista}(\text{Nixon})$ . A körülíró következtető rendszer tehát megfelelően tudatlannak mutatkozik a pacifista Nixon ügyében. Ha ezentúl szeretnénk kijelenteni, hogy a vallásos meggyőződések elsőbbséget élveznek a politikai nézetekkel szemben, a prioritásos körtüllírás (**prioritized circumscription**) formalizmusát használhatjuk, hogy azoknak a modelleknek adjunk elsőbbséget, ahol az  $\text{Abnormális}_3$  minimálizálva van.

Az **alapértelmezett logika** (**default logic**) egy olyan formalizmus, ahol **alapértelmezett szabályokat** (**default rules**) lehet írni feltételes, nemmonoton következtetések generálásához. Egy alapértelmezett szabály a következőképpen néz ki:

$$\text{Madár}(x) : \text{Repül}(x) / \text{Repül}(x)$$

A szabály jelentése, hogy ha a  $\text{Madár}(x)$  igaz, és ha a  $\text{Repül}(x)$  a tudásbázissal konzisz- tens, akkor a  $\text{Repül}(x)$ -et alapértelmezésben lekövetkeztethetjük. Általánosságban egy alapértelmezett szabály formája a:

$$P : J_1, \dots, J_n / C$$

ahol  $P$ -t előfeltételnek,  $C$ -t konklúziónak és a  $J_i$ -ket igazolásoknak nevezik – ha ezek közül bármelyik igazolhatóan hamis, a konklúziót levonni nem szabad. A  $C$ -ben és a  $J_i$ -ben előforduló minden változónak  $P$ -ben is meg kell jelennie. A Nixon-gyémánt példáját alapértelmezett logikában egy tényvel és két alapértelmezett szabállyal fejezzük ki:

$$\text{Republikánus}(\text{Nixon}) \wedge \text{Kvéker}(\text{Nixon})$$

$$\text{Republikánus}(x) : \neg \text{Pacifista}(x) / \neg \text{Pacifista}(x)$$

$$\text{Kvéker}(x) : \text{Pacifista}(x) / \text{Pacifista}(x)$$

Egy alapértelmezett szabály jelentésének interpretálásához az alapértelmezett elmélet **kiterjesztését** (**extension**) kell definiálnunk, az elmélet konklúziójának maximális hal-mazaként. Egy  $S$  kiterjesztés tehát az eredetileg ismert ténykből és az alapértelmezett szabályokból levont konklúzióhalmazból áll úgy, hogy  $S$ -ból semmilyen más konklúziót levonni már nem lehet, és  $S$ -beli alapértelmezett konklúzió minden igazolása  $S$ -sel konzisz-tens. A körülírásban tárgyalta preferált modellekhez hasonlóan a Nixon-gyémántra két lehetséges kiterjesztésünk van: egy, amelyben pacifista és egy, amelyben nem az. Léteznek prioritásos sémák, ahol egyes alapértelmezett szabályoknak precedenciát lehet biztosítani más szabályokkal szemben, és ezzel a nem egyértelmű helyzeteket valame-lyest fel lehet oldani.

<sup>16</sup> A „gyémánt” elnevezés a két úton futó öröklődési lánc gráfszerű alakjától származik. (A ford.)

1980 óta, amikor is a nemmonoton logikákat első ízben javasolták, igen nagy előrehabilitás történt a matematikai tulajdonságuk megértésében. A késői 1990-es évektől kezdve a logikai programozáson alapuló gyakorlati rendszerek igéretesnek bizonyultak a tudás-reprezentációs eszközök szerepében. Vannak azonban még meg nem válaszolt kérdések. Így például ha „a gépkocsiknak négy kerekük van” hamis, mit is jelent, ha egy ilyen állítás a tudásbázis része? Milyen az alapértelmezett szabályok egy jó halmaza? Ha minden szabályról külön-külön nem tudjuk eldöntení, hogy a tudásbázisunkhoz tartozik-e, akkor igen komoly problémánk van a modularitás hiányával. Végül, hogyan lehet alapértelmezett státussal rendelkező hiedelmeket a döntéshozatalban felhasználni? Ez talán az alapértelmezett következetés legnehezebb problémája. A döntések sokszor kompromisszumokkal járnak együtt, és így szükség van különböző cselekvések eredményeként megjelenő hiedelmek erősségét összehasonlítani. Azokban az esetekben, amikor ugyanilyen jellegű döntéset sokszor hozunk meg, lehetőség adódik, hogy az alapértelmezett szabályokat „küszöb-valószínűségi” állításokként értelmezzük. Például „a fékjeim rendben vannak” alapértelmezett szabály valójában azt jelenti, hogy „annak a valószínűsége, hogy a fékjeim rendben vannak, feltéve, hogy más információm nincs, elegendően magas ahhoz, hogy az optimális döntés számomra az, hogy induljak útnak a fékek ellenőrzése nélkül”. Amikor a döntés kontextusa megváltozik – például amikor egy súlyosan megrakott teherkocsit vezetünk egy lejtős hegyi úton –, az alapértelmezett szabály hirtelen alkalmatlanná válik, akkor is, ha semmilyen új evidenciánk nincs, ami azt sugallná, hogy a fékek hibásak. Az ilyen mérlegelések néhány kutatót arra vezettek, hogy mérlegeljék, hogyan ágyazzák be az alapértelmezett következetést a valószínűség-elméletbe.

## 10.8. IGAZSÁG-KARBANTARTÓ RENDSZEREK

Az előbbi részben arról érvettünk, hogy egy tudásreprezentációs rendszer által levont sok következetésnek csupán alapértelmezett és nem tökéletesen biztos státusa lehet. Az ilyen kikövetkeztetett konklúziókból néhány szükségszerűen hibásnak fog bizonyulni, és új információ fényében vissza kell vonnunk. Ez a **hiedelemrevízió** (belief revision) folyamata.<sup>17</sup> Tegyük fel, hogy a tudásbázis tartalmazza  $P$  állítást – az előre-csatolt algoritmussal levont alapértelmezett konklúziót vagy egyszerűen egy hibás feltételezést –, és szándékunkban áll végrehajtani az ÁLLÍT( $TB, \neg P$ )-t. Hogy az ellentmondást elkerüljük, először végre kell hajtani a VISSZAVON( $TB, P$ )-t. Ez elég egyszerűnek tűnik. Probléma akkor jelentkezik, ha  $P$ -ból *további* állításokat következtettünk le és jegyeztünk fel a tudásbázisba. A  $P \Rightarrow Q$  implikációt például felhasználtuk a  $Q$  tudásbázishoz való hozzáadásához. A nyilvánvaló „megoldás” – a  $P$ -ból kikövetkeztetett összes állítás visszavonása – kudarcba fut, mert ilyen állításoknak  $P$ -n túlmenően más igazolásuk is lehet. Ha  $R$  és  $R \Rightarrow Q$  is bekerült a tudásbázisba, akkor  $Q$ -t mégsem kell eltávolítanunk. Az **igazság-karbantartó rendszerek** (truth maintenance systems) ilyen bonyodalmakat kezelésére terveztek.

<sup>17</sup> A hiedelemrevíziót sokszor a **hiedelemfrissítéssel** (belief update) állítják szembe, amelyről akkor van szó, amikor a tudásbázis inkább a világbeli változások hatására, és nem a rögzített vilagról befutó új információ hatására kerül felülvizsgálásra. A hiedelemfrissítés összekapsolja a hiedelemrevíziót az időre és a változásra vonatkozó következetéssel. Kapcsolatban áll a 15. fejezetben leírt szűrés (filtering) folyamatával.

Az igazság-karbantartás egyszerű módja, hogy nyomon követjük, ahogyan sorban hozzáadjuk a mondatokat a tudásbázishoz, a mondatok  $P_1$ -től  $P_n$ -ig történő megszámolásával. Amikor a VISSZAVON( $TB, P_i$ ) hívás megtörténik, a rendszert a  $P_i$  hozzáadása előtti állapotába állítjuk vissza, a  $P_i$ -t és a  $P_i$ -ból levont összes következetést visszavonva. Ha szükséges, akkor a  $P_{i+1}$  és a  $P_n$  közötti mondatok ismét hozzáadhatók. Ez egyszerű megoldás, és garantálja, hogy a tudásbázis konzisztens lesz, de azt jelenti, hogy a  $P_i$  visszavonása  $n - i$  állítás visszavonását és újból kijelentését, valamint ezen állításokból történő következetések visszavonását és újból lefuttatását igényli. Olyan rendszerekben, ahol sok tény kerül hozzáadásra, az ilyen megoldás nem praktikus.

Hatókonyabb megközelítés az igazolásalapú igazság-karbantartó rendszer vagy JTMS (justification-based truth maintenance system). Egy JTMS-ben a tudásbázis minden mondatát olyan igazolással (justification) látják el, amely meghatározza azokat a mondatokat, amelyekből lekövetkeztették azokat. Például ha a tudásbázis már tartalmazza a  $P \Rightarrow Q$ -t, akkor az ÁLLÍT( $P$ ) hatására a  $Q$ -t hozzáadjuk, az  $\{P, P \Rightarrow Q\}$  igazolás-sal. Általánosságban bizonyos mondatoknak egy-nél több igazolása lehet. Az igazolások felhasználhatók hatékony visszavonásokra. A VISSZAVON( $P$ ) hívást követően, a JTMS pontosan azokat a mondatokat fogja a tudásbázisból eltávolítani, amelyeknél  $P$  minden igazolás része. Így, ha egy  $Q$  mondatnak a  $\{P, P \Rightarrow Q\}$  lenne az egyetlen igazolása, akkor törlődne; ha az  $\{P, P \vee R \Rightarrow Q\}$  igazolása is létezne, még mindig visszavonásra kerülne, de ha az  $\{R, P \vee R \Rightarrow Q\}$  igazolása is létezne, akkor megmaradna. Ily módon a  $P$  visszavonásához szükséges idő csak azon múlik, hogy  $P$ -ból hány állítást vezettünk le, és nem azon, hogy  $P$  hozzáadása után hány állítás került még be a tudásbázisba.

A JTMS feltételezi, hogy az egyszer figyelembe vett mondatokat máskor is használni fogjuk, így ahelyett, hogy törölnék a tudásbázisból, amikor elveszik az összes igazolását, csak megjelöljük őket, mint *kinitlevőket*. Ha egy későbbi állítás visszaállítja az igazolások egyikét, akkor a mondat ismét *bentlevő* jelölést kap. Ily módon a JTMS megtartja a használt összes következetési láncot, és nem kell újra lekövetkeztetnie azokat a mondatokat, amelyek igazolása ismét érvényessé válik.

Azon túl, hogy a TMSeK a hibás információt visszavonják, felhasználhatók a többszörös hipotetikus helyzet elemzésének a felgyorsításához is. Tegyük fel, hogy a Román Olimpiai Bizottság helyszíneket keres a 2048-ban Romániában megrendezendő olimpiai játékokhoz az úszó, a könnyűatlétikai és a lovás versenyszámok számára. Legyen az első hipotézis a következő: *Helyszín(Úszás, Pitești)*, *Helyszín(Könnyűatlétika, Bukarest)* és *Helyszín(Lovasverseny, Arad)*. A választás logisztikai következményeinek kiderítéséhez, és így a helyszínek kívántatosságának meghatározásához igen sok következetet szükséges. Ha figyelembe akarjuk venni a *Helyszín(Könnyűatlétika, Nagyszeben)*-t, a TMS elkerüli annak a szükségességét, hogy újra nullából indulunk ki. Egyszerűen viszavonjuk a *Helyszín(Könnyűatlétika, Bukarest)*-t, és hozzáadjuk a *Helyszín(Könnyűatlétika, Nagyszeben)*-t, és a TMS a szükséges revíziókat magára vállalja. A Bukarestre vonatkozó következetési láncok Nagyszeben esetén újrafelhasználhatók, feltéve, hogy a konklúziók ugyanazok lesznek.

Egy feltételezésalapú igazság-karbantartó rendszert (assumption-based truth maintenance system, ATMS) arra terveztek, hogy ezt a fajta hipotetikus világok közötti kontextusátkapcsolást kifejezzenten hatékonyá tegye. Egy JTMS-ben az igazolások karbantartása lehetővé teszi, hogy az egyik állapotból gyorsan egy másikba lépjünk át néhány visszavonással és kijelentéssel, de minden időpillanatban csak egyetlen állá-

potot reprezentálunk. Egy ATMS az összes állapotot egyszerre reprezentálja, amivel valaha is foglalkoztunk. Míg egy JTMS egyszerűen *bent-* vagy *kintlevőnek* jelöli a mondatokat, egy ATMS nyomon követi minden egyes mondatra, hogy a mondatot mely feltételezések tennék igazzá. Más szavakkal minden egyes mondatnak van egy címkéje, amely a feltételezés halmazokat tartalmazó halmazból áll. A mondat csak azon esetben áll fenn, amikor egy feltételezés halmaz összes feltételezése fennáll.

Az igazság-karbantartó rendszerek mechanizmust biztosítanak **magyarázatok** (*explanations*) generálásához is. Technikailag, *P* mondat magyarázatát olyan *E* mondatok halmazával definiáljuk, melyekre *E* maga után vonja *P*-t. Ha *E* mondatai igazak, akkor *E* egyszerűen *P* igazolásához egy szükségszerű alap. Azonban a magyarázat részei lehetnek **feltételezések** (*assumptions*) is – olyan állítások, amelyek nem igazak, de ha igazak lennének, elegendők lennének *P* igazolásához. Lehetséges például, hogy nincs elég tény bebizonyítani, hogy az autó nem fog indulni, de egy jó magyarázat része lehet az elromlott akkumulátor. Ez, összekapcsolva a gépkocsik működésére vonatkozó tudással, elegendő, hogy a megfigyelt jelenséget megmagyarázza. Az esetek többségében egy minimális *E* magyarázatot fogunk preferálni, azaz olyan magyarázatot, amelynek nincs olyan megfelelő részhalmaza, amelyik szintén egy magyarázat lenne. Egy ATMS a „gépkoci nem indul” problémához magyarázatokat generál azáltal, hogy tetszőleges sorrendben feltételezéset tesz (mint például „benzin a tankban” vagy „hibás az akkumulátor”), akkor is, ha azok egy része ellentmondásos. Elég majd a „gépkoci nem indul” állítás címkéjét megnézni, hogy az állítást igazoló feltételezések halmazát megismerjük.

Az igazság-karbantartó rendszereket megvalósító algoritmusok egy kicsit komplikáltak, és itt nem tárgyaljuk őket. Az igazság-karbantartási probléma komplexitása legalább olyan nagy, mint az ítéletlogikai következtetés – azaz NP-teljes. Ezért nem várható el az, hogy az igazság-karbantartás csodászer legyen. Ugyanakkor, ha körültekintően használják, a TMS-rendszer lényegesen javíthat egy logikai rendszer komplex környezeteket és hipotéziseket kezelő képességén.

## 10.9. ÖSSZEFoglalás

Ez a fejezet eddig a könyv legrészletesebb fejezete. Azzal, hogy sokféle tudás reprezentálásának részleteivel foglalkozunk, remélhetően érzékelhetnival tudtuk az olvasóval, hogy hogyan hozhatók létre valódi tudásbázisok. A lényegi témaik az alábbiak:

- Nagyméretű tudásreprezentáció egy általános rendeltetésű ontológiát igényel, a specifikus problématerületek szervezéséhez és az egymással való összekapcsolásához.
- Egy általános ontológiának a tudás széles választékát kell lefednie, és elvben képesnek kell lennie bármilyen problématerület kezelésére.
- Bemutattunk egy **felső ontológiát** (*upper ontology*), amely a kategóriákon és az eseménykalkulussal alapul. Foglalkoztunk strukturált objektumokkal, az idővel és a térrrel, a változással, folyamatokkal, szubsztanciákkal és hiedelmekkel.
- A cselekvéseket, az eseményeket és az időt vagy szituációkalkulusban, vagy az olyan nagyobb kifejezőrejű reprezentációkban, mint az eseménykalkulus vagy a folyó esemény kalkulus fejezhetjük ki. Az ilyen reprezentációk révén egy ágens cselekvési terveket konstruálhat logikai következtetés segítségével.

- Egy ágens mentális állapotait a hiedelmeket jelentő füzérekkel lehet reprezentálni.
- Az internetes bevásárlási problématerület részletes elemzését mutattuk be általános ontológia bevetésével, megmutatva, hogy egy bevásárló ágens hogyan használhatja a problématerületi tudását.
- A kategóriahierarchia szervezéséhez speciális rendeltetésű rendszereket terveztek, mint például a **szemantikus hálók** (*semantic nets*) és a **leíró logikák** (*description logics*). A következtetés fontos esete az **öröklődés** (*inheritance*), mely révén az objektumok tulajdonságait a kategóriatagságuktól ki lehet következtetni.
- A logikai programokban a **zárt világ feltételezést** (*closed-world assumption*) implementáljuk, hogy rengeteg negatív információ megadását elkerülhessük. A legjobb, ha ezt alapeseti helyzetnek tekintjük, amit további információ felülríthat.
- A **nemmonoton logikák** (*nonmonotonic logics*), amilyen például a **körülírás** (*circumscription*) vagy az **alapeseti logika** (*default logic*), az általános alapeseti következtetés leírására készültek. Nemmonoton következtetést a **válaszhalmaz-programozás** (*answer set programming*) lényegesen gyorsítja, hasonlóan ahhoz, ahogy a WALKSAT megyorsítja az ítéletlogikai következtetést.
- Az **igazság-karbantartó rendszerek** (*truth maintenance systems*) hatékonyan kezelik a tudásfelfrissítést és a tudásrevíziót.

## Irodalmi és történeti megjegyzések

Hihetőnek hangzik (Briggs, 1985), hogy a formális tudásreprezentáció kezdete a sásztrai szanszkrit nyelvtanáról alkotott klasszikus indiai elméletekkel kezdődött az i. e. első évezredben.<sup>18</sup> A nyugati világban az ókori görög matematikusok által használt definíciók tekinthetők a tudásreprezentáció legkorábbi megjelenésének. Tény, hogy a műszaki szóhasználat vagy egy mesterséges nyelv kialakítása, bármilyen területről van is szó, tekinthető a tudásreprezentáció egyfajta formájának.

Az MI-ben a reprezentációról folytatott korai viták inkább a „*problémareprezentáció*”, mint a „*tudásreprezentációra*” irányultak [lásd például a misszionáriusok és kannibálok problémájáról szóló Amareltől származó elemzést (Amarel, 1968)]. Az 1970-es években az MI-ben a hangsúly a „szakértő rendszerek” („tudásalapú rendszereknek” is nevezett) fejlesztésén volt, amelyek a megfelelő problématerületi tudás birtokában, szűkebb definiált feladatok esetén, az emberi szakértők hatékonyságát megközelítették, vagy akár túl is szárnyálták. Az első szakértő rendszer, a DENDRAL, például a tömegspektrométer (a szerves vegyi anyagok struktúraelemzését segítő berendezés) kimeneti adatait képes volt szakértő vegyészek szintjén interpretálni (Feigenbaum és társai, 1971; Lindsay és társai, 1980). Bár a DENDRAL sikere meghatározó volt abban, hogy az MI kutatói ráébredjenek a tudásreprezentáció fontosságára, a benne alkalmazott reprezentációs formalizmusok igen specifikusak voltak, és kifejezetten a vegyészeti problématerülethez alakították ki azokat. Idővel a kutatók a szabványosított tudásreprezentációs formalizmusok és az ontológiák felé fordultak, hogy így a korábban még fel nem tárta területeken alkalmazandó új szakértő rendszerek előállítási nehézségeit csökkentsék. Emellett a kutatás révén olyan területre merészkezdetek ki, amit korábban a tudio-

<sup>18</sup> A szanszkrit nyelv több változata közül a sásztrai szanszkrit a vallás és a tudomány nyelve. (A *ford.*)

mányfilozófusok és a nyelvészettel foglalkozó filozófusok műveltek. Az elméletek „munkába” állításával az MI területén kialakult új diszciplína sokkal mélyebb és gyorsabb előrehaladáshoz vezetett a korábbi időkhöz képest, amikor ezek a problémák kizárolag a filozófia tárgyát képezték (bár időnként előfordult a spanyolviasz ismételt felfedezése is).

Átfogó taxonómiák, illetve osztályozások megalkotása az ókorai időig nyúlik vissza. Az osztályozási és kategorizálási sémák fontosságát nyomatékosan hangsúlyozta Arisztotelész. Halála után hallgatói révén megszerkesztett *Organon*, amely logikai munkák gyűjteménye volt, tartalmazta a *Kategóriák* c. tanulmányt is, amelyben Arisztotelész megkísérelt kimerítő osztályozást adni arra, amit ma felső ontológiának nevezünk. Alacsonyabb szintű osztályozás céljára vezette a **faj** (*genus*) és a **fajta** (*species*) fogalmakat, bár e fogalmak akkor nélkülöztek a ma velük asszociált precíz és specifikusan biológiai jelentést. A biológiai osztályozás mai rendszerét, beleértve a „binomiális nömenklatúrát” (technikai értelemben a faj-, fajtaalapú osztályozást) is, Carolus Linnaeus vagy másnéven Carl von Linné (1707–1778), svéd biológus találta fel. A természetes fajtákkal és a pontatlan kategóriahatárokkal kapcsolatos problémákkal többek közt Wittgenstein, Quine, Lakoff és Schwartz foglalkozott (Wittgenstein, 1953; Quine, 1953; Lakoff, 1987; Schwartz, 1977).

Az érdeklődés a nagymérőtű ontológiák iránt növekszik. A CYC projekt (Lenat, 1995; Lenat és Guha, 1990) kapcsán egy kb. 60 000 tényt és 6000 fogalmat tartalmazó felső ontológiát hoztak nyilvánosságra, és egy sokkal nagyobb globális ontológiát szabadalmaztattak. Az IEEE létrehozta a P1600.1 albizottságot – a Standard Upper Ontology Working Groupot, az Open Mind Initiative viszont több mint 7000 internetfelhasználót kérte fel, hogy több mint 400 000 tényt vigyenek be közhasználatú fogalmakról. A weben alakulóban vannak az olyan standardok, mint az RDF, az XML és a szemantikus háló (Berners-Lee és társai, 2001), ám bár azokat még nem használják széles körben. A *Formal Ontology in Information Systems* (FOIS) konferenciákon sok érdekes cikket publikálnak minden területspecifikus, minden általános ontológiáról.

A jelen fejezetben kialakított taxonómia a szerzőktől származik, és részben azokon a tapasztalatokon alapul, melyeket a CYC projektből nyertek, részben Hwang, Schubert és Davis munkájára támaszkodva (Hwang és Schubert, 1993; Davis, 1990). Józan észreprezentációs projekt inspiráló vitái Hayes *The Naive Physics Manifesto*jában jelentek meg (Hayes, 1978; 1985b).

Az időt, a változást, a cselekvéseket és az eseményeket mind az MI-ben, mind a filozófiában és az elméleti számítástudományban intenzíven tanulmányozták. A legrégebbi megközelítés a **temporális logika** (*temporal logic*), ami egy speciálizált logika, ahol minden modell egy teljes trajektoriát ír le az (általában lineáris vagy elágazó) időben, a statikus relációs struktúrák helyett. A logika **modális operátorokat** (*modal operators*) tartalmaz, amelyeket állításokra alkalmaznak. A  $\Box p$  azt jelenti, hogy „ $p$  a jövőben minden pillanatban igaz lesz”, a  $\Diamond p$  pedig azt, hogy „ $p$  a jövőben valamikor igaz lesz”. A temporális logika tanulmányozását az ókori Görögországban Arisztotelész, valamint a Megara és a sztoikus iskola kezdeményezte. A modern időkben elsőnek Findlay vetette fel az időről való következetet „formális kalkulusának” a gondolatát (Findlay, 1941), azonban a legnagyobb hatásúnak Arthur Prior munkássága számít (Prior, 1967). Jó tankönyvek Rescher és Urquhart, valamint van Benthem munkái (Rescher és Urquhart, 1971; van Benthem, 1983).

Az elméleti számítástudomány kutatói régóta érdeklődtek az iránt, hogy számítási cselekvések szekvenciájaként értelmezett programok tulajdonságait hogyan lehetne formalizálni. A modális logikát számítógépes programokról való következtetésre Burstall vezette be (Burstall, 1974). Rövidesen utána Vaughan Pratt a **dinamikus logikát** (*dynamic logic*) dolgozta ki (Pratt, 1976), amelyben a modális operátorok programok vagy más cselekvések hatását jelzik (lásd még Harel, 1984). Így például ha dinamikus logikában  $\alpha$  egy program neve, akkor „ $\alpha \rightarrow p$ ” azt jelenti, hogy „ $p$  igaz lesz a világ minden olyan állapotában, amely az  $\alpha$  programnak a jelenlegi állapotból való indításából származik”. Az „ $\alpha \rightarrow p$ ” pedig azt jelenti, hogy „ $p$  igaz lesz a világ legalább egy olyan állapotában, amely az  $\alpha$  program mostani állapotából való indításából származik”. Programok konkrét elemzésére a dinamikus logikát Fischer és Ladner használta (Fischer és Ladner, 1977). Pnueli programokról való következtetésre klasszikus temporális logikát javasolt (Pnueli, 1977).

A temporális logikában az időt általában a nyelv modellelméletébe ágyazzák be. Az MI-ben a tendencia az volt, hogy az időpontokra és az eseményekre vonatkozó axiómákat explicit módon a tudásbázisban írjuk fel, anélkül hogy az időnek a logikában valamilyen speciális státust adnánk. Az ilyen megközelítés egyes esetekben nagyobb rugalmasságot és áttekinthetőséget biztosít. Ráadásul az elsőrendű logikában kifejezett temporális logika nagyobb eséllyel integrálható az ebben a formalizmusban felhalmozott más tudásanyaggal.

Az MI-ben az idő és a cselekvés legkorábbi kezelő apparátusa John McCarthy szituációkalkulusa volt (McCarthy, 1963). A QA3 volt az első MI-rendszer, amely nagyban felhasználta a cselekvésre vonatkozó általános rendeltetésű elsőrendű logikai következtetést (Green, 1969b). Kowalski fejlesztette ki az állítások reifikálását a szituációkalkuluson belül (Kowalski, 1979b).

A keretproblémát, mint olyant, elsőként McCarthy és Hayes ismerték fel (McCarthy és Hayes, 1969). Számos kutató a problémát megoldhatatlannak tartotta az elsőrendű logikán belül, és ez intenzív kutatáshoz vezetett a nemmonoton logikák irányába. A filozófusok Dreyfustól (Dreyfus, 1972) Crockettig (Crockett, 1994) a keretproblémát az egész MI-vállalkozás szükségszerű kudarca egyik tünetének tartották. A reprezentációs keretprobléma részleges megoldása követő állapot axiómák segítségével Ray Reiter-től származik (Reiter, 1991). A következtetési keretprobléma megoldása Holldobler és Schneeberger munkájára vezethető vissza (Holldobler és Schneeberger, 1990), amit most folyó esemény kalkulusnak nevezünk (Thielscher, 1999). Jelen fejezetben bemutatott elemzés Lin és Reiter, valamint Thielscher elemzésein alapul (Lin és Reiter, 1997; Thielscher, 1999). A szituációkalkulusban a cselekvésről való következtetés teljes, korszerű tárgyalását Shanahan és Reiter könyvei adják (Shanahan, 1997; Reiter, 2001b).

A keretprobléma részleges megoldásával újra fellángolt az érdeklődés a cselekvések-ről való következtetés deklaratív megközelítései iránt, amelyeket az 1970-es évek elejétől a speciális tervkészítő rendszerek háttérbe szorítottak (lásd 11. fejezet). A **kognitív robotika** (*cognitive robotics*) zászlaja alatt lényeges előrehaladás történt az idő és a cselekvés logikai reprezentációja terén. A GOLOG programozási nyelv a cselekvések és a tervezés kifejezésére a logikai programozás teljes kifejező erejét felhasználja (Levesque és társai, 1997a). A nyelvet a parallel cselekvések (Giacomo és társai, 2000), a sztochasztikus környezetek (Boutilier és társai, 2000) és az érzékelés (Reiter, 2001a) kezelésére is kiterjesztették.

Az eseménykalkulust Kowalski és Sergot vezették be folytonos idő kezelésre (Kowalski és Sergot, 1986). Ennek számos változata is született (Sadri és Kowalski, 1995). Shanahan ad róla egy jó és rövid áttekintést (Shanahan, 1999). Ugyanerre a célra James Allen vezette be az időintervallumokat (Allen 1983, 1984) azzal érvelve, hogy azok a szituációknál termézetesebb eszközt jelentenek a kiterjedő és a konkurrens események leírására. Peter Ladkin „konkáv” időintervallumokat (intervallumok szakadásokkal, lényegében a közönséges „konvex” időintervallumok uniói) vezetett be, és az időreprezentációhoz alkalmazta a matematikai absztrakt algebra módszertanát (Ladkin, 1986a; Ladkin, 1986b). Allen szisztematikusan vizsgálta az idő reprezentálásához ma hozzáférhető technikák széles választékát (Allen, 1991). Shoham leírta az események reifikációját, és e célból egy új sémát vezetett be (Shoham, 1987). Lényeges egybeesés lehetséges fel az e fejezetben említett eseményalapú ontológia és Donald Davidson filozófus eseményelemezése között (Davidson, 1980). Hasonló jellegű a folyadékok Pat Hayes-tól származó ontológiája is (Hayes, 1985a).

A szubsztanciák ontológiai státusának kérdése hosszú történet. Platón szerint a szubsztanciák a fizikai objektumuktól teljesen eltérő absztrakt entitások. A  $Vaj_3 \in Vaj$  helyett ö inkább  $BólVan(Vaj_3, Vaj)$ -at mondana. Ez elvezet a szubsztanciák egy olyan hierarchiájához, ahol például  $SózatlanVaj$  a  $Vaj$ -nál specifikusabb szubsztancia. A jelen fejezetben elfogadott álláspontot, miszerint a szubsztanciák objektumok kategóriái, Richard Montague vette védelmébe (Montague, 1973). Ezt az álláspontot a CYC projektben is elfogadták. Ezzel szemben egy komoly, ám nem kivédhetetlen támadást intézett Copeland (Copeland, 1993). A fejezetben említett alternatív megközelítést, ahol a vaj egy egyedi objektum, ami az univerzum összes vajszerű objektumából áll, eredetileg Leśniewski lengyel logikus javasolta (Leśniewski, 1916). **Mereológiája (mereology)** (a név a „rész” görög nevéből származik) a rész-egész relációt matematikai halmazelmélet helyettesítésére használja a célból, hogy az olyan absztrakt entitásokat, mint a halmazok, eltiönnesse. A mereológia olvashatóbb változatát Goodman és Leonard adták (Leonard és Goodman, 1940). Goodman *The Structure of Appearance* c. művében ezeket az ötleteket különböző problémák megoldására alkalmazza a tudásreprezentációk területén (Goodman, 1977). Bár a mereologikus megközelítés néhány aspektusa nehézkes – például a rész-egész reláción alapuló külön öröklődési mechanizmusra van szükség –, a megközelítést Quine is támogatta (Quine, 1960). A tudásreprezentációban való használatának kimerítő elemzését Harry Bunt adta meg (Bunt, 1985).

A mentális objektumokat és állapotokat a filozófiában és az MI-ben is intenzíven tanulmányozták. A filozófiában a **modális logika** (modal logic) a tudásról való következtetés klasszikus eszköze. Modális logika az elsőrendű logikát bizonyos modális operátorokkal, mint a *B* (hisz, *believes*) és a *K* (tud, *knows*) kiterjeszti, ezek az operátorok nem a termekre, hanem az állításokra vonatkoznak. A modális logika bizonyításelmélete a behelyettesítést a modális kontextusra korlátozza, és így referenciális átláthatatlansághoz vezet. A tudás modális logikáját Jaakko Hintikka találta fel (Hintikka, 1962). A modális logika szemantikáját Saul Kripke definiálta **lehetséges világok (possible worlds)** segítségével (Kripke, 1963). Durván fogalmazva egy világ lehetséges egy ágens részére, ha konzisztens mindenbelivel, amiről az ágensnek tudomása van. Ebből már meg lehet fogalmazni a *K* operátorra vonatkozó következtetési szabályokat. Robert C. Moore (Moore, 1980; 1985a) kapcsolatba hozza tudás modális logikáját azzal a következtetési stilussal, amely az elsőrendű logikában közvetlen módon a lehetséges világokra

hivatkozik. A modális logika ijesztően titokzatos terület lehet, azonban elosztott számítógépes rendszereknél az információról való következetést illetően jelentős alkalmazásra talált. A modális megközelítéshez részletes bevezetőt Fagin *Reasoning about Knowledge* c. könyve ad (Fagin, 1995). A kétévente megrendezett *Theoretical Aspects of Reasoning About Knowledge* (TARK) konferenciák mutatják be a tudáselmélet alkalmazását az MI-ben, a gazdasági tudományokban és az elosztott rendszerekben.

A mentális objektumok szintaktikai elméletét részletesen először Kaplan és Montague tanulmányozták, akik megmutatták, hogy az elmélet, megfelelő elővígázatosság hiányában, paradoxonokhoz vezet (Kaplan és Montague, 1960). Mivel ez a hiedelmeket egy fizikai rendszer konfigurációival kapcsolja össze, és ehhez rendelkezik egy természetes modellel egy emberi agy vagy egy számítógép formájában, ez az elmélet az MI területén az utóbbi években igen népszerű volt. Az elméletet korlátos hatékonyságú következető gépek leírására Konolige és Haas használta, Morgenstern viszont megmutatta, hogyan használható előfeltételek ábrázolására tervkészítés esetén (Konolige, 1982; Haas, 1986; Morgenstern, 1987). A megfigyelési cselekvések tervezésének a 13. fejezetben található módszerei szintaktikai elméleten alapulnak. A tudás szintaktikai és modális elméleteinek kiváló összehasonlítása található a (Davis, 1990)-ben.

Arisztotelész *Kategoriák* c. művéhez kommentárt fűzve Porfiriusz görög filozófus (kb. i. e. 234–305) rajzolt fel valamit, amit első szemantikus hálónak lehetne minősíteni. Charles S. Peirce korszerű logikára alapozva egzisztenciális gráfokat fejlesztett ki, mintegy kifejlesztve az első modern logikát használó szemantikus háló formalizmusát (Peirce, 1909). Az MI-n belül a szemantikus hálók kutatását Ross Quillian kezdeményezte (Quillian, 1961), a nyelvfeldolgozás és az emberi memória iránti érdeklődéséből indulva. Marvin Minsky nagy hatású cikkében a szemantikus hálók kereteknek (*frames*) nevezett változatát mutatta be (Minsky, 1975). A keret egy objektum vagy egy kategória reprezentációja más objektumokkal vagy kategóriákkal képzett relációival. Bár a cikk érdeklődést keltett *magának* a tudásreprezentációnak a területe iránt, mégis kritizálták, hogy a Minsky cikkét megelőző objektumorientált programozás eszközeit, illetve az öröklődés és az alapértelmezett értékek használatát újból felkínálja (Dahl és társai, 1970; Birtwistle és társai, 1973). Nem világos, hogy az objektumorientált programozásról szóló későbbi cikkeket – viszonzásképpen – mennyire befolyásolta a szemantikus hálókkal kapcsolatos korai MI-munka.

A szemantika kérdése különösen élesen merül fel Quillan szemantikus hálójával kapcsolatban (és mindenkorral kapcsolatban, akik Quillan megközelítését követték), a szemantikus hálóban mindenütt jelen levő és nagyon bizonytalan „ISA-kapcsolatok”, valamint egyéb korai tudásreprezentációs formalizmusok – mint amilyenek MERLIN (Moore és Newell, 1973) misztikus „*laps*” és „*fedi*” operációi – miatt. Az MI-kutatók figyelmét Woods híres *What's In a Link?* c. cikke irányította a precíz szemantika kialakításának szükségességére a tudásreprezentációs formalizmusban (Woods, 1975). Brachman dolgozta fel ezt a témát és a problémára megoldásokat is javasolt (Brachman, 1979). Patrick Hayes *The Logic of Frames* c. cikke még mélyebbre vágott azt állítva, hogy „a keret formalizmusának többsége nem más, mint egy új szintaktika az elsőrendű logika egy részére” (Hayes, 1979). Drew McDermott *Tarskian Semantics, or, No Notation Without Denotation!* c. cikkében amellett érvelt, hogy az elsőrendű logika modellalapú szemantikáját kell alkalmazni mindenféle tudásreprezentációs formalizmus esetében (McDermott, 1978b). Ez vitatott kérdés maradt. Megemlíteni, hogy McDermott maga is felülvizsgálta

álláspontját A *Critique of Pure Reason* (McDermott, 1987) c. cikkében. A NETL (Fahlman, 1979) egy kifinomult szemantikus háló rendszer volt, melynek ISA-kapesolatai (amit virtuális másolásnak – virtual copy –, azaz VC-kapcsolatnak hívtak) sokkal inkább a keretrendszer vagy az objektumorientált programozási nyelvek „öröklődés” jellemvonásának jelölésén alapult, mint a részhalmazkapcsolaton, és sokkal pontosabban volt definiálva, mint Quillian kapcsolatai a Woods előtti időkből. A NETL különösen érdekes, mivel párhuzamos hardveren szándékoztak megvalósítani, hogy így leküzdhessék a nagy szemantikus hálókból történő információ-visszakeresés problémáját. David Touretzky az öröklődést rigorózus matematikai analízisnek vetette alá (Touretzky, 1986). Selman és Levesque a kivételekkel kiegészített öröklődés komplexitását taglalja, meghatározva, hogy ez a legtöbb formalizmusban NP-teljes (Selman és Levesque, 1993).

A leíró logikák fejlődése csupán egy hosszú kutatási folyamat jelenlegi állása, mely az elsőrendű logika olyan használható részhalmazainak megtalálását célozza, melyekre a következetés számítástechnikai alapon kezelhető. Hector Levesque és Ron Brachman meghatározták, hogy bizonyos logikai konstrukciók, például a diszjunkció és a negálás bizonyos használata felelős elsődlegesen a logikai következetet kezelhetetlenségéért (Levesque és Brachman, 1987). A KL-ONE rendszerre (Schmolze és Lipkis, 1983) alapozva számos olyan rendszert fejlesztettek ki, melyek tervezése magában foglalta az elmeleti komplexitásanalízis eredményeit. Leginkább említhesz méltó ilyen rendszerek a KRYPTON (Brachman és társai, 1983) és a CLASSIC (Borgida és társai, 1989). Az eredmény a következetés sebességének lényeges növekedése lett, és a következetető rendszerek komplexitása és kifejezőkézsége közötti hatások sokkal jobb megismerése. A kutatások állását Calvanese foglalta össze (Calvanese és társai, 1999). Más részről, ahogy Doyle és Patil érvelnek, egy nyelv kifejezőkézségenek korlátozása vagy lehetetlenné teszi bizonyos problémák megoldását, vagy a nyelv nem logikai eszközökkel történő kitágítására ösztönöz (Doyle és Patil, 1991).

A nemmonoton következetés mind a három fő formalizmusát – a körülírást (McCarthy, 1980), az alapértelmezett logikát (Reiter, 1980) és a modális nemmonoton logikát (McDermott és Doyle, 1980) – az *AI Journal* egyetlen különszámban publikálták. A válaszhalmaz-programozást a negálás mint kudarc kiterjesztének vagy a körülírás finomításának lehet nézni. A mögötte lévő stabil modell szemantikát Gelfond és Lifschitz vezették be (Gelfond és Lifschitz, 1988). A vezető válaszhalmaz-programozási rendszerek a DVL (Eiter és társai, 1998) és SMODELS (Niemelä és társai, 2000). A diszkmeghajtó példa a SMODELS felhasználói kézikönyvből származik (Syrjänen, 2000). A válaszhalmazprogramozás használatát a tervkészítés számára Lifschitz tárgyalja (Lifschitz, 2001). A nemmonoton logika különböző megközelítéseiről jó áttekintést ad Brewka (Brewka és társai, 1997). A logikai programozás negálás mint kudarc megközelítésével és a Clark-lezárással Clark foglalkozik (Clark, 1978). Van Emden és Kowalski azt mutatták ki, hogy minden negálás nélküli Prolog programnak létezik egyértelmű minimálmodellje. Az utóbbi években tanúi lehetünk, ahogy növekszik az érdeklődés a nemmonoton logika nagyméretű tudásreprezentációs rendszerekben való alkalmazása iránt. Egy nemmonoton öröklődési rendszer első sikeres kereskedelmi alkalmazása talán a BENINQ rendszereké volt a biztosítási feltételek megtudakolásához (Morgenstern, 1998). A *Logic Programming and Nonmonotonic Reasoning* (LPNMR) konferenciák kiadványai sok különböző, logikai programozáson alapuló nemmonoton következetető rendszerről számolnak be.

Az igazság-karbantartó rendszerek tanulmányozása TMS (Doyle, 1979) és RUP rendszerekkel (McAllester, 1980) kezdődött, mindenkorral alapvetően JTMS-rendszer volt. Az ATMS megközelítést Johan de Kleer cikkeinek sorozata írta le (De Kleer, 1986c; 1986a; 1986b). A *Building Problem Solvers* (Forbus és De Kleer, 1993) részletesen elmagyarázza, hogyan használhatók a TMS-ek MI-alkalmazásokban. Nayak és Williams megmutatják, hogy egy hatékony TMS hogyan teszi lehetővé, hogy a NASA-űrhajók műveleteit valós időben megtervezhessük (Nayak és Williams, 1997).

Nyilvánvaló okoknál fogva a jelen fejezet a tudásreprezentáció minden területével elmélyülten nem foglalkozik. A három fő kihagyott témakör az alábbi:

- **Kvalitatív fizika (qualitative physics):** A tudásreprezentáció egy olyan részterülete, amely speciálisan a fizikai objektumok és folyamatok logikai nemnumerikus elméletének megkonstruálásával foglalkozik. A kvalitatív fizika elnevezést Johan de Kleer alkotta meg (de Kleer, 1975), bár állítható, hogy a kutatás Fahlman BUILD rendszerével indult meg (Fahlman, 1974). A BUILD egy komplex tervkészítő rendszer volt, bonyolult kockatornyok építéséhez. Fejlesztése során Fahlman észrevette, hogy az erőfeszítések zöme (becslése szerint 80%-a) nem magára a tervkészítésre fordítódott, hanem a különböző kocka részstruktúrák stabilitásának előtöntésére. Fahlman egy hipotetikus naiv fizikai jellegű folyamatot vázol, hogy megmagyarázza, miért képesek a kisgyerekek BUILD-szerű problémákat megoldani a BUILD-del végzett fizikai modellezésnél használt, nagy sebességű lebegőpontos aritmetika nélkül. Hayes a „történetek” – a tér-idő 4-dimenziójú, Davidson eseményeire hasonlító szeletei – felhasználásával megalkotja a folyadékok igen bonyolult naiv fizikáját (Hayes, 1985a). Hayes volt az első, aki bebizonyította, hogy ha egy csapból folyamatosan folyik a víz, miközben a lefolyót bedugaszoljuk, a kád végül túlcordul, és azt is, hogy aki beleesik a tóba, az teljesen elázik. De Kleer és Brown, valamint Ken Forbus megkísérítétek a világ egyfajta, a fizikai egyenletek kvalitatív absztrakcióján alapuló általános rendeltetésű elméletének a felállítását (De Kleer és Brown, 1985; Forbus, 1985). Az utóbbi időben a kvalitatív fizika eljutott addig a pontig, ahol már lehetséges a komplex fizikai rendszerek széles választékának az elemzése (Sacks és Joskowicz, 1993; Yip, 1991). Kvalitatív technikákat használtak teljesen új tervezésű órák, szélvédőtörök és hatlábu sétálorobotok megkonstruálásához is (Subramanian, 1993; Subramanian és Wang, 1994). A területhez jó bevezető olvasmány a *Readings in Qualitative Reasoning about Physical Systems* c. gyűjtemény (Weld és De Kleer, 1990).
- **Térbeli következtetés (spatial reasoning):** A wumpus és a bevásárló világban a navigáláshoz szükséges következtetés triviális a valódi világ gazdag térbeli struktúrájához képest. A térről történő józan ész következtetés legteljesebb kísérlete Ernest Davis (Davis, 1986; 1990) munkájában található. Cohn régiókapcsolódási kalkulusa (Cohn és társai, 1997) az alapja az egyfajta kvalitatív térbeli következtetésnek, ami elvezetett a földrajzi információs rendszerek új fajtához. A kvalitatív fizikához hasonlóan, úgy tűnik, hogy egy ágens sokáig elboldogul egy teljes metrikus reprezentáció használata nélkül. Ha egy ilyen reprezentációra van szükség, használhatók a robotikában kifejlesztett megoldások (lásd 25. fejezet).
- **Pszichológiai következtetés (psychological reasoning):** A pszichológiai következtetés magában foglalja egy működőképes pszichológia kifejlesztését a mesterséges

ágensek számára, hogy az ágensek képesek legyenek saját magukra és más ágensekre vonatkozó következtetéseket meghozni. Ennek gyakran az ún. „népi pszichológia” („folk psychology”) az alapja, amely egy olyan elmélet, amiről azt gondoljuk, hogy az emberek saját maguk és mások megítélésében használják. Amikor az MI-kutatók mesterséges ágenseiket más ágensekre vonatkozó következtetést támogató pszichológiai elméletekkel látják el, ezen elméletek alapja gyakran a logikai ágens rendszertervének a kutató által megfogalmazott leírása. A pszichológiai következtetés pillanatnyilag a természetes nyelv megértésének kontextusán belül a leghasznosabb, ahol elsődleges fontosságú a beszélő intencióját megjósolni.

Az ezen a területen folyó munkák legfrissebb forrását a *Principles of Knowledge Representation and Reasoning* nemzetközi konferenciák kiadványai jelentik. A *Readings in Knowledge Representation* (Brachman és Levesque, 1985) és a *Formal Theories of the Commonsense World* (Hobbs és Moore, 1985) a tudásreprezentáció kiváló antolói. Az előbbi a reprezentációs nyelvek és formalizmusok történelmileg fontos publikációira összpontosít, az utóbbi pedig magának a tudásnak az akkumulálására helyezi a hangsúlyt. Davis, Stefik és Sowa könyvei jó bevezető munkák a tudásreprezentációhoz (Davis, 1990; Stefik, 1995; Sowa, 1999).

## Feladatok

- 10.1.** Írja fel a wumpusvilág-beli *Lőj* cselekvés hatását leíró állításokat. Írja le a wumpusra kifejtett hatását, és ne felejtse, hogy a lövéshez az ágensnek nyílra van szüksége.
- 10.2.** A szituációkalkulon belül fogalmazzon meg egy olyan axiómát, amelyben a 0. időpillanatot az  $S_0$  szituációval kapcsolja össze, valamint egy olyan másik axiómát, ahol a  $t$  időpillanatot az  $S_0$ -ból a  $t$  számú cselekvés szekvenciájával előállított szituációval kapcsolja össze.
- 10.3.** Ebben a feladatban egy út megtervezésével foglalkozunk egy robot számára, két város között. A robot alapcselekvése a  $Megy(x, y)$ , mely révén az  $x$  városból az  $y$  városba kerül át, ha a két város között létezik közvetlen út. A  $KözvetlenÚt(x, y)$  akkor és csak akkor igaz, ha az  $x$  és az  $y$  között létezik egy közvetlen útszakasz. Feltetelezhetjük, hogy az ilyen tények a rendszer tudásbázisába már bekerültek (lásd a 102. oldalon lévő térképet). A robot Aradnál kezd, és Bukarestet kell elérnie.
  - (a) Adja meg robot kezdő állapotának egy alkalmas logikai leírását.
  - (b) Írjon fel egy alkalmas logikai kérdést, amelynek megoldásai megadják a célhoz vezető lehetséges utakat.
  - (c) Írja fel a *Megy* cselekvést leíró logikai állítást.
  - (d) Tegyük most fel, hogy két város között a közvetlen út követése a közvetlen út hosszával arányos üzemanyag-fogyasztást jelent. A robot teli tankkal indul. Bővítsre reprezentációját, hogy ezek a szempontok helyet kapjanak benne. A cselekvés leírásának olyannak kell lennie, hogy az előbb megfogalmazott kérdés megválaszolásának továbbra is a lehetséges terveket kell megadnia.

- (e) Írja le a kezdeti szituációt, és adja meg a *Megy cselekvést leíró új szabályt*, illetve szabályokat.
- (f) Tegyük most fel, hogy a csomópontok némelyike benzinkút is egyben, ahol a robot az üzemanyagtankját tehetőlheti. Bővítsen ennek megfelelően a reprezentációját, és írja fel a benzinkutak leírásához szükséges új szabályokat, a *TeleTölt cselekvést* is beleértve.
- 10.4.** Vizsgálja meg annak a lehetőségét, hogy az eseménykalkulust az *egyidejű* események kezelésével bővíthesse. Elkerülhető-e az axiómák kombinatorikus robbanása?
- 10.5.** A fejezetben kifejlesztett reprezentációt felhasználva és kiterjesztve adja meg az alábbi hat állítás reprezentációját.
- 0 °C és 100 °C között cseppekkel forrasztott víz.
  - A víz 100 °C-nál forraszt.
  - János palackjában megfagyott a víz.
  - Az ásványvíz egyfajta víz.
  - János palackjában ásványvíz van.
  - Minden folyadéknak van fagyási pontja.
  - Egy liter víz súlya több, mint egy liter alkohol súlya.
- Ismételje meg a feladatot mereológiai megközelítésen alapuló reprezentációt felhasználva, amelyben például a *Víz* egy objektum, amely a világ minden vizét, mint az objektum részét tartalmazza.
- 10.6.** Írja fel a:
- KimerítőRészFelosztás*
  - RészPartíció*
  - PáronkéntKölcsönösenKizáráó*
- definícióját a *KimerítőFelosztás*, a *Partíció* és a *KölcsönösenKizáráó* definíciójával analóg módon. Igaz-e, hogy *RészPartíció(s, Kötége(s))*? Ha igen, bizonyítsa be, ha nem, adjon meg egy ellenpéldát, és definiáljon elégséges feltételeket, amelyek mellett igaz lesz.
- 10.7.** Írjon fel egy olyan állításhalmazt, amely lehetővé teszi a különálló paradicsomok (vagy más objektumok) ármegállapítását, ha adott a kilónkénti ár. Az elmélet kiterjesztésével tegye lehetővé egy zacskó paradicsom árának a kiszámítását.
- 10.8.** Mértékeket reprezentáló alternatív sémában az egységtüggvényeket egy absztrakt hosszúság objektumra alkalmazzuk. Az ilyen sémában azt lehetne írni, hogy: *Centiméterek(Hossz(L<sub>1</sub>)) = 15*. Hogyan viszonyul ez a séma a fejezetbelihez? A szempontok többek között az átváltási axiómák, az absztrakt mennyiségek elnevezései (mint például „ezér forint”) és az eltérő egységekben kifejezett absztrakt mérések összehasonlítása (50 hüvelyk több, mint 50 cm).
- 10.9.** Konstruáljon a valuták átváltási árfolyamára egy olyan reprezentációt, amely lehetővé teszi a napi fluktuációk figyelembevételét.

- 10.10.** Ebben a feladatban az eseménykategóriák és azon időintervallumok közötti relációkkal foglalkozunk, amelyekben az események megtörténnék.
- (a) Definiálja a  $T(c, i)$  predikátumot a *Közben* és a  $\in$  segítségével.
  - (b) Adja meg a precíz magyarázatát annak, hogy a konjunktív eseménykategória leírásánál miért nincs szükség a kétféle jelölésmódra.
  - (c) Adja meg a  $T(\text{Egyik}(p, q), i)$  és a  $T(\text{VagyVagy}(p, q), i)$  formális definícióját.
  - (d) Magyarázza meg, miért értelmes a kétfajta diszjunkcióval analóg módon az események kétfajta negálásával rendelkezni. *Nem*-nek és *Soha nem*-nek nevezze el őket, és adja meg a formális definíciójukat.
- 10.11.** Definiálja a *Rögzített* predikátumot, ahol a *Rögzített(Helye(x))* azt jelenti, hogy az  $x$  objektum helye az időben rögzített.
- 10.12.** Definiálja az *Előtte*, az *Utána*, a *Közben* és az *Átlapolódik* predikátumokat a *Találkozik* predikátum és a *Kezdet* és a *Vég* függvények segítségével, de az *Idő* függvény és a  $<$  predikátum nélkül.
- 10.13.** A 10.5. alfejezetben a *Link* és a *LinkSzöveg* predikátumokat használjuk weblapok közötti kapcsolat leírására. Többek közt a *Címkeben* és a *VeddElőLapot* predikátumok felhasználásával írja fel a *Link* és a *LinkSzöveg* definíciót.
- 10.14.** A vásárlási folyamat egyik része, amivel e fejezetben nem foglalkoztunk, az egyes tételek közötti kompatibilitás ellenőrzése. Ha a kliens például egy számítógépet rendel, illeszkedik-e az a megfelelő perifériákhoz? Ha digitális kamerát rendel, megvan-e a hozzá való memóriakártya és szárazelem? Írjon fel egy olyan tudásbázist, amely eldönti, hogy a tételek adott halmaza kompatibilis-e. És ezt a tudásbázist fel lehessen használni cserékre, illetve további tételek vásárlására vonatkozó javaslatok generálására, ha mégsem lennének kompatibilisek. Bizonyosodjon meg arról, hogy a tudásbázis legalább a termékek egy körével jól működik, és könnyen terjeszthető ki másokra is.
- 10.15.** Adjon szabályokat a *Név(s, c)* predikátum definíciójának olyan esetekre történő kiterjesztéséhez, hogy az olyan füzérek, mint például a „laptop számítógép” több bolthoz tartozó idevágó kategóriának illeszkedjen. Kíséreljen meg általános érvényű definíciót alkotni. A definíciót tesztelje tfz online boltra és az általuk három kategóriára használt elnevezésekre. A laptop kategória esetén például a „Notebooks”, „Laptops”, *Notebook számítógépek*”, „*Notebook*”, „*Notebooks* és *Laptops*” és „*Notebook PC*” neveket találtuk. Az egyes elnevezéseket explicit *Név* tényekkel, mások a többes számot, a konjunkciót stb. kezelő szabályokkal lefedhetők.
- 10.16.** A vásárló vásárlásleírására vonatkozó pontatlan illeszkedés teljes megoldása nagyon nehéz, és szükségessé teszi a természetes nyelvfeldolgozási és információkinyerési technikák egész sorát (lásd 22. és 23. fejezet). Egy kis lépés megengedni a felhasználónak, hogy a különböző attribútumok minimális és maximális értékeit

határozhassa meg. Megköveteljük, hogy a vásárló a termékek leírásához az alábbi nyelvtant vegye igénybe:

<i>Leírás</i>	→ <i>Kategória [Kapcsolat Módosító]*</i>
<i>Kapcsolat</i>	→ „val” „és” „,”
<i>Módosító</i>	→ <i>Attribútum Attribútum Op Érték</i>
<i>Op</i>	→ „=” „>” „<”

Itt a *Kategória* a termék kategóriája, az *Attribútum* a termék valamelyen tulajdon-sága, mint például „CPU” vagy „ár”, és az *Érték* egy konkrét termék attribútumának az értéke. A „számítógép, min 2,5 GHz CPU, 250 000 Ft alatt” lekérdezést tehát úgy kell kifejezni, hogy „számítógép, CPU > 2,5 GHz és ár < 250 000 Ft”. Implementálja az ilyen nyelven megfogalmazott termékleírásokat elfogadó bevásárló ágenst.

- 10.17.** Az internetes bevásárlásról szóló leírásunk nélkülözte az alapvető fontosságú lépést – a termék tényleges *megvásárlását*. Adja meg a vásárlás formális logikai leírását eseménykalkulusra támaszkodva. Azaz definiálja az események olyan sorozatát, amelyek megtörténnek, ha a vásárló hitelkártyával fizet, és végül sor kerül a számlázásra és az áru leszállítására.
- 10.18.** Írja le a valamit valamiért kereskedés eseményét. A vásárlást egyfajta kereskedésként írja le, ahol a kereskedésben az egyik részt vevő termék a pénz.
- 10.19.** A két megelőző feladatban a tulajdonviszony igen egyszerű fogalmát használtuk.  A vásárló például úgy indul, hogy a forint bankjegyek már a *tulajdonában* vannak. A modell kezd szétesni, ha például a pénze bankban van, hiszen ilyenkor semmilyen konkrét bankjegymennyisége nincs, amiről ki lehetne jelenteni, hogy a tulajdonában van. A helyzetet a kölcsön, a bérlet, a haszonbérlet és a letétbe helyezés még tovább bonyolítja. Vizsgálja meg a tulajdonviszony józan ész és jogi koncepcióit, és javasoljon egy olyan sémát, amelyben ezek formálisan kifejezhetők.
- 10.20.** A feladata egy tanácsadó rendszer kifejlesztése, amely az informatika szak hallgatóinak ajánlaná, hogy milyen témaikat válasszanak a képzés feltételeinek teljesítéséhez (az intézményre jellemző feltételekkel dolgozzon). Először döntse el, hogy az összes információ reprezentálásához milyen szükséget fog használni, majd végezze el a reprezentáció építését. Megfelelő kérdésekkel érje el, hogy a rendszer egy legális képzési programot adjon vissza. A rendszernek az egyes hallgatók igényeihöz kell illeszkednie, például a fakultatív témaikat javaslatánál.

Javasoljon módszereket a rendszer tökéletesítésére, felhasználva például a hallgatói preferenciákat, a terhelésre, a jó/rossz előadókra vonatkozó információkat. minden egyes tudásfajta esetén magyarázza meg, hogy hogyan lehetne azt logikailag kifejezni. Könnyen be tudná-e a rendszere fogadni ezt az információt úgy, hogy egy hallgató részére a *legjobb* képzési programot ajánlja fel?

**10.21.** A 10.1. ábra a dolgok hierarchiájának felső szintjeit mutatja. Terjessze ki az ábrát úgy, hogy a lehető legtöbb valódi kategóriát tartalmazzon. Erre jó módszer, ha a minden nap életével kapcsolatos minden dolgot leír. Ez objektumokat és eseményeket fog magában foglalni. Kezdje a reggeli felkeléssel, rendszeresen haladjon tovább, megfigyelve, hogy mit érint, mit csinál, mire gondol. Egy véletlen mintavétel eredménye lehetne például a következő: zene, hírek, tej, séta, kocsivezetés, benzin, Skála, szönyeg, beszélgetés, Kovács tanár úr, csirkepörkölt, nyelv, 150 Ft, napi újságok stb.

El kellene készíteni egy egységes hierarchiafát (egy nagy papírlapon) és egy listát az objektumokról és a kategóriákról, az egyes kategóriaegyedek által kielégített egy vagy több relációval együtt. Mindegyik objektumnak egy kategóriához kell tartoznia, és minden kategóriának a hierarchia részének kell lennie.

**10.22.** (Doug Lenat egy példájának adaptációja.) A feladat az, hogy elegendő, logikai formában kifejezett kapcsolódó tudást gyűjtsön össze, hogy az alábbi egyszerű mondatra vonatkozó kérdéssort meg tudja válaszolni:

Tegnap János a Skálába ment, és vett két kiló paradicsomot és fél kiló darált húst.

Kezdje azzal, hogy a mondat tartalmát egy sor kijelentéssel reprezentálja. Áttekinthető logikai struktúrájú állításokat kell írnia (például állítások arról, hogy az objektumoknak bizonyos tulajdonságai vannak, hogy az objektumok bizonyos relációban vannak egymással, és hogy egy adott tulajdonságot teljesítő minden objektum egy másik tulajdonsággal is rendelkezik). Az alábbiak feltehetően segítségére lesznek:

- Milyen osztályokra, egyedekre, relációkra stb. lesz szüksége? Mik ezeknek a szuper- és alosztályai? (Többek között eseményekre és időrendi rendezésre is szüksége lesz.)
- Milyen módon lehetnének ezek a részei egy általánosabb hierarchiának?
- Milyenek a közöttük lévő kényszerek és relációk?
- Mennyire részletesen kell leírni az egyes fogalmakat?

A megkonstruálandó tudásbázisnak képesnek kell lennie a továbbiakban közölt kérdéslistát megválaszolni. A kérdések közül néhány közvetlenül magával a történettel foglalkozik, de a többségük a háttértudást – a sorok közötti olvasást – igényli. Foglalkozni kell azzal, hogy milyen dolgokat lehet egy bevásárlóközpontban vásárolni, hogy a kiválasztott tételek vásárlásánál mi történik, hogy mi a vásárolt tételek rendeltetése stb. Kísérelje meg a reprezentációját a lehető a legáltalánosabba kialakítani. Egy triviális példával szemléltve, ne azt állítsa, hogy „Az emberek a Skálában ennivalót vásárolnak”, mert ez nem segít azok kezelésében, akik máshol szoktak vásárolni. Ne azt állítsa, hogy „János darált hússal és paradicsommal spagetti készített”, mert ez semmilyen más dolog kezelésében nem segít. Ne rejtsen válaszokat a kérdésekbe, a (c) kérdés például azt kérdezi, hogy „Vásárolt János húst?”, és nem azt, hogy „Vásárolt-e János fél kiló darált húst?“.

Adja meg a kérdéseket megválaszoló következetes sémáját. Ennek során minden bizonnyal új fogalmakat is kell kreálnia és új állításokat kell megfogal-

maznia stb. Ha lehetséges, használjon egy logikai következtető rendszert, hogy kimutassa a tudásbázis elégességét. Sok dolog, amit leír, a valóságban csak közelítőleg lesz igaz. Ne törödjön ezzel túl sokat, a cél azt a józan ész tudást ki-nyerni, ami a kérdések megválaszolását egyáltalán lehetséges teszi. E kérdés-kör igazán teljes megválaszolása *rendkívül* nehéz, valószínűleg meg is haladja a mai tudásreprezentációk lehetőségeit.

- (a) János gyerek vagy felnőtt? [Felnőtt]
- (b) Van Jánosnak most legalább 2 db paradicsoma? [Igen]
- (c) Vásárolt János húst? [Igen]
- (d) Látta-e János Marit, ha Mari ugyanabban az időben vásárolt paradicsomot? [Igen]
- (e) A paradicsomot a bevásárlóközpontban készítik-e? [Nem]
- (f) Mit fog csinálni János a paradicsommal? [Megeszi]
- (g) Kapható-e a Skálában dezodor? [Igen]
- (h) Hozott-e János magával elég pénzt a bevásárlóközpontba? [Igen]
- (i) A vásárlást követően kevesebb pénze van-e Jánosnak? [Igen]

**10.23.** Az előbbi példa tudásbázisához készítsen szükséges bővítéseket/módosításokat úgy, hogy megválaszolhatók legyenek az alábbi kérdések. Mutassa ki, hogy a tudásbázis tényleg képes ezeket megválaszolni, és a jelentésében számoljon be a javításokról, miért volt a javításokra szükség, és arról is, hogy a javítások lényegtelenek, illetve lényegesek voltak.

- (a) Vannak-e a Skálában más emberek, amikor János ott vásárol? [Igen – a személyzet!]
- (b) Vegetáriánus-e János? [Nem]
- (c) A Skálában ki a dezodor tulajdonosa? [Skála Rt.]
- (d) Kapott-e János 1 kg darált húst? [Igen]
- (e) A szomszédos MOL benzinkútán van-e benzin? [Igen]
- (f) Belefér-e a paradicsom János kocsijába? [Igen]

**10.24.** Emlékezzen arra, hogy a szemantikus háló öröklődési információját logikailag alkalmas implikációs állításokkal ki lehet fejezni. Ebben a feladatban megfontoljuk az ilyen állításokra épülő öröklődési következtetés hatékonyságát.

- (a) Vegyük egy használtautó-katalógus információtartalmát – hogy az 1999-es Mitsubishi Carisma értéke 3 millió Ft. Tegyük fel, hogy az összes információ (a több tízezer modell részére) a fejezetben ajánlott logikai szabályok formájában van megfogalmazva. Írjon fel három ilyen szabályt, az egyik az 1999-es Carismáról szóljon. Hogyan használná a szabályokat, hogy egy konkrét gépkocsi értékét megtalálhassa (például XY, ami egy 1999-es Carisma), ha adott egy hátrafelé láncolt tételelbizonyító, mint amilyen például a Prolog?
- (b) Hasonlítsa össze az ennek a problémának a megoldására alkalmazott hátrafelé láncolt következtetési módszer és a szemantikus hálóban használt öröklődési séma hatékonyságát.
- (c) Magyarázza meg, hogy az előrefelé láncolt következtetés hogyan teszi lehetővé egy logikai rendszer számára ugyanannak a problémának egy haté-

kony megoldását, feltéve, hogy a tudásbázis csak 10 000 szabályt tartalmaz az árakról.

- (d) Írjon le egy olyan helyzetet, ahol sem az előrefelé, sem a hátrafelé láncolt következtetés nem teszi lehetővé a konkrét gépkocsira vonatkozó árlekérdezés hatékony megvalósítását.
- (e) Tud olyan megoldást ajánlani, ami lehetővé teszi, hogy ezt a fajta lekérdezést a logikai rendszerek mindegyikében hatékonyan tudjuk kezelní? (*Sugározás: emlékezzen arra, hogy az ugyanahhoz a kategóriához tartozó két gépkocsinak ugyanaz az ára.*)

**10.25. Feltételezhető, hogy egy szemantikus hálóban a sima és az egy vonallal bekötött kapcsolat közötti szintaktikai különbségtétel felesleges, mert az utóbbi kapcsolatok csak a kategóriákhoz tartoznak. Az öröklődési algoritmusnak elég lenne feltételeznie, hogy a kategóriához csatolt sima kapcsolat szándékoltan a kategória minden tagjára vonatkozik. Mutassa meg a lehetséges hibák megadásával, hogy az érv hamis.**

# **IV. RÉSZ**

# **TERVKÉSZÍTÉS**

---

# 11. TERVKÉSZÍTÉS

*Ebben a fejezetben láthatjuk, hogy hogyan használhatja ki egy ágens a probléma szerkezetét egy összetett cselekvési terv kialakításához.*

Egy cél elérésére irányuló cselekvéssorozat kialakítását **tervkészítésnek** (*planning*) nevezzük. Tervkészítő ágensekre eddig két példát látunk: a 3. fejezet keresésalapú problémamegoldó ágensét és a 10. fejezetben bemutatott logikai tervkészítő ágenst. Ez a fejezet elsősorban az olyan nagyobb bonyolultságú tervezési feladatokra való felskálázással foglalkozik, melyek meghaladják az eddig ismertetett megközelítések képességeit.

A 11.1. fejezet egy kifejező, de mégis megfelelően korlátozott nyelvet mutat be az állapotokat és cselekvéset tartalmazó tervkészítési feladatok leírására. Ez a nyelv szoros kapcsolatban áll a 7. és 10. fejezetben szereplő ítéletlogikai, illetve elsőrendű leírásokkal. A 11.2. fejezet megmutatja, hogy az előre- és a hátrafelé kereső algoritmusok hogyan tudják előnyösen kihasználni ezt a reprezentációt, elsősorban a leírás szerkezetéből automatikusan levezethető heurisztikák segítségével. (Ez analóg az 5. fejezetben bemutatott, a kényszerkielégítési problémához kialakított heurisztika elkészítésével.) A 11.3–11.5. alfejezetben olyan tervkészítő algoritmusok kerülnek bemutatásra, amelyek felhasználva a probléma reprezentációját, képességeikben túlmutatnak az előre-, illetve hátrafelé keresésen. Nevezetesen olyan eljárásokat vizsgálunk meg, amelyek nem csak teljesen rendezett cselekvések sorozatát képesek figyelembe venni.

Ebben a fejezetben csak olyan környezetekre szorítkozunk, melyek teljesen megfigyelhetők, determinisztikusak, végesek, statikusak (azaz ahol változások csak akkor történnek, amikor az ágens cselekszik), diszkrétek (időben, cselekvésekben, objektumokban és hatásokban). Ezeket **klasszikus tervkészítési** (*classical planning*) környezeteknek nevezzük. Ezzel ellentétben a nem klasszikus tervkészítés a részben megfigyelhető vagy a sztochasztikus környezetek kezelésére szolgál, és itt megjelennek eltérő, a 12. és 17. fejezetben bemutatott, algoritmusok és ágensterek.

## 11.1. A TERVKÉSZÍTÉSI PROBLÉMA

Fontoljuk meg, hogy mi történne, ha egy közönséges, a standard keresési algoritmusokat – mélységi keresés, A\* stb.– használó problémamegoldó ágens valós nagyméretű problémákkal kerülne szembe. Ez segíthet abban, hogy jobb tervkészítő ágenseket tervezzünk.

A legkézenfekvőbb nehézség, hogy a problémamegoldó ágenst lebéníthatnák a szükségtelen cselekvések. Vegyük például a *Mesterséges intelligencia modern megközelítésben* c. könyv internetes megvásárlásának a feladatát. Tegyük fel, hogy minden egyes

10 jegyű ISBN szám megvásárlása egy-egy cselekvés, ez összesen 10 milliárd cselekvést jelent. A keresési algoritmusnak minden 10 milliárd cselekvés végállapotát meg kellene vizsgálnia, hogy megtalálja a célnak megfelelőt, nevezetesen, hogy birtokoljuk a 9635454112 ISBN számú könyvet. Másrészről egy értelmes tervkészítő ágensnek képesnek kell lenni arra, hogy a pontos cél *Birtokol*(ISBN9635454112) leírásából viszszafelé dolgozva közvetlenül eljusson a *Vásárol*(ISBN9635454112) cselekvéshez. Hogy ezt megtehesse, az ágensnek arra az általános tudásra van szüksége, hogy a *Vásárol*( $x$ ) következménye a *Birtokol*( $x$ ). Ha adott ez a tudás, a tervező egy egyszerűsítés lépében el tudja dönteni, hogy a *Vásárol*(ISBN9635454112) a helyes cselekvés.

A következő nehézség, egy jó **heurisztika** (heuristic function) meghatározása. Tegyük fel, hogy az ágens feladata négy különböző könyv megvásárlása. Ebből  $10^{40}$  négyelépeses terv adódik, azaz nem kérdéses, hogy egy megfelelő heurisztika nélküli keresés értelmetlen. Az ember számára egy állapot költségének becslésére kézenfekvő heurisztika a továbbiakban még megvásárolandó könyvek száma. Sajnos ez nem nyilvánvaló egy problémamegoldó ágens számára, hisz az a céltesztelet egy fekete dobozként látja, mely minden állapotra mindenkorral egy igaz-hamis értéket ad vissza. Ennek következményeképp a problémamegoldó ágens nem autonóm; azaz emberi beavatkozást igényel minden új problémánál az alkalmazható heurisztika megadására. Másrésztől, ha az ágensnek rendelkezésére áll a cél leírása, mint részcélok konjunkciója, akkor használhat egy egyszerű feladatfüggetlen heurisztikát: a még nem teljesített részcélok számát. A könyvvásárlási feladat esetén a cél  $Birtokol(A) \wedge Birtokol(B) \wedge Birtokol(C) \wedge Birtokol(D)$  lenne, és a  $Birtokol(A) \wedge Birtokol(C)$  állapot költsége 2. Így az ágens számára automatikusan elérhető a helyes heurisztika erre és más problémákra is. A fejezet későbbi részében látni fogjuk, hogy hogyan lehet létrehozni olyan kifinomultabb heuristikákat, ami a célstruktúráián túl számba veszi a végrehajtható cselekvéseket is.

Végül a problémamegoldó ágens nem hatékony, mert nem tudja kihasználni a **problémadekompozíció** (**problem decomposition**) lehetőségét. Vagyük például a következő feladatot: több csomagot kell kiszállítanunk a megfelelő címekre, melyek Ausztrália különböző pontjain találhatók. Jó megközelítés, ha megkeressük a célpontokhoz legközelebb eső reptereket, és felosztjuk a teljes problémát több részfeladatra; repterenként egyre. Az egy reptéren keresztül irányított csomagok esetén a további dekompozíció lehetősége a célvárostól függ. Az 5. fejezetben láttuk, hogy egy ilyen felbontás képessége hozzájárul a kényszerkielégítési feladatmegoldók hatékonyságához. A tervkészítőkre ugyanez igaz: a legrosszabb esetben  $n$  csomag legjobb kiszállítási tervének elkészítése  $O(n!)$ , míg ha a feladat  $k$  egyenlő részre bontható ez mindenossze  $O((n/k)! \times k)$  komplexitású feladat.

Ahogy az 5. fejezetben megjegyeztük, a teljesen dekomponálható problémák jók, de ritkák.<sup>1</sup> A legtöbb tervkészítő rendszer felépítése – különösen a 11.3. fejezetben bemutatásra kerülő részben rendezett tervkészítő felépítése – azon a feltételezésen alapul, mely szerint a legtöbb valós, életszerű probléma majdnem dekomponálható (*nearly decomposable*). Ez annyit jelent, hogy a tervkészítő dolgozhat független részcélokon,

<sup>1</sup> Vegyük észre, hogy még a csomagkiszállítási feladat sem teljesen dekomponálható. Vannak esetek, amikor jobb a csomagokat mégis egy távolabbi reptére irányítani, ha ezzel megspórolhatunk egy külön repülőjáratot a közelére. Mindemellett a legtöbb szállítócég inkább a már bejárattott dekomponált megoldásokhoz ragaszkodik, hogy csökkentse a számítási és szervezési nehézségeket.

de a résztervezek összekombinálása további feladatokat eredményezhet. Néhány feladat esetén ez a feltételezés nem helytálló, mert az egyik részcél kidolgozása gyakran meg-bontja a másik részcélt. Ezek a részcélok közötti kölcsönhatások azok, amelyek a fejtörőket (mint a nyolcas kirakó) valójában fejtörővé teszik.

## A tervkészítési problémák nyelve

A fent leírtak alapján a tervkészítési problémák reprezentációjának – ami az állapotokat, cselekvéseket és célokat jelenti – lehetőséget kellene biztosítania a tervkészítő algoritmus számára, hogy a feladat logikai struktúráját kihasználhassa. Ennek kulcsa, hogy olyan nyelvet találunk, ami kellően kifejező ahhoz, hogy a problémák egy széles körét leírja, ugyanakkor kellően szigorú ahhoz, hogy a leírásokon hatékony algoritmusok működheszenek. Ebben a fejezetben először körülönbelül a klasszikus tervkészítők által használt alapnyelvet, ami STRIPS néven ismert.<sup>2</sup> Később rámutatunk a számos módosítási lehetőségből néhányra a STRIPS-szerű nyelvekben.

**A állapotok leírása.** A tervkészítők a világot logikai feltételekre dekomponálják, és az állapotokat a pozitív literálok konjunkciójaként írják le. Tekintsük az ítéletlogikai literálokat; például a *Szegény  $\wedge$  Ismeretlen* reprezentálhatja egy szerencsétlen ágens állapotát. Elsőrendű literálokat is felhasználunk; például *Ott(Repülő<sub>1</sub>, Melbourne)  $\wedge$  Ott(Repülő<sub>2</sub>, Sydney)* a csomagszállítási feladat egy állapotát írhatja le. Az elsőrendű logikai állapotleírások literáljainak **alap-** és **függvénymentes literálnak** (*ground and function-free*) kell lenniük. Az olyan literálok, mint az *Ott(x, y)* vagy az *Ott(Apja(Ferenc), Sydney)* nem megengedettek. A **zárt világ feltételezést** (*closed-world assumption*) használjuk, ami annyit tesz, hogy a nem felsorolt állításokat hamisnak vesszük.

**A célok leírása.** A cél egy részlegesen definiált állapot, melyet pozitív alapliterálok konjunkciója reprezentál, mint *Gazdag  $\wedge$  Híres* vagy *Ott(P<sub>2</sub>, Tahiti)*. Az s ítéletlogikai állapot **kielégíti a c célt** (*goal satisfaction*), ha s tartalmazza a c-ben szereplő összes atomot (és esetleg még továbbiakat). Például a *Gazdag  $\wedge$  Híres  $\wedge$  Szomorú* állapot kielégíti a *Gazdag  $\wedge$  Híres* célt.

**A cselekvések leírása.** A cselekvést a következő két állapot határozza meg: az elő-feltétel, aminek teljesülni kell az akció végrehajtásához, és a következmény, ami a végrehajtás eredményeként lép fel. Például két állomás közötti repülés leírása az alábbi:

*Cselekvés(Repül(p, honnan, hova),*

**ELŐFELTÉTEL:** *Ott(p, honnan)  $\wedge$  Repülő(p)  $\wedge$  Repülőtér(honnan)  $\wedge$  Repülőtér(hova)*

**KÖVETKEZMÉNY:**  *$\neg$ Ott(p, honnan)  $\wedge$  Ott(p, hova))*

Ezt pontosabban **cselekvési sémának** (*action schema*) nevezzük, ami azt takarja, hogy ez számos különböző cselekvést reprezentál, ami a p, honnan és a hova változók különböző behelyettesítéseivel származtatható. Általánosságban a cselekvési séma három fő részből áll:

- A cselekvés megnevezése és paraméterlistája – például a *Repül(p, honnan, hova)* – a cselekvés azonosítására szolgál.

<sup>2</sup> A STRIPS a STanford Research Institute Problem Solver rövidítése.

- **Az előfeltétel (precondition)**, függvényektől mentes pozitív literálok konjunkciója, azt mutatva, hogy milyen feltételeknek kell előzetesen teljesülni a cselekvés végrehajtásához.
- **A következmény (effect)**, függvényektől mentes literálok konjunkciója, ami leírja, hogy az állapot hogyan változik, amikor a cselekvés végrehajtásra kerül. A cselekmény eredményeképp adódó következményrészben szereplő  $P$  pozitív literál igaz értéket kap, míg a  $\neg P$  hamis értéket vesz fel. A következményrészben szereplő változóknak a cselekvés előfeltételei között is szerepelni kell.

Az olvashatóság javítása érdekében néhány tervkészítő következményrész szétválasztja egy hozzáadás listára (**add list**) a pozitív és egy törlés listára (**delete list**) a negatív literáloknak.

Most hogy a tervkészítők reprezentációjának szintaxisát definiáltuk, adjuk meg a szemantikát is. Ennek legegyszerűbb módja, ha leírjuk, hogy a cselekvések hogyan módosítják az állapotot. (Egy másik lehetséges módszer, hogy egy direkt fordítást specifikálunk a következő állapot axiómákra, melyek szemantikája az elsőrendű logikából származik. Lásd 11.3. feladat.) Először is azt mondjuk, hogy egy cselekvés **alkalmazható (applicable)** minden állapotban, ami kielégíti az előfeltételeket; egyébként a cselekvés hatástalan. Egy elsőrendű séma esetében az alkalmazhatóság elérése az előfeltételek egy  $\theta$  behelyettesítését vonja maga után. Tegyük fel például, hogy a jelen állapot leírása:

$$\begin{aligned} & Ott(P_1, JFK) \wedge Ott(P_2, SFO) \wedge Repülő(P_1) \wedge Repülő(P_2) \\ & \wedge Repülőtér(JFK) \wedge Repülőtér(SFO) \end{aligned}$$

Ez teljesíti az

$$Ott(p, honnan) \wedge Repülő(p) \wedge Repülőtér(honnan) \wedge Repülőtér(hova)$$

előfeltételeit a  $\{p/P_1, honnan/JFK, hova/SFO\}$  behelyettesítésekkel (és másokkal is – lásd 11.2. feladat). Így a konkrét  $Repül(P_1, JFK, SFO)$  cselekvés alkalmazható.

Az s állapotból kiindulva az alkalmazható a cselekvés végrehajtásának *eredménye* az  $s'$  állapot, ami azonos  $s$ -sel, kivéve, hogy az a cselekvés következményrészében szereplő pozitív  $P$  literálokat az  $s'$ -höz adjuk, míg bármilyen  $\neg P$  negatív literált eltávolítjuk  $s'$ -ből. Így a  $Repül(P_1, JFK, SFO)$  cselekvés után az állapot a következő:

$$\begin{aligned} & Ott(P_1, SFO) \wedge Ott(P_2, SFO) \wedge Repülő(P_1) \wedge Repülő(P_2) \\ & \wedge Repülőtér(JFK) \wedge Repülőtér(SFO) \end{aligned}$$

Vegyük észre, hogy a már szereplő pozitív következményeket nem szúrjuk be még egyszer, illetve ha egy negatív literál nem szerepel az állapotleírásban, akkor a következmény ezen része figyelmen kívül hagyható. Ez a definíció testesíti meg az úgynevezett STRIPS feltételezést (**assumption**): minden a következményben nem szereplő literál változatlan marad. Így a STRIPS elkerüli a 10. fejezetben bemutatott **reprezentációs keret problémát (representational frame problem)**.

Végezetül definiálhatjuk a tervkészítési probléma **megoldását (solution)**. Legegyszerűbb formájában ez csak egy cselekvéssorozat, melyet a kiindulási állapotból végrehajtva a célállapotot eredményezi. A fejezet további részeiben a megoldások cselekvések részben rendezett sorozatai is lehetnek, amennyiben minden cselekvéssorozat, ami megfelel ennek a részben rendezésnek, megoldás.

## Kifejezőképesség és kiterjesztések

A sokféle megkötés, korlátozás amit a STRIPS nyelv tartalmaz, abban a reményben került beépítésre, hogy a tervkészítő algoritmusok egyszerűbbek és hatékonyabbak lehessenek, anélkül hogy a valós problémák leírását megnehezítenék. Egyike a legfontosabb megkötéseknek, hogy a literáloknak függvénymenteseknek kell lenniük. Ezzel a megkötéssel biztosíthatjuk, hogy egy adott problémához tartozó bármely akció séma ítéletkalkulus formára, azaz változómentes ítéletlogikai cselekvés reprezentációk véges halmazára hozható. (A téma bővebb leírását lásd a 9. fejezetben.) Például a légi szállítási problémakörben 10 repülő és 5 repülőtér esetén a  $Repül(p, honnan, hova)$  séma  $10 \times 5 \times 5 = 250$  ítéletlogikai cselekvésre fordítható. A 11.4. és 11.5. alfejezet tervkészítői közvetlenül az ítéletkalkulusra hozott leírással dolgoznak. Ha függvényszimbólumokat is megengedünk, akkor végtelen sok állapot és cselekvés határozható meg.

STRIPS nyelv	ADL nyelv
Csak pozitív literálok az állapotokban: <i>Szegény</i> $\wedge$ <i>Ismertetlen</i>	Pozitív és negatív literálok az állapotokban: $\neg$ <i>Gazdag</i> $\wedge$ $\neg$ <i>Hires</i>
Zárt világ feltételezés: A nem szereplő literálok harmisak.	Nyílt világ feltételezés: A nem szereplő literálok meghatározatlanok.
A $P \wedge \neg Q$ jelentése: hozzáadjuk $P$ -t, és töröljük $Q$ -t.	A $P \wedge \neg Q$ jelentése: hozzáadjuk $P$ -t és $\neg Q$ -t, valamint töröljük $\neg P$ -t és $Q$ -t.
A célokban csak alapliterálok szerepelnek: <i>Gazdag</i> $\wedge$ <i>Hires</i>	Kvantifikált változók a célokban: $\exists x$ <i>Ott</i> ( $P_1, x$ ) $\wedge$ <i>Ott</i> ( $P_2, x$ ) cél azt adja meg, hogy $P_1$ és $P_2$ azonos helyen van.
A célok konjunkciók: <i>Gazdag</i> $\wedge$ <i>Hires</i>	A célok konjunkciókat és diszjunkciókat is tartalmazhatnak: $\neg$ <i>Szegény</i> $\wedge$ ( <i>Hires</i> $\vee$ <i>Okos</i> )
A következmények konjunkciók.	Feltételes következmények is megengedettek: amikor $P$ : $E$ jelentése, hogy $E$ akkor érvényes, ha $P$ teljesül.
Nincs támogatás az egyenlőségvizsgálathoz.	Béépített egyenlőség ( $x = y$ ) predikátum.
Nincs támogatás típusokhoz.	A változóknak lehet típusa, mint ( $p : Repülő$ ).

**11.1. ábra.** A Strips és az ADL nyelv összehasonlítása a tervkészítési feladatok reprezentációjának szempontjából. Mindkét esetben a célok úgy viselkednek, mint egy paraméterek nélküli cselekvés előfeltételei

Napjainkra, nyilvánvalóvá vált, hogy a STRIPS nem elégé kifejező néhány valós problémakörhöz. Ennek eredményeképpen számos nyelvváltozatot dolgoztak ki. A 11.1. ábra röviden összefoglalja az egyik legfontosabbat, a cselekvésleíró nyelvet (Action Description Language – ADL) úgy, hogy összehasonlíta azt a STRIPS alapverziójával. ADL nyelven a *Repülés* leírása az alábbi:

*Cselekvés*( $Repül(p : Repülő, honnan : Repülőtér, hova : Repülőtér)$ ),

ELŐFELTÉTEL:  $Ott(p, honnan) \wedge (honnan \neq hova)$

KÖVETKEZMÉNY:  $\neg Ott(p, honnan) \wedge Ott(p, hova))$

A  $p : Repülő$  írásmód az előfeltételek a paraméterlistájában a  $Repülő(p)$  egy rövidítése, ami nem növeli a kifejezőképességet, de javítja az olvashatóságot. (Mindemellett redukálja a létrehozható ítéletlogikai cselekvések számát.) A ( $honnán \neq hova$ ) előfeltétel azt a tényt fejezi ki, hogy egy repülőút kiindulási és célállomása nem lehet azonos. Ezt a STRIPS nyelvben nem lehetne tömörén kifejezni.

A mesterséges intelligenciában használt változatos tervkészítő formalizmusokat egy szabványos szintaxisba rendszerezik, amit **tervkészítési terület definíciós nyelvnek** (**Planning Domain Definition Language – PDDL**) neveznek. Ez a nyelv lehetővé teszi a kutatók számára, hogy benchmark problémákat cseréljenek ki egymás között, és összevessék az eredményeket. A PDDL résznyelveket tartalmaz az ADL és a 12. fejezetben bemutatásra kerülő hierarchikus feladathálózatok számára.

A STRIPS és az ADL jelölésrendszer számos valós problémakörre megfelelő. A következő alfejezetek néhány egyszerű példát mutatnak be. Néhány számottevő megkötés azért még megmaradt. A legnyilvánvalóbb, hogy közvetlenül nem tartalmazzák a cselekvések véghatásait (**ramifications**). Például ha vannak emberek, csomagok vagy porcikák egy repülőn, akkor mindenkor helyet változtatnak egy repülés során. Ezeket a változásokat leírhatjuk, mint a repülés egyenes következményeit, így természetesennek tűnik a repülőgép tartalmának helyét, mint a gép helyének logikai következményét ábrázolni. Az ilyen állapotmegkötésekre (**state constraints**) a 11.5. alfejezet tartalmaz példákat. A hagyományos tervkészítő rendszerek meg sem próbálják megoldani a **kvalifikációs problémát** (**qualification problem**), azaz a nem reprezentált körülmények problémáját, melyek a cselekvés meghiúsulását okozhatják. A 12. fejezetben látni fogjuk, hogy a kvalifikációs probléma hogyan közelíthető meg.

## Példa: Légi te herszállítás

A 11.2. ábra egy te herszállítási problémát mutat be, ami a teher be-, illetve kirakodását és állomások közötti légi szállítását tartalmazza. A probléma három cselekvéssel írható le: *Berakodás*, *Kirakodás* és *Repülés*. A cselekvések két predikátumot érintenek: a *Benne(c, p)* jelentése, hogy a *c* teher a *p* repülőgépen van, és az *Ott(x, a)* jelentése,

*Kiindulás*( $Ott(C_1, SFO) \wedge Ott(C_2, JFK) \wedge Ott(P_1, SFO) \wedge Ott(P_2, JFK)$   
 $\wedge Teher(C_1) \wedge Teher(C_2) \wedge Repülő(P_1) \wedge Repülő(P_2)$   
 $\wedge Repülőtér(JFK) \wedge Repülőtér(SFO))$

*Cél*( $Ott(C_1, JFK) \wedge Ott(C_2, SFO)$ )

*Cselekvés*(*Berakodás*(*c*, *p*, *a*),  
 ELŐFELTÉTEL:  $Ott(c, a) \wedge Ott(p, a) \wedge Teher(c) \wedge Repülő(p) \wedge Repülőtér(a)$   
 KÖVETKEZMÉNY:  $\neg Ott(c, a) \wedge Benne(c, p)$ )  
*Cselekvés*(*Kirakodás*(*c*, *p*, *a*),  
 ELŐFELTÉTEL:  $Benne(c, p) \wedge Ott(p, a) \wedge Teher(c) \wedge Repülő(p) \wedge Repülőtér(a)$   
 KÖVETKEZMÉNY:  $Ott(c, a) \wedge \neg Benne(c, p)$ )  
*Cselekvés*(*Repülést*(*p*, *honnán*, *hova*),  
 ELŐFELTÉTEL:  $Ott(p, honnan) \wedge Repülő(p) \wedge Repülőtér(honnán) \wedge Repülőtér(hova)$   
 KÖVETKEZMÉNY:  $\neg Ott(p, honnan) \wedge Ott(p, hova)$ )

11.2. ábra. A STRIPS-probléma repülőterek közötti légi te herszállítási feladathoz

hogy az  $x$  objektum (teher vagy repülőgép) az  $a$  repülőtéren található. Vegyük észre, hogy a teher nincs *Ott* sehol, ha *Benne* van egy repülőgépben, azaz az *Ott* valójában azt jelenti, hogy az objektum „elérhető a megadott helyen”. Tapasztalattal kell rendelkezni a cselekvésdefiníciók területén, ahhoz hogy az ilyen részleteket konzisztensen ábrázoljuk. A következő terv megoldása a feladatnak:

[*Berakodás(C<sub>1</sub>, P<sub>1</sub>, SFO)*, *Repülés(P<sub>1</sub>, SFO, JFK)*, *Kirakodás(C<sub>1</sub>, P<sub>1</sub>, JFK)*  
*Berakodás(C<sub>2</sub>, P<sub>2</sub>, JFK)*, *Repülés(P<sub>2</sub>, JFK, SFO)*, *Kirakodás(C<sub>2</sub>, P<sub>2</sub>, SFO)*]

A mi reprezentációnk tisztán STRIPS nyelvű. Nevezetesen ez engedélyezi, hogy egy repülő kiinduló és célállomása azonos repülőtér legyen. Az ADL egyenlőtlenség operátorai ezt kizárhathatnák.

## Példa: A pótkerék probléma

Vegyük a kerékcseré problémáját. Pontosabban a célunk az, hogy egy jó pótkerék legyen felszerelve az autó tengelyére, ahol a kiinduló állapotban egy lapos kerék van felszerelve, míg a jó pótkerék a csomagtartóban található. Az egyszerűség kedvéért a mi feladatunk elég absztrakt, azaz nincsenek beragadt csavarok vagy egyéb más nehézségek. Csak négy cselekvés van: a pótkerék kivétele a csomagtartóból, a lapos kerék eltávolítása a tengelyről, a pótkerék felszerelése és az autó magára hagyása reggelig. Feltelezzük, hogy az autó egy igen rossz környéken áll, ahol az autó magára hagyása azt eredményezi, hogy reggelre eltúnnék a kerekek.

A 11.3. ábra a feladat ADL leírását tartalmazza. Vegyük észre, hogy ez a megadás pusztán ítéletlogikai. A STRIPS nyelven túlmutat, hogy a *TeddFel(Pótkerék, Tengely)* cselekvéshez a  $\neg$ *Ott(Pótkerék, Tengely)* negált előfeltétel tartozik. Ezt elkerülhetnék egy *SzabaddáTesz(Tengely)* használatával, amint azt a következő példában látni fogjuk.

$\text{Kiindulás}(\text{Ott}(\text{LaposKerék}, \text{Tengely}) \wedge \text{Ott}(\text{Pótkerék}, \text{Csomagtartó}))$ $\text{Cél}(\text{Ott}(\text{Pótkerék}, \text{Tengely}))$ <i>Cselekvés(Eltávolít(Pótkerék, Csomagtartó),</i> <i>Előfeltétel: Ott(Pótkerék, Csomagtartó)</i> <i>Következmény: <math>\neg</math>Ott(Pótkerék, Csomagtartó) <math>\wedge</math> Ott(Pótkerék, Föld))</i> <i>Cselekvés(Eltávolít(LaposKerék, Tengely),</i> <i>Előfeltétel: Ott(LaposKerék, Tengely)</i> <i>Következmény: <math>\neg</math>Ott(LaposKerék, Tengely) <math>\wedge</math> Ott(LaposKerék, Föld))</i> <i>Cselekvés(Felszerel(Pótkerék, Tengely),</i> <i>Előfeltétel: Ott(Pótkerék, Föld) <math>\wedge</math> <math>\neg</math>Ott(LaposKerék, Tengely)</i> <i>Következmény: <math>\neg</math>Ott(Pótkerék, Föld) <math>\wedge</math> Ott(Pótkerék, Tengely))</i> <i>Cselekvés(OtthagyÉjszakára,</i> <i>Előfeltétel:</i> <i>Következmény: <math>\neg</math>Ott(Pótkerék, Föld) <math>\wedge</math> <math>\neg</math>Ott(Pótkerék, Tengely) <math>\wedge</math> <math>\neg</math>Ott(Pótkerék, Csomagtartó)</i> $\wedge$ $\neg$ Ott(LaposKerék, Föld) $\wedge$ $\neg$ Ott(LaposKerék, Tengely))
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

11.3. ábra. Az egyszerű pótkerék probléma

## Példa: A kockavilág

Az egyik leghíresebb tervkészítési terület a **kockavilág** (*blocks world*) probléma. A terület asztalapon elhelyezett kockákból áll.<sup>3</sup> A kockákat egymásra rakhajtuk, de egy kockán közvetlenül minden csak egyetlen másik helyezhető el. A kockákat egy robotkarral mozgathatjuk, amely fel tud venni egy kockát, majd azt vagy az asztalra, vagy egy másik kocka tetejére le tudja tenni. A robotkar egyszerre csak egy kockát tud felmelni, vagyis olyat nem, amelynek a tetején egy másik kocka van. A cél minden egy vagy több kockaoszlop építése, amelyekben a kockák egymáshoz képesti elhelyezkedése meghatározott. A cél lehet például két oszlop építése, amelyek közül az egyikben az *A* kocka a *B* tetején van, a másikban pedig a *C* kocka van a *D* tetején.

A *Rajta(b, x)* jelölést használjuk annak leírására, hogy a *b* kocka az *x*-en van, ahol az *x* egy másik kockát vagy az asztalapot jelenti. A *Mozgat(b, x, y)* cselekvés a *b* kockát a *x* tetejéről az *y* tetejére mozgatja. A *b* kocka mozgatásának előfeltétele, hogy rajta semmi ne legyen. Ennek leírása az elsőrendű logikában a  $\neg \exists x \text{ Rajta}(x, b)$  vagy  $\forall x \neg \text{Rajta}(x, b)$ . Az ADL nyelvben ezek előfeltételek lehetnének. A STRIPS nyelv keretei között maradhatunk az *Üres(x)* predikátum bevezetésével, ami akkor igaz, ha semmi nincs *x*-en.

A *Mozgat* cselekvés a *b* kockát az *x*-ről az *y*-ra mozgatja, ha minden a *b*, minden pedig az *y* üres. A mozgatás után az *x* üres, de az *y* már nem. A *Mozgat* formális leírása STRIPS-ben a következő:

*Cselekvés(Mozgat(b, x, y))*

ELŐFELTÉTEL: *Rajta(b, x)  $\wedge$  Üres(b)  $\wedge$  Üres(y)*

KÖVETKEZMÉNY: *Rajta(b, y)  $\wedge$  Üres(x)  $\wedge$   $\neg \text{Rajta}(b, x) \wedge \neg \text{Üres}(y)$*

Sajnos ez a cselekvés nem kezeli jól az *Üres* predikátumot, ha az *x* vagy az *y* az asztalon van.

Ha *x = Asztal*, a cselekvés következményei között szerepel az *Üres(Asztal)* is, de az asztalnak nem kell kiürülnie a mozgatás után, ha pedig *y = Asztal*, megjelenik az *Üres(Asztal)* előfeltétel, holott az asztalnak nem kell üresnek lenni ahhoz, hogy bármit is tehessünk rá. Ennek kiküszöbölésére két dolgot tehetünk. Először is bevezetünk egy új cselekvést, amellyel egy *b* kockát az asztalra tehetünk:

*Cselekvés(AsztalraTesz(b, x, y))*

ELŐFELTÉTEL: *Rajta(b, x)  $\wedge$  Üres(b)*

KÖVETKEZMÉNY: *Rajta(b, Asztal)  $\wedge$  Üres(x)  $\wedge$   $\neg \text{Rajta}(b, x)$*

Másodszor az *Üres(b)* predikátumot úgy értelmezhetjük, hogy „*b* tetején van elég szabad hely, ahol a kockát letehetjük”. Ezek szerint az *Üres(Asztal)* minden igaz. Az egyetlen probléma, hogy semmi nem gátolja a tervkészítőt abban, hogy a *Mozgat(b, x, Asztal)* cselekvést alkalmazza az *AsztalraTesz(b, x)* helyett. Vagy együtt élünk ezzel a problémával (amely egyébként a szükségesnél nagyobb keresési teret eredményez, de nem vezet hibás válaszokhoz), vagy bevezethetjük a *Kocka* predikátumot, és a *Mozgat* cselekvés előfeltételeihez hozzátehetjük a *Kocka(x)  $\wedge$  Kocka(b)* részt.

<sup>3</sup> A tervkészítés kutatásában használt kockavilág sokkal egyszerűbb, mint az SHRDLU verzió, amit az 52. oldalon mutattunk be.

$Kiindulás(Rajta(A, Asztal) \wedge Rajta(B, Asztal) \wedge Rajta(C, Asztal)$   
 $\wedge Kocka(A) \wedge Kocka(B) \wedge Kocka(C)$   
 $\wedge Üres(A) \wedge Üres(B) \wedge Üres(C))$   
**Cél(Rajta(A, B) \wedge Rajta(B, C))**  
**Cselekvés(Mozgat(b, x, y),**  
 ELŐFELTÉTEL:  $Rajta(b, x) \wedge Üres(b) \wedge Üres(y) \wedge Kocka(b) \wedge$   
 $(b \neq x) \wedge (b \neq y) \wedge (x \neq y)$   
 KÖVETKEZMÉNY:  $Rajta(b, y) \wedge Üres(x) \wedge \neg Rajta(b, x) \wedge \neg Üres(y))$   
**Cselekvés(AsztalraTesz(b, x),**  
 ELŐFELTÉTEL:  $Rajta(b, x) \wedge Üres(b) \wedge Kocka(b) \wedge (b \neq x)$   
 KÖVETKEZMÉNY:  $Rajta(b, Asztal) \wedge Üres(x) \wedge \neg Rajta(b, x))$

**11.4. ábra.** A kockavilág tervkészítési problémája: egy három kockából álló torony építése. A  $[Mozgat(B, Asztal, C), Mozgat(A, Asztal, B)]$  cselekvéssor egy lehetséges megoldás.

Végül itt van még az olyan hibás műveletek esete, mint a  $Mozgat(B, C, C)$ , amely hatástanlan kellene legyen, ehelyett azonban ellentmondó következményekhez vezet. Általában nem foglalkozunk az ilyen jellegű problémákkal, mert nemigen vannak hatással az előállított tervezekre. A helyes megközelítés, hogy egyenlőtlenségi előfeltételeket vezetünk be, amint azt a 11.4. ábra mutatja.

## 11.2. TERVKÉSZÍTÉS ÁLLAPOTTÉR-KERESÉssel

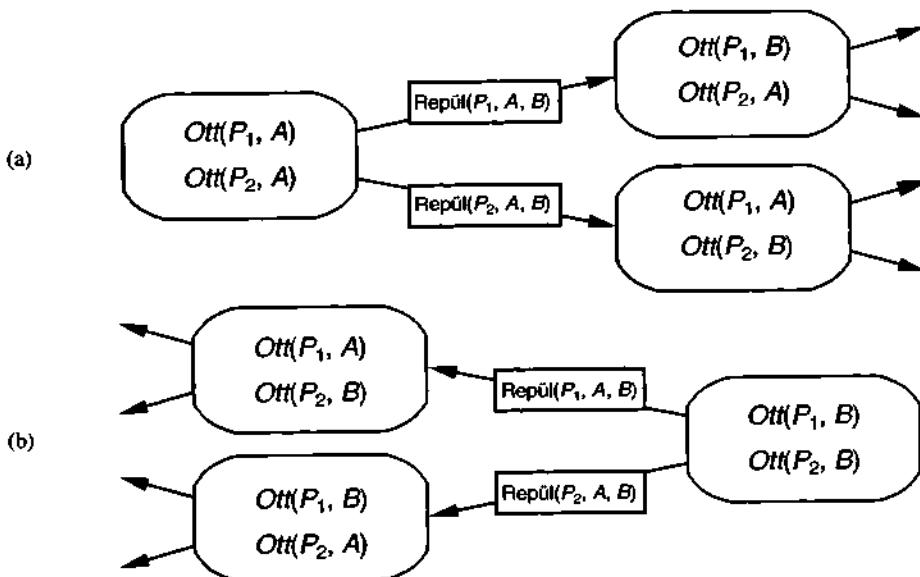
Most fordítunk figyelmünket a tervkészítő algoritmusok felé. A legkézenfekvőbb megközelítés az állapottér-keresés. Mivel a tervkészítési problémák cselekvéseinek leírása tartalmazza az előfeltételeket és a következményeket, a keresés minden irányban végrehajtható: vagy előrefelé egy kiindulási állapotból, vagy visszafelé a céltól, amint azt a 11.5. ábra is mutatja. Mindezeken túl felhasználhatjuk az explicit cselekvés- és cél-reprezentációkat, hogy automatikusan hatékony heurisztikákat származtassunk.

### Előrefelé keresés az állapottérben

Ez előrefelé kereséssel történő tervkészítés hasonlít a 3. fejezetben bemutatott probléma-megoldó megközelítéshez. Ezt progresszív tervkészítésnek nevezzük, mert előrefelé haladva dolgozik.

A feladat kiinduló állapotából indulva végignézzük a lehetséges cselekvéssorozatokat, amíg olyat nem találunk, ami a célállapothoz vezet. A tervkészítési probléma állapottér-keresési problémaként formalizálva a következő:

- A keresés **kiindulási állapota (initial state)** megegyezik a tervkészítési probléma kiindulási állapotával. Általanosságban minden állapot pozitív alapliterálok egy hal-maza; az itt nem szereplő literálok hamisak.
- Egy állapotban minden olyan **cselekvés (action)** alkalmazható, aminek az előfeltételei teljesülnek. A cselekvés végrehajtása után következő állapotot úgy állítjuk elő,



**11.5. ábra.** A tervkészítés két megközelítése. (a) Előrefelé (progresszív) állapottér-keresés, a kiinduló állapotból indulva és a probléma cselekvéseit használva halad a cél felé. (b) Visszafelé (regressziós) állapottér-keresés: valószínűségi állapot keresés (lásd 124. oldal) a céllállapot(ok)ból indulva a cselekvések inverzét alkalmazva keresi visszafelé a kezdeti állapotot.

hogy a következményrész pozitív literáljait hozzáadjuk, negatív literáljait pedig töröljük. (Elsőrendű logika esetén az előfeltételekből az egyesítőt alkalmazni kell a következményliterálokra.) Megjegyezzük, hogy annak következményeként, hogy egy explicit cselekvés reprezentációt használunk, egyetlen állapotátmenet-függvény működik az összes tervkészítő problémára.

- A célteszt (goal test) ellenőrzi, hogy az adott állapot kielégíti-e a tervkészítési probléma célját.
- A lépésköltség (step cost) minden cselekvésre tipikusan 1. Habár könnyű lenne a különböző cselekvésekhez különböző költségeket hozzárendelni, ezt a STRIPS tervkészítőkben nagyon ritkán alkalmazzák.

Emlékezzünk vissza, hogy a függvényszimbólumok hiányában a tervkészítési probléma állapottere véges, ezért teljes gráf keresési algoritmusok, mint például az A\* keresés, egyben teljes tervkészítő algoritmusok is.

A tervkészítők kutatásának korai napjaitól (kb. 1961-től) egészen mostanáig (kb. 1998-ig) az a feltételezés élte, hogy az előrefelé kereső algoritmusok nem eléggyé hatékonyak a gyakorlati alkalmazásra. Ma már, tekintsünk csak vissza a 11.1. alfejezetre, ezt nem nehéz megindokolni. Először is az előrefelé keresés nem foglalkozik a lényegtelen cselekvések problémájával, mivel minden állapotban minden alkalmazható cselekvést figyelembe vesz. Másodszor pedig ez a megközelítés gyorsan elakad egy jó heurisztika nélkül. Vegyük például a légi csomagszállítási problémát 10 repülőtérrrel, egyenként 5-5 repülővel és 20 darab csomaggal. A cél, hogy az A repülőtér

összes terhét a  $B$  repülőtére szállítsuk. Van egy egyszerű megoldása a feladatnak: tegyük be minden a 20 csomagot az egyik  $A$ -beli repülőgéphez, ezzel repüljünk  $B$ -be, és ott rakodjunk ki. A megoldás megkeresése ellenben nagyon nehéz lehet, hiszen az átlagos elágazási faktor óriási: minden az 50 repülő 9 másik reptérre tud repülni, míg minden a 200 csomag kirakható (ha be volt rakva), vagy berakható a reptér bármelyik repülőjébe. Átlagosan mondjuk 1000 lehetséges cselekvés hajtható végre, így a keresési fában a kézenfekvő megoldás mélységéig közel 1000<sup>41</sup> csomópont van. Ebből nyilvánvaló, hogy egy megfelelő heurisztikára lesz szükség, hogy ezt a keresést hatékonytá tegyük. A hátrafelé keresés áttekintése után néhány lehetséges heurisztikát mutatunk be.

## Hátrafelé keresés az állapottérben

Az állapottérben történő hátrafelé keresést a kétirányú keresés egy részeként a 3. fejezetben röviden már bemutattuk. Ott megjegyeztük, hogy a hátrafelé keresés megvalósítása nagyon nehéz lehet, ha a célállapotokat az explicit megadás helyett megkötlesekkel határozzuk meg. Nevezetesen nem minden nyilvánvaló, hogy hogyan generáljuk le a célállapotok halmozának a lehetséges elődállapotait (*predecessors*). Látjuk majd, hogy a STRIPS reprezentáció esetén ez meglehetősen egyszerű, mert az állapothalmazokat azokkal a literálokkal írhatjuk le, amelyeknek igaz értéküknek kell lenni az adott állapotokban.

A hátrafelé keresés fő előnye az, hogy lehetővé teszi számunkra, hogy csak a **releváns (relevant)** cselekvéseket vegyük figyelembe. Egy cselekvés releváns egy konjunktív cél szempontjából, ha eléri a cél egy konjunktját. Például a 10-repülőteres légi csomagszállítási problémánk esetében a cél az, hogy a 20 csomag a  $B$  repülőtéren legyen, vagy pontosabban

$$Ott(C_1, B) \wedge Ott(C_2, B) \wedge \dots \wedge Ott(C_{20}, B)$$

Vegyük az  $Ott(C_1, B)$  literált. Hátrafelé dolgozva, olyan cselekvéseket kereshetünk, melyeknek ez a következménye. Ilyen csak egy van:  $Kirakodás(C_1, p, B)$ , ahol a  $p$  repülőgép meghatározatlan.

Vegyük észre, hogy számos *irreleváns* cselekvés is van, ami szintén a célállapotra vezet. Például elrepítetünk egy üres gépet a *JFK* repülőtérről (John Fitzgerald Kennedy repülőtér New Yorkban) az *SFO*-ra (San Francisco repülőtere), ami a célállapothoz vezet egy olyan elődállapotból, melyben a repülőgép a *JFK*-n van, és minden cél-feltétel teljesül. Egy hátrafelé keresés, ami megengedi a hatástalan cselekvéseket még teljes, de sokkal kevésbé hatékony. Ha létezik megoldás, akkor azt egy olyan hátrafelé keresés is megtalálja, amely csak releváns cselekvéseket enged meg. A hátrafelé keresés szűkítése a releváns cselekvésekre, gyakran jóval alacsonyabb elágazási tényezőt eredményez, mint az előrefelé keresés. Például a légi szállítási problémánk esetén a kiindulási állapotból előrefelé közel 1000 cselekvés hajtható végre, míg a célból visszafelé indulva minden össze 20.

A visszafelé keresést **regressziós (regression)** tervkészítésnek is nevezik. A regressziós keresés meghatározó kérdése a következő: melyek azok az állapotok, amelyekből egy adott cselekvés a célhoz vezet? Ezen állapotok leírásának számítását a cél adott

állapoton kereszti **regresszálásának (regressing)** nevezzük. Hogy lássuk ennek menetét, vegyük a légi szállítási feladatot. Adott a cél az

$$Ott(C_1, B) \wedge Ott(C_2, B) \wedge \dots \wedge Ott(C_{20}, B)$$

és a releváns cselekvés a *Kirakodás*( $C_1, p, B$ ), ami az első literált adja. A cselekvés csak akkor működik, ha az előfeltételei teljesülnek. Ebből kifolyólag az elődállapotnak tartalmaznia kell a *Benne*( $C_1, p$ )  $\wedge$  *Ott*( $p, B$ ) előfeltételeket. Mindezeken túl az *Ott*( $C_1, B$ ) részcél nem lehet igaz a megelőző állapotban.<sup>4</sup> Így a megelőző állapot leírása az alábbi:

$$Benne(C_1, p) \wedge Ott(p, B) \wedge Ott(C_2, B) \wedge \dots \wedge Ott(C_{20}, B)$$

Ennek tetejébe, ha elvárjuk, hogy egy cselekvés kielégítsen egy literált, azt is biztosítanunk kell, hogy nem *ront el* egy másikat. Az olyan cselekvést, ami megfelel ennek az elvárásnak **konzisztensnek (consistent)** nevezzük. Például a *Berakodás*( $C_2, p$ ) cselekvés nem lenne konzisztens az aktuális céllal, mert negálja az *Ott*( $C_2, B$ ) literált.

Most, hogy a relevanciát és a konzisztenciát definiáltuk, megadhatunk egy általános módszert a hátrafelé keresés elődállapotának kereséséhez. Adott egy  $G$  cél leírása. Legyen az  $A$  cselekvés releváns és konzisztens. A megfelelő elődállapot a következő:

- Az  $A$  minden pozitív következményét, ami szerepel  $G$ -ben, töröljük.
- Az  $A$  minden, még nem szereplő előfeltételét hozzáadjuk.

A standard keresési algoritmusok bármelyike felhasználható a keresésre. A leállási feltétel, hogy a generált elődállapot leírását a keresés kiindulási állapota kielégítse. Az elsőrendű esetben, ehhez néhány változó behelyettesítésére is szükség lehet az elődállapotának leírásában. Például az előző bekezdés elődállapotának leírását a

$$Benne(C_1, P_{12}) \wedge Ott(P_{12}, B) \wedge Ott(C_2, B) \wedge \dots \wedge Ott(C_{20}, B)$$

kiindulási állapot a  $\{p/P_{12}\}$  behelyettesítéssel teljesíti. A behelyettesítéseket arra a cselekvésre kell végrehajtani, ami az állapottól a célohoz vezet, így eredményezve a [*Kirakodás*( $C_1, P_{12}, B$ )] megoldást.

## Állapottér-keresési heurisztikák

Sem az előre-, sem pedig a hátrafelé keresés nem hatékony egy jó heurisztika nélkül. Emlékezzünk vissza a 4. fejezetből, hogy egy heurisztikus függvény egy állapot cél-állapottól való távolságát becsl; a STRIPS esetén minden cselekvés költsége 1, így a távolság az alkalmazandó cselekvések száma. Az alapötlet, hogy nézzük a cselekvés következményeit és a célokat, és becsüljük meg, hogy hány lépéstre van szükség a célok elérésére. A pontos szám meghatározása NP-teljes probléma, de a legtöbb esetben találhatók kellően pontos becslők, melyek nem nagyon számításigényesek. Szintén képesek lehetünk egy **elfogadható (admissible)** heurisztika származtatására, azaz olyanra, ami nem becsül túl. Ez az A\* kereséssel együtt használható optimális megoldások megkeresésére.

<sup>4</sup> Ha a részről igaz lenne az elődállapotban, akkor a cselekvés még mindig a célállapothoz vezetné. Ellenben az ilyen cselekvések hatástalanok (irrelevant), mivel nem teszik igazzá a célt.

Két megközelítést lehet kipróbálni. Az egyik, hogy a megadott feladatleírásból egy relaxált problémát (*relaxed problem*) származtatunk, ahogy azt a 4. fejezetben bemutattuk. A relaxált problémára (ami várhatóan könnyen megoldható) adott optimális megoldás költsége elfogadható heurisztikát ad az eredeti problémára. A második megközelítés során feltételezzük, hogy a tiszta „oszd meg és uralkodj” algoritmus működni fog. Ezt nevezzük a részcélfüggetlenség (*subgoal independence*) feltételezésnek: a részcélok konjunkciójának megoldási költségét, az egymástól *függetlenül* megoldott részcélok költségeinek összegével becsüljük. A részcélfüggetlenség feltételezés lehet optimista vagy pesszimista. Akkor optimista, hogyha vannak negatív kölcsönhatások a részcélokokhoz készített részterek között (például amikor az egyik részterv egy cselekvése töröl egy célt, melyet egy másik részterv ért el). Akkor pesszimista és egyben elfogadhatatlan is, ha a részterek redundáns cselekvéseket tartalmaznak (például két cselekvés egygyel helyettesíthető az összefésült tervben).

Vizsgáljuk meg, hogyan származtathatunk relaxált tervkészítési problémákat. Mivel az előfeltételek és a következmények explicit megadásai rendelkezésre állnak, az eljárás ezeket a reprezentációkat módosítja. (Vessük össze ezt a megközelítést a keresési feladatokkal, ahol az állapotámenet-függvény egy fekete doboz.) A legegyszerűbb ötlet, hogy lazítsuk a problémát úgy, hogy *eltávolítjuk az összes előfeltételt a cselekvések ből*. Ekkor minden cselekvés minden alkalmazhatóvá válik, és minden literál elérhető egy lépésben (ha van alkalmazható cselekvés, egyébként a cél elérése lehetetlen). Úgy tűnhet, ez azt jelenti, hogy a célok konjunkciójához vezető lépésszám azonos a kielégítetlen célliterálok számával, de nem egészen; (1) lehet két cselekvés, melyek kölcsönösen törlik a másik által teljesített célliterált, (2) néhány cselekvés több célt is elérhet. Ha kombináljuk a relaxált problémát és a részcél függetlenségi feltételezést, minden problémát kiküszöböltnek vehetjük, és az előálló heurisztika pontosan a még kielégítetlen célok száma.

Sokszor pontosabb heurisztika nyerhető, ha legalább a több célt kielégítő cselekvések pozitív kölcsönhatásait figyelembe vesszük. Először tovább relaxáljuk a problémát azáltal, hogy *eltávolítjuk a negatív következményeket* (lásd 11.6. feladat). Ezután kiszámítjuk, hogy mennyi az a minimális cselekvésszám, melyek pozitív következményei-nek uniója kielégíti a célt. Vegyük például a

*Cél(A  $\wedge$  B  $\wedge$  C)*

*Cselekvés(X, KÖVETKEZMÉNY: A  $\wedge$  P)*

*Cselekvés(Y, KÖVETKEZMÉNY: B  $\wedge$  C  $\wedge$  Q)*

*Cselekvés(Z, KÖVETKEZMÉNY: B  $\wedge$  P  $\wedge$  Q)*

A minimális halmaz az {X, Y} cselekvés, ami az {A, B, C} célt kielégíti, így a fedőhalmaz-heurisztika szerinti költség 2. Ez javít a részcél függetlenség feltételezésen, ami 3-as költséget adna. Egyetlen apró probléma van: a fedőhalmaz probléma NP-teljes. Az egyszerű fedőhalmaz-keresési algoritmus garantáltan visszaad egy értéket, ami egy log  $n$ -es faktor pontossággal az igazi minimumérték, ahol  $n$  a cél literáljainak száma, és a gyakorlatban rendszerint ennél sokkal pontosabb. Sajnos a mohó algoritmus elveszíti a heurisztika elfogadhatóságának garanciáját.

Az is lehetséges, hogy a relaxált problémát a negatív következmények eltávolításával állítjuk elő anélkül, hogy az előfeltételeket elhagynánk. Ez annyit tesz, hogy ha egy cselekvés következménye  $A \wedge \neg B$  az eredeti problémában, akkor A a relaxált problémában. Ez annyit tesz, hogy nem kell foglalkoznunk a részterek közötti negatív köl-

csönhatásokkal, mert egyetlen cselekvés sem törölhet egy másik által előállított literált. Az így előálló relaxált feladat megoldásának költsége az üres-törlési-lista (*empty-delete-list*) heurisztikát adja. A heurisztika meglehetősen pontos, de kiszámítása egy (egyszerű) tervkészítő algoritmus futtatását igényli. A gyakorlatban a relaxált probléma megoldása elég gyors ahhoz, hogy megérje alkalmazni.

Az itt bemutatott heurisztika mind a progresszív (előrefelé), mind pedig a regresszív (hátrafelé) irányba használható. A könyv írásának pillanatában az üres-törlési-lista heurisztikát használó előrefelé keresők a listavezetők. Ez valószínűleg változik, ahogy újabb heurisztikák és keresési technikák jelennek meg. Mivel a tervkészítés exponenciális komplexitású,<sup>5</sup> nincs olyan algoritmus, amely minden problémára hatékony lesz, de számos gyakorlati feladat oldható meg ezen fejezet heurisztikáival – sokkal több, mint ami néhány éve megoldható volt.

### 11.3. RÉSZBEN RENDEZETT TERVKÉSZÍTÉS

Az előre- és hátrafelé keresés a *teljesen rendezett* terv keresés speciális fajtái. Ezek a cselekvéseknek csak szigorúan lineáris sorozatait tárgák fel, melyek közvetlenül kapcsolódnak a kiinduláshoz vagy a célhöz. Ez annyit tesz, hogy nem használhatják ki a problémadekompozíció lehetőségeit. Ahelyett hogy a részproblémákon külön dolgoznának, minden arról kell döntenüük, hogy hogyan sorrendezzék a részproblémák cselekvéseinak összességét. Előnyösebb lenne egy olyan megközelítés, ami több részcélon dolgozik függetlenül, ezeket több résztervel megoldja, majd összerakja a részterveket.

Egy ilyen megközelítésnek megvan az az előnye is, hogy rugalmas a terv összeállításának sorrendjében. A tervkészítő így nem kényszerül arra, hogy a lépésekben időrendi sorrendben dolgozzon, hanem először inkább a „nyilvánvaló” vagy „fontos” döntéseken dolgozhat. Például egy Berkeleyben lévő tervkészítő ágens, amely Monte Carlóból igyekszik, először egy San Francisco-Párizs járatot vizsgál meg, majd amikor ismeri a pontos indulási és érkezési dátumokat, foglalkozhat a reptérre kijutás, illetve az onnan továbbjutás problémájával.

Azt az általános stratégiát, mely szerint egy választást készletetünk egy keresés során, **legkisebb megkötés** (*least commitment*) stratégiának nevezzük. A legkisebb megkötésnek nincs formális definíciója, és nyilvánvaló, hogy valamennyi megkötés szükséges, különben a keresés nem haladna. Az informális definíció ellenére a legkisebb megkötés nagyon hasznos megközelítés annak meghatározására, hogy a döntéseket mikor kell meghozni egy keresési problémában. Az első példánk jóval egyszerűbb, mint egy nyaralás megtervezése. Vegyük a cipőfelvétel egyszerű esetét. Ezt formálisan a következő tervkészítési problémával írhatjuk le:

*Cél(JobbCipőFenn  $\wedge$  BalCipőFenn)*

*Kiindulás()*

*Cselekvés(JobbCipő, ELŐFELTÉTEL: JobbZokniFenn, KÖVETKEZMÉNY: JobbCipőFenn)*

*Cselekvés(JobbZokni, KÖVETKEZMÉNY: JobbZokniFenn)*

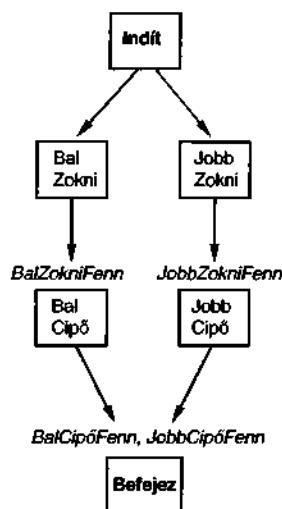
*Cselekvés(BalCipő, ELŐFELTÉTEL: BalZokniFenn, KÖVETKEZMÉNY: BalCipőFenn)*

*Cselekvés(BalZokni, KÖVETKEZMÉNY: BalZokniFenn)*

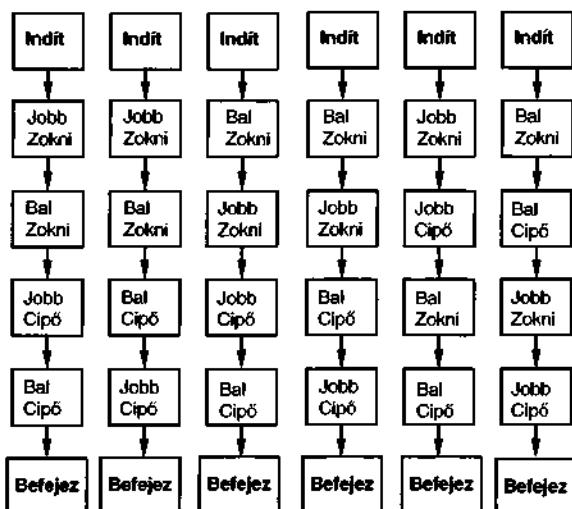
<sup>5</sup> Technikailag a STRIPS-alapú tervkészítés PSPACE-teljes, kivéve, ha a cselekvéseknek csak pozitív előfeltételei vannak egy következmény literálal (Bylander, 1994).

A tervkészítőnek két cselekvéssorozattal kell előállnia, a *JobbZokni*-t a *JobbCipő* követi, hogy a cél első konjunktját elérjük és a *BalZokni*-t a *BalCipő* követi a második konjunkthoz. A két cselekvéssorozat kombinálható, hogy teljes tervet nyerjünk. Így a tervkészítő a két részsorozatot egymástól függetlenül kezelheti anélkül, hogy felállítsa bármi megkötést arra, hogy az egyik sorozat egy cselekvése a másikhoz képest előbb vagy később következik. Bármely olyan tervkészítő algoritmust, ami két cselekvést be tud illeszteni egy tervbe anélkül, hogy meghatározná azok sorrendjét, részben rendezett tervkészítőnek nevezünk (*partial-order planner*). A 11.6. ábra a cipő és zokni felvételére készített részben rendezett tervet szemlélteti. Vegyük észre, hogy a megoldást a cselekvések gráfja ábrázolja, nem egy szekvencia. Érdemes megfigyelni az *Indít* és a *Befejez* „technikai” cselekvéseket, melyek a terv kezdetét, illetve végét jelölik. Ezeket cselekvésnek nevezni egyszerűsíti a dolgokat, mert így a terv minden lépései egy cselekvés. A részben rendezett megoldás hat teljesen rendezett megoldásnak felel meg, melyek mindegyike a részben rendezett megoldásnak egy sorba rendezése (*linearization*).

Részben rendezett terv:



Teljesen rendezett terv:



11.6. ábra. Egy részben rendezett terv a zokni és a cipő felvételéhez, és a hat lehetséges sorba rendezés a teljesen rendezett tervhez

A részben rendezett tervkészítés megvalósítható, mint egy keresés a részben rendezett tervek terében. (Ezentúl ezeket egyszerűen csak „terv”-nek nevezzük.) Egy üres tervvvel indulunk. Ezután a terv finomításának módjait keressük egészen addig, míg elő nem állunk a teljes tervvvel, ami megoldása a problémának. A cselekvések ebben a keresésben nem a való világ cselekvései, hanem terveken végrehajtott műveletek: egy lépés hozzáadása a tervhöz, a cselekvésekhez egy sorrend rendelése, ami egy cselekvést egy másik elő helyez és így tovább.

A továbbiakban definiálni fogjuk az RRT algoritmust a részben rendezett tervkészítéshez. Hagyományosan az RRT-t mint egy önálló programot szokás bemutatni, ehelyett

mi a részben rendezett tervkészítést, mint a keresési probléma egy példányát fogjuk megfogalmazni. Ez lehetővé teszi, hogy az alkalmazható tervfinomító lépésekre koncentráljunk ahelyett, hogy azt vizsgálnánk, hogyan térképezi fel a keresési teret az algoritmus. Valójában, ha a keresési feladatot megfogalmaztuk a nem informált vagy heurisztikus keresési módszerek széles választéka áll rendelkezésre.

Emlékezzünk vissza, hogy a keresésünk állapotai (többnyire még befejezetlen) tervek lesznek. Hogy elkerüljük a valós cselekvésekkel való keveredést, itt tervekről és nem állapotokról fogunk beszélni. minden terv a következő négy komponensből áll, ahol az első kettő a terv lépései definiálja, míg az utolsó kettő naplózásra szolgál, hogy abból meghatározzuk, hogyan bővíthetők a tervezek:

- Cselekvések (actions)** halmaza, amelyek a terv lépései adják. Ezeket a tervkészítési probléma cselekvéshalmazából vesszük. Egy „üres” terv csak az *Indít* és a *Befejez* cselekvéseket tartalmazza. Az *Indít*-nak nincsenek előfeltételei, míg a következményes a tervkészítési feladat kiindulási állapotának literáljait tartalmazza. A *Befejez* cselekvésnek nincsen következmény része, míg előfeltételei megegyeznek a célállapot literáljaival.
- Rendezési kényszerek (ordering constraints).** minden rendezési kényszer  $A \prec B$  alakkú, amit „ $B$  előtt  $A$ ”-nak olvasunk, és azt jelenti, hogy az  $A$ -t valamikor a  $B$  előtt kell végrehajtani, de nem feltétlenül közvetlen előtte. A rendezési kényszerek egy megfelelő részben rendezést kell leírjanak. Bármilyen ciklus (mint az  $A \prec B$  és a  $B \prec A$ ) ellentmondást jelent, ezért olyan rendezési kényszer nem adható a tervhez, amely ciklust hozna létre.
- Okozati kapcsolatok (causal links).** Az  $A$  és  $B$  cselekvések közötti okozati kapcsolatot  $A \xrightarrow{p} B$ -vel jelöljük és úgy olvassuk, hogy „ $A$  teljesíti  $p$ -t  $B$ -hez”. Például a *JobbZokni*  $\xrightarrow{\text{JobbZokniFenn}}$  *JobbCipő* okozati kapcsolat jelentése, hogy a *JobbZokniFenn* egy következménye a *JobbZokni* cselekvésnek és előfeltétele a *JobbCipő*-nek. Tartalmazza azt a megköztét is, hogy a *JobbZokniFenn*-nek igaznak kell maradnia a *JobbZokni* cselekvéstől a *JobbCipő* cselekvésig. Más szavakkal, a terv nem minden bővíthető egy új  $C$  cselekvéssel, ami ütközik egy okozati kapcsolattal. Egy  $C$  cselekvés ütközik az  $A \xrightarrow{p} B$ -vel, ha  $C$  következménye a  $\neg p$ , és ha  $C$  az  $A$  után és  $B$  előtt jöhét (a rendezési kényszereknek megfelelően). Néhány szerző az okozati kapcsolatokat védeott tartományoknak (*protection intervals*) nevezi, mert az  $A \xrightarrow{p} B$  megvédi  $p$ -t a negálástól az  $A$ - $B$  intervallumban.
- Nyitott előfeltételek (open preconditions).** Egy előfeltétel nyitott, ha nem teljesül a terv egy akciójának hatására. A tervkészítők feladata, hogy a nyitott előfeltételek halmazát üres halmazra csökkentse, ellentmondások bevezetése nélkül.

Például a 11.6. ábrán bemutatott teljes terv a következő komponenseket tartalmazza (nem tüntettük fel a rendezési kényszereket, amelyek minden cselekvést az *Indít* után és a *Befejez* elő helyeznek):

Cselekvések: {*JobbZokni*, *JobbCipő*, *BalZokni*, *BalCipő*, *Indít*, *Befejez*}

Rendezések: {*JobbZokni*  $\prec$  *JobbCipő*, *BalZokni*  $\prec$  *BalCipő*}

Kapcsolatok: {*JobbZokni*  $\xrightarrow{\text{JobbZokniFenn}}$  *JobbCipő*, *BalZokni*  $\xrightarrow{\text{BalZokniFenn}}$  *BalCipő*,  
*JobbCipő*  $\xrightarrow{\text{JobbCipőFenn}}$  *Befejez*, *BalCipő*  $\xrightarrow{\text{BalCipőFenn}}$  *Befejez*}

Nyitott előfeltételek: {}

**Konzisztens tervnek (consistent plan)** nevezik azt a tervet, amelyben nincsenek ciklusok a rendezési megkötésekben és nincsenek ellentmondások az okozati kapcsolatokkal. Egy konzisztens terv, mely nyitott előfeltételektől mentes, a **megoldás (solution)**. Rövid gondolkodással belátható: *egy részben rendezett megoldás minden sorba rendezése egy teljesen rendezett megoldás, melynek végrehajtása az induló állapotból egy célállapotra vezet.* Ez annyit tesz, hogy a „terv végrehajtását” a teljesen rendezett tervekről kiterjeszthetjük a részben rendezett tervekre. Egy részben rendezett terv végrehajtása során ismétlődően minden választunk egy következő cselekvést, a lehetséges következő cselekvések **bármelyikét**. A 12. fejezetben látni fogjuk, hogy az ágens számára a terv végrehajtásának rugalmassága nagyon hasznos lehet, ha a környezet nem az elvártaknak megfelelően működik. A rugalmas sorrendezés egyszerűsíti a kisebb tervek összekombinálását nagyobbakká, mert minden részterv átrendezheti a cselekvéseinek sorrendjét, hogy elkerülje a más tervekkel való ütközést.

Most már készen állunk arra, hogy formalizáljuk azt a keresési problémát, amit az RRT megold. Egy olyan formalizálással kezdünk, ami az ítéletlogikai tervkészítési problémákra alkalmas, az elsőrendű logikai megoldással járó komplikációkat későbbre hagyjuk. Mint rendszerint, a definíció tartalmazza a kiinduló állapotot, a cselekvéseket és a céltesztet.

- A kiindulási terv egy *Indít* és egy *Befejez* cselekvést, egy *Indít*  $\prec$  *Befejez* rendezést tartalmaz, és nincsenek okozati kapcsolatok. A *Befejez* minden előfeltétele nyitott.
- Az állapotátmenet-függvény önkényesen kiválasztja *B* egy *p* nyitott előfeltételét és legenerál egy következő tervet egy *A* cselekvés minden olyan lehetséges konzisztens megválasztásához, mely teljesíti *p*-t. A konziszenciát a következőképp biztosítjuk:
  1. Az  $A \xrightarrow{p} B$  okozati kapcsolatot és az  $A \prec B$  rendezési megkötést hozzáadjuk a tervhez. Az *A* cselekvés lehet a terv egy már létező vagy egy új cselekvése. Amennyiben új, hozzáadjuk a tervhez az *Indít*  $\prec$  *A* és az *A*  $\prec$  *Befejez* megkötésekkel.
  2. Feloldunk minden konfliktust, az új okozati kapcsolat és az összes létező cselekvés, valamint az *A* cselekvés (ha új) és minden létező okozati kapcsolat között. Egy az  $A \xrightarrow{p} B$  és *C* között fennálló konfliktust azáltal oldunk fel, hogy *C*-t valamikor kívül helyezzük a védett intervallumon a  $B \prec C$  vagy a  $C \prec A$  hozzáadásával. Amennyiben konzisztens tervet adnak, beszúrjuk a követő állapotokat, valamelyikhez vagy akár minden kettőhöz.
- A célteszt ellenőrzi, hogy a terv megoldása-e az eredeti tervkészítési problémának. Mivel csak konzisztens terveket állítunk elő, a céltesztnek csak azt kell ellenőrizni, hogy vannak-e nem nyitott előfeltételek.

Emlékezzünk, hogy keresési algoritmus által vizsgált cselekvések ebben a felírásban sokkal inkább tervfinomító lépések, mint valós cselekvések a feladattérből. Az út költsége ezért szigorú értelemben véve irreleváns, hiszen az egyetlen ami fontos, az azon terv valós cselekvéseinek költsége, amire ez az út vezetett. Mindazonáltal *lehetséges* egy útvonal-költség-függvény specifikálása, ami a valós terv költségeit tükrözi: minden valós cselekvésnek a tervhez való hozzáadását egységnyi költséggel, míg a finomító lépéseket 0 költséggel vesszük figyelembe. Így  $g(n)$ , ahol *n* a terv, egyenlő lesz a tervben található valós cselekvések számával. Egy heurisztikus  $h(n)$  becslés szintén alkalmazható.

Első ránézésre azt gondolhatjuk, hogy az állapotátmenet-függvénynek minden nyitott *p*-hez tartalmaznia kellene egy következő állapotot, nem csak egyhez. Ez azonban,

redundáns lenne és nem lenne hatékony, ugyanazon okból, amiért a kényszerkielégítési algoritmusok nem szúrnak be minden lehetséges változóhoz utódot: a nyitott előfeltételek kezelési sorrendje (csakúgy, mint a kényszerkielégítési probléma változóinak kezelési sorrendje) kommutatív. (Lásd 187. oldal.) Így tetszőleges sorrendet választhatunk, és az algoritmusaink még mindig teljesek. A helyes rendezés megválasztása gyorsabb kereséshez vezethet, de minden sorrendezés ugyanazokra a megoldásjelöltekre vezet.

## Példa a részben rendezett tervkészítésre

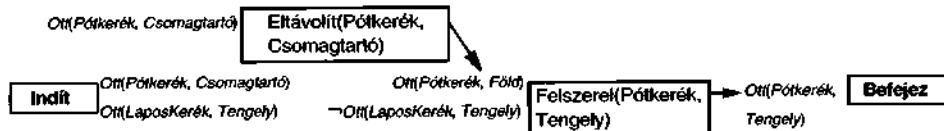
Most nézzük meg, hogy az RRT hogyan oldja meg a 11.1. alfejezetben szereplő pótkerék problémát. A probléma a leírását a 11.7. ábrán megismételjük.

*Kiindulás(Ott(LaposKerék, Tengely)  $\wedge$  Ott(Pótkerék, Csomagtartó)  
 Cél(Ott(Pótkerék, Tengely))  
 Cselekvés(Eltávolít(Pótkerék, Csomagtartó),  
 Előfeltétel: Ott(Pótkerék, Csomagtartó)  
 Következmény:  $\neg$ Ott(Pótkerék, Csomagtartó)  $\wedge$  Ott(Pótkerék, Föld))  
 Cselekvés(Eltávolít(LaposKerék, Tengely),  
 Előfeltétel: Ott(LaposKerék, Tengely)  
 Következmény:  $\neg$ Ott(LaposKerék, Tengely)  $\wedge$  Ott(LaposKerék, Föld))  
 Cselekvés(Felszerel(Pótkerék, Tengely),  
 Előfeltétel: Ott(Pótkerék, Föld)  $\wedge$   $\neg$ Ott(LaposKerék, Tengely)  
 Következmény:  $\neg$ Ott(Pótkerék, Föld)  $\wedge$  Ott(Pótkerék, Tengely))  
 Cselekvés(OtthagyÉjszakára,  
 Előfeltétel:  
 Következmény:  $\neg$ Ott(Pótkerék, Föld)  $\wedge$   $\neg$ Ott(Pótkerék, Tengely)  $\wedge$   $\neg$ Ott(Pótkerék, Csomagtartó)  
 $\wedge$   $\neg$ Ott(LaposKerék, Föld)  $\wedge$   $\neg$ Ott(LaposKerék, Tengely))*

11.7. ábra. Az egyszerű kerékcseré probléma

A megoldás keresése egy kezdeti tervből indul, ami egy *Indít* cselekvést tartalmaz az *Ott(Pótkerék, Csomagtartó)  $\wedge$  Ott(LaposKerék, Tengely)* következménnyel és egy *Befejez* cselekvést az *Ott(Pótkerék, Tengely)* előfeltétellel. Ezután követő állapotokat állítunk elő úgy, hogy egy nyitott előfeltételt választunk a munkához (visszavonhatatlanul), majd választunk az ehhez vezető lehetséges cselekvések közül. Most nem foglalkozunk a heurisztikákkal, amik segíthetik ezt a döntést; látszólag véletlenszerű döntést hozunk. Az események sorrendje a következő:

1. Vegyük az egyetlen nyitott előfeltételt, a *Befejez* cselekvés *Ott(Pótkerék, Tengely)* feltételét. Válasszuk ki az egyetlen alkalmazható cselekvést, a *Felszerel(Pótkerék, Tengely)-t*.
2. Vegyük a *Felszerel(Pótkerék, Tengely)* cselekvés *Ott(Pótkerék, Föld)* előfeltételét. Válasszuk az *Eltávolít(Pótkerék, Csomagtartó)* cselekvést, ami az egyetlen alkalmazható cselekvés az előfeltétel biztosításához. A végső terv a 11.8. ábrán látható.



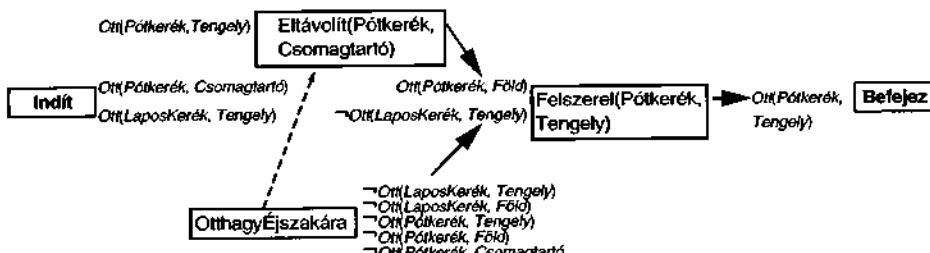
**11.8. ábra.** A nem teljes részben rendezett terv a keréksere problémához, miután megválasztottuk a cselekvéseket az első két nyitott előfeltételhez. A dobozok cselekvéseket tesztelnek meg, a bal oldalon az előfeltételekkel, a jobb oldalon pedig a következményekkel. (Az *Indít* cselekvést kivéve a következményeket elhagytuk.) A fekete nyílak az okozati kapcsolatokat mutatják, melyek a nyíl fejénél szereplő állást védik.

3. Vegyük a *Felszerel(Pótkerék, Tengely)* cselekvés  $\neg\text{Ott}(\text{LaposKerék}, \text{Tengely})$  előfeltételét. Hogy ellenkezzünk egy kicsit, válasszuk az *OthagyÉjszakára* cselekvést, a kézenfekvő *Eltávolít(LaposKerék, Tengely)* helyett. Vegyük észre, hogy az *OthagyÉjszakára* cselekvésnek szintén következménye a  $\neg\text{Ott}(\text{Pótkerék}, \text{Föld})$ , ami ütközik az



okozati kapcsolattal. Hogy feloldjuk ezt az ütközést, egy rendezési megkötést illesztünk be, ami az *OthagyÉjszakára* cselekvést az *Eltávolít(Pótkerék, Csomagtartó)* elő rendeli. Az így előálló terv a 11.9. ábrán látható. (Miért oldja ez fel az ütközést és miért nincs erre más lehetőség?)

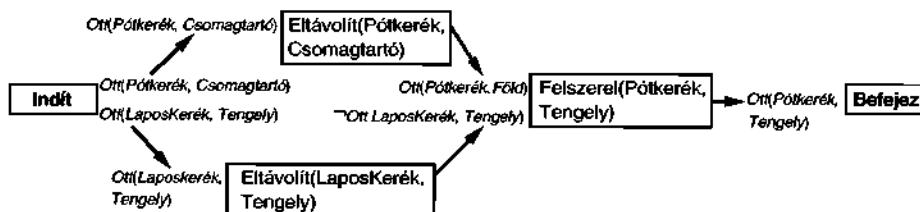
4. Most az egyetlen fennmaradó nyitott előfeltétel az *Ott(Pótkerék, Csomagtartó)*, ami az *Eltávolít(Pótkerék, Csomagtartó)* cselekvés előfeltétele. Az egyetlen cselekvés, ami ezt elérheti, a meglévő *Indít* cselekvés, de az *Indít* és az *Eltávolít(Pótkerék, Csomagtartó)* közötti okozati kapcsolat ütközik az *OthagyÉjszakára* cselekvés  $\neg\text{Ott}(\text{Pótkerék}, \text{Csonagtartó})$  következményével. Most nincs mód arra, hogy feloldjuk az ütközést az *OthagyÉjszakára* cselekvéssel: nem sorolhatjuk az *Indít* elő (mert semmi nem kerülhet az *Indít* elő), és nem rendelhető az *Eltávolít(Pótkerék, Csonagtartó)* után (mert már van egy rendezés, ami elő helyezi). Ezért muszáj visszalépnünk, törölünk az *Eltávolít(Pótkerék, Csonagtartó)* cselekvést és a két utolsó okozati kapcsolatot, és visszalépnünk a 11.8. ábrán bemutatott állapothoz.



**11.9. ábra.** A terv alakulása, az *OthagyÉjszakára* választása után, amit a  $\neg\text{Ott}(\text{LaposKerék}, \text{Tengely})$  előréshoz választottunk. Hogy elkerüljük az ütközést az *Ott(Pótkerék, Csonagtartó)*-t védő, az *Eltávolít(Pótkerék, Csonagtartó)*-ból induló okozati kapcsolattal, az *OthagyÉjszakára* cselekvést egy rendezési megkötéssel az *Eltávolít(Pótkerék, Csonagtartó)* elő helyeztük, amit a szaggatott nyíllal jelöltünk.

Összefoglalva, a tervkészítő bebizonyította, hogy az *OtthagyÉjszakára* nem használható fel kerékcserére.

5. Vegyük újra a *Felszerel(Pótkerék, Tengely)* cselekvés  $\neg\text{Ott}(\text{LaposKerék}, \text{Tengely})$  előfeltételét. Ez alkalommal válasszuk az *Eltávolít(LaposKerék, Tengely)* cselekvést.
6. Vegyük újra az *Eltávolít(Pótkerék, Csomagtartó)* cselekvés *Ott(Pótkerék, Csomagtartó)* előfeltételét, és válasszuk az *Indít* cselekvést, hogy ezt elérjük. Ez alkalommal nincsenek ütközések.
7. Vegyük az *Eltávolít(LaposKerék, Tengely)* cselekvés *Ott(LaposKerék, Tengely)* előfeltételét, valamint ennek eléréséhez az *Indít* cselekvést. Ez egy teljes és konzisztens tervet ad, vagy más szavakkal egy megoldást, amint azt a 11.10. ábra mutatja.



**11.10. ábra.** A kerékcsera probléma végső megoldása. Vegyük észre, hogy az *Eltávolít(Pótkerék, Csonagtartó)* és az *Eltávolít(LaposKerék, Tengely)* tetszőleges sorrendben végrehajtható a *Felszerel(Pótkerék, Tengely)* előtt.

Bár ez a példa nagyon egyszerű, jól szemlélteti a részben rendezett tervkészítés erősségeit. Először is az okozati kapcsolatok a keresési tér (fa) egy korai szűkítéséhez meteszéséhez vezettek, mert kizártak olyan területeket, amelyek a feloldhatatlan ütközések miatt nem tartalmaznak megoldásokat. Másodszor a 11.10. ábra megoldása egy részben rendezett terv. Ebben az esetben ennek előnye nem nagy, mert csak két lehetséges sorba rendezés létezik, mégis egy ágens számára előnyös lehet ez a rugalmasság, például ha a kereket sűrű forgalomban kellene lecserélni.

A példa szintén rámutat néhány javítási lehetőségre. Például egy próbálkozás duplán történt: az *Indít* az *Eltávolít(Pótkerék, Csonagtartó)* cselekvéshez kapcsolódik még mielőtt az ütközés visszalépést váltana ki, és a visszalépés hatására a kapcsolat felborulik, bár nem vesz részt az ütközésben. A keresés folytatásakor a kapcsolat újra létrejön. Ez tipikus az időrendi visszalépés esetén és elkerülhető, ha függőségevezérelt visszalépést használunk.

## Részben rendezett tervkészítés kötetlen változókkal

Ebben a fejezetben a változókat tartalmazó elsőrendű cselekvés reprezentációra alkalmazott részben rendezett tervkészítés során felmerülő komplikációkkal foglalkozunk. Tegyük fel, hogy a kockavilág (lásd 11.4. ábra) problémát kell megoldani a *Rajta(A, B)* nyitott előfeltétellel és a

*Cselekvés(Mozgat(b, x, y),*

**ELŐFELTÉTEL:** *Rajta(b, x)  $\wedge$  Üres(x)  $\wedge$  Üres(y)*

**KÖVETKEZMÉNY:** *Rajta(b, y)  $\wedge$  Üres(x)  $\wedge$   $\neg$ Rajta(b, x)  $\wedge$   $\neg$ Üres(y))*

cselekvéssel. Ez a cselekvés teljesíti a  $Rajta(A, B)$  literált, mert a  $Rajta(b, u)$  következmény egyesíthető a  $Rajta(A, B)$ -vel a  $\{b/A, y/B\}$  behelyettesítéssel. Alkalmazzuk tehát ezt a behelyettesítést a cselekvésre, aminek az eredménye a

*Cselekvés(Mozgat(A, x, B),*

*ELŐFELTÉTEL: Rajta(A, x)  $\wedge$  Üres(x)  $\wedge$  Üres(B)*

*KÖVETKEZMÉNY: Rajta(A, B)  $\wedge$  Üres(x)  $\wedge$   $\neg$ Rajta(b, x)  $\wedge$   $\neg$ Üres(B))*

Ez az  $x$  változót kötetlenül hagyja. Ez annyit tesz, hogy a cselekvés annyit mond, hogy mozgasd A-t valahonnan, de azt nem, hogy honnét. Ez egy másik példa a legkisebb megkötés elvre: elhalaszthatjuk a döntéseket egészen addig, amíg a terv egy másik lépése ezt meghozza nekünk. Tegyük fel például, hogy a  $Rajta(A, D)$  a kiinduló állapotunk. Az  $x$ -et  $D$ -vel behelyettesítve az *Indít* cselekvés használható fel, hogy elérjük a  $Rajta(A, x)$ -et. A módszer, hogy több információt kívárunk, mielőtt megválasztjuk  $x$ -et, gyakran sokkal hatékonyabb, mint az  $x$  minden értékének kipróbálása és a visszalépés. ha ez sikertelen.

A változók jelenléte az előfeltételekben és a cselekvésekben nehezíti az ütközések detektálását és feloldását. Például amikor a  $Mozgat(A, x, B)$  a terv része lesz, a

$Mozgat(A, x, B) \xrightarrow{\text{Rajta}(A, B)} \text{Befejez}$

okozati kapcsolatra is szükség van. Ha van egy  $M_2$  cselekvés  $\neg$ Rajta(A, z) következménnyel, akkor ez csak akkor okoz ütközést, ha a z értéke B. Hogy kezeljük ezt a lehetőséget, kiterjesztjük a tervezek reprezentációját, hogy tartalmazzanak  $z \neq X$  alakú **egyenlőtlenségi kényszereket (inequality constraints)**, ahol z egy változó és X egy változó vagy egy konstansszimbólum. Ebben az esetben az ütközést a  $z \neq B$  hozzáadásával oldhatjuk fel, ami azt jelenti hogy a terv később B kivételével bármire behelyettesítheti z-t. Bármikor amikor egy behelyettesítést hajtunk végre a tervben, ellenőriznünk kell, hogy az nem ellenkezik-e az egyenlőtlenségi korlátokkal. Például egy  $x/y$  behelyettesítés ütközik az  $x \neq y$  kényszerrel. Az ilyen ellentétek nem oldhatók fel, ezért a terv-készítő visszalépéstre kényszerül.

A 12.6. alfejezetben egy bővebb példát adunk a részben rendezett tervkészítésre, ami a változókat tartalmazó kockavilág problémára.

## Heurisztikák a részben rendezett tervkészítésre

A teljesen rendezett tervkészítéssel összehasonlítva a részben rendezett tervkészítés egyértelmű előnye, hogy a probléma részfeladatokra bontható. A hátránya, hogy nem reprezentálja megfelelően az állapotokat, így nehezebb megbecsülni, hogy a részben rendezett terv milyen messze van a céltól. Jelenleg jóval kevesebb az ismeretanyag arról, hogy hogyan adható pontos heurisztika a részben rendezett tervkészítéshez, mint a teljesen rendezett tervkészítések esetén.

A legkézenfekvőbb heurisztika, hogy megszámoljuk az eltérő nyitott előfeltételeket. Ez javítható azáltal, ha kivonjuk azon nyitott előfeltételek számát, melyek illeszkednek az *Indít* állapot egy literáljaival. Hasonlóan a teljesen rendezett esethez ez túlbecsüli a költséget, ha vannak cselekvések, melyek több célt érnek el, és alulbecsül, ha negatív kölcsönhatások vannak a terv lépései között. A következő alfejezet egy olyan

megközelítést mutat be, ami lehetővé teszi, hogy sokkal pontosabb heurisztikákat kapunk egy relaxált problémából.

A heurisztikus függvényt arra használjuk, hogy kiválasszuk a tovább finomítandó tervet. Ezzel a választással az algoritmus követő állapotokat generál egyetlen kiválasztott nyitott előfeltételből kiindulva. Mint a változókiválasztás a kényszerkielégítési algoritmusban, ez a választás nagy hatással van a hatékonyságra. A kényszerkielégítési probléma **legjobban-korlátozott-változó** (*most-constrained-variable*) heurisztikája alkalmazható a tervkészítő algoritmusokra, és látszólag jól működik. Az alapötlet, hogy válasszuk azt a nyitott feltételt, mely a *lehető legkevesebb* módon elégíthető ki. Ennek a heurisztikának két speciális esete van. Először, ha egy nyitott feltétel nem érhető el egyetlen cselekvéssel sem, akkor a heurisztika kiválasztja. Ez jó ötlet, hiszen a teljesítetlenség korai érzékelése sok munkát megspórolhat. Másodszor, ha egy nyitott feltétel csak egyféléképp teljesíthető, akkor célszerű kiválasztani, hiszen ez egy elkerülhetetlen döntés, ami további megkötéseket eredményezhet a későbbi választásokhoz. Habár minden nyitott feltételhez meglehetősen kölcsönös kiszámolni, hogy azok hányszámonképpen teljesíthetők, és ez nem is minden éri meg, a kísérletek azt mutatják, hogy a két speciális eset kezelése jelentős gyorsulást eredményezhet.

## 11.4. TERVKÉSZÍTÉSI GRÁFOK

Az összes javasolt heurisztika a teljesen rendezett és a részben rendezett tervkészítésre pontatlanságokkal terhelt. Ez a fejezet bemutatja, hogy egy speciális adatszerkezet a tervkészítési gráf (*planning graph*) felhasználható, hogy jobb heurisztikus becsléseket nyerjünk. Ezek a heurisztikák bármely eddig tárgyalt keresési technikával használhatók. Egy másik lehetőség, hogy a megoldást közvetlenül a tervkészítési gráfból nyerjük ki egy erre kiélezett algoritmussal, mint amilyen például a GRAPHPLAN.

A tervkészítési gráf a terv időrendi lépéseinek megfelelő szintekből áll, ahol a 0-dik szint a kiinduló állapot. minden szín egy literálhalmazt és egy cselekvéshalmazt tartalmaz. A literálok durván azok, melyek igazak *lehetnek* az adott lépésnél, az addig végrehajtott cselekvések függvényében. Ismét durván fogalmazva, a cselekvések azok, melyeknek az előfeltételei teljesülhetnek az adott időlépésben attól függően, hogy aktuálisan mely literálok teljesülnek. Azért mondjuk, hogy „durván”, mert a tervkészítési gráf csak a cselekvések közötti lehetséges negatív kölcsönhatások egy korlátozott részhalmazát tartalmazza, ezért a tervkészítési gráf optimista lehet a literál teljesítéséhez szükséges minimális lépések számával kapcsolatban. Mindazonáltal a tervkészítési gráfban ez a lépésszám egy jó becslés arra, hogy milyen nehéz egy literált teljesíteni a kiindulási állapotból. Még fontosabb, hogy a tervkészítési gráfot úgy definiáltuk, hogy könnyen elkészíthető legyen.

A tervkészítési gráf csak ítéletlogikai tervkészítési problémára alkalmazható – olyanakra, melyek nem tartalmaznak változókat. Ahogy a 11.1. alfejezetben említettük, mind a STRIPS, mind az ADL reprezentáció ítéletlogikára alakítható. A nagyszámú entitást tartalmazó problémákat ez jelentősen felduzzasztja a cselekvéssémák számában. Ennek ellenére a tervkészítési gráfok hatékony eszközöknek bizonyultak a nehéz tervkészítési problémák megoldásában.

A tervkészítési gráfokat egy egyszerű példán mutatjuk be. (Az összetettebb feladatok olyan gráfhoz vezetnek, amelyek nem férnének ki egy oldalra.) A 11.11. ábra a feladatot,

míg a 11.12. ábra az ehhez tartozó gráfot mutatja. Az  $S_0$  állapotszintről indulunk, ami a probléma kiinduló állapotának felel meg. Ezt az  $A_0$  cselekvésszinttel folytatjuk, amibe azokat a cselekvéseket helyezzük, melyek előfeltételei az előző szinten teljesülnek. minden cselekvés össze van kötve az  $S_0$ -ban található előfeltételeivel, illetve az  $S_1$ -ben található következményeivel, ami ebben az esetben az  $S_0$ -ban nem szereplő literálok bevezetését jelenti  $S_1$ -ben.

```

Kiindulás(Van(Süti))
Cél(Van(Süti) ∧ Megegett(Süti))
Cselekvés(Eszik(Süti))
ELŐFELTÉTEL: Van(Süti)
KÖVETKEZMÉNY: ¬Van(Süti) ∧ Megegett(Süti))
Cselekvés(Süt(Süti))
ELŐFELTÉTEL: ¬Van(Süti)
KÖVETKEZMÉNY: Van(Süti)

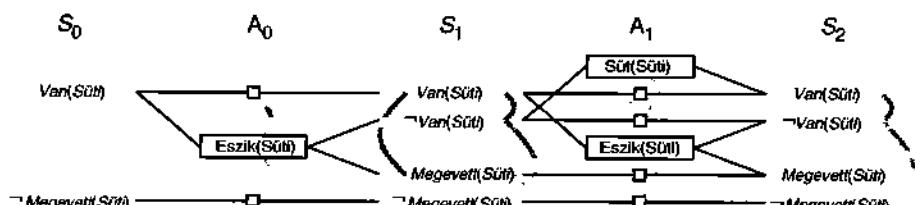
```

11.11. ábra. A „legyen süti és együnk is” probléma

A tervkészítési gráfnak a nem cselekvést ugyanúgy kell reprezentálni, mint a cselekvéseket. Ez annyit tesz, hogy a szituációkalkulus keretaxiomáinak megfelelő működésre van szükség, ami alapján egy literál két állapon keresztül igaz maradhat, ha nincs ezt módosító cselekvés. Egy tervkészítési gráfban ezt megőrző cselekvésekkel (**persistence actions**) oldjuk meg. minden pozitív és negatív C literálhoz egy megőrző cselekvést szűrünk be C előfeltétellel és C következménnyel. A 11.12. ábra az  $A_0$  szinten egyetlen „valós” cselekvést az *Eszik(Süti)*-t tartalmazza, két megőrző cselekvéssel, amit kicsi négyzetek jelölnek.

Az  $A_0$  tartalmazza az összes cselekvést, ami az  $S_0$  állapotban előfordulhat, de ugyanilyen fontos, hogy rögzíti a cselekvések közötti ütközéseket is, amelyek megakadályozzák, hogy egyszerre történjenek. A 11.12. ábra szürke vonalai ezeket a kölcsönös kizárási (**mutual exclusions** vagy **mutex**) kapcsolatokat jelölik. Például az *Enni(Süti)* kölcsönös kizárássban van a *Van(Süti)* vagy a *¬Megegett(Süti)* megőrzésével. Hamarosan látjuk, hogy a mutex kapcsolatokat hogyan számíthatjuk.

Az  $S_1$  az összes olyan literált tartalmazza, ami az  $A_0$  szint akcióinak bármely részhalmazát választva elérhető. Szintén tartalmaz mutex kapcsolatokat (szürke vonalakat)



11.12. ábra. A „legyen süti és együnk is” probléma tervkészítési gráfja az  $S_2$  szintig. A téglalapok az akciókat jelentik (a kicsi négyzetek a megőrző cselekvések), az egyenesek az előfeltételeket és a következményeket jelölik. A kölcsönös kizárásokat félvétű vonalak jelölik.

jelölve az olyan literálokat, melyek nem teljesülhetnek egyszerre, a választott cselekvésekkel függetlenül. Például a *Van(Süti)* és a *Megevett(Süti)* kölcsönösen kizáráják egymást. Az  $A_0$  kiválasztott cselekvéseitől függően az eredmény vagy az egyik vagy a másik lehet, de a kettő egyszerre nem. Más szavakkal az  $S_1$ , csakúgy, mint a regressziós állapottér-keresés, több állapotot reprezentál, a kizárási kapcsolatok pedig kényszerék, melyek a lehetséges állapotok halmazát definiálják.

És ez így megy tovább. Az  $S_i$  állapot- és az  $A_i$  cselekvésszintek között mozgunk, egészen addig, míg elérünk egy szintet, ahol két egymást követő szint azonos. Ekkor azt mondjuk, hogy a gráf **kiegyenlítődött (leveled off)**. minden egymást követő szint azonos, így további kiterjesztésre nincsen szükség.

Egy olyan struktúrát kaptunk, ahol az  $A_i$  szint tartalmaz minden  $S_j$ -ben alkalmazható cselekvést, a kényszerekkel együtt, melyek megmondják, hogy melyik cselekvéspár nem hajtható végre egyidejűleg. minden  $S_i$  szint tartalmazza az összes literált, ami az  $A_{i-1}$  összes lehetséges cselekvéshalmazának hatására teljesül, a kényszerekkel együtt, amelyek a nem lehetséges literálpárokat jelölik ki. Fontos megjegyezni, hogy a tervkészítési gráf elkészítési folyamatához *nem* kell választanunk a cselekvések között, ami kombinatorikus keresést eredményezne. Ezzel szemben, a tervkészítési gráf konstrukciója csak a lehetetlen választásokat rögzíti, mutex kapcsolatok felhasználásával. Egy ilyen tervkészítési gráf elkészítésének bonyolultsága egy alacsonyrendű polinomiális komplexitás a cselekvések számát tekintve, ahol az állapottér exponenciális a literálok számában.

Most mutex kapcsolatokat definiálunk, mind a cselekvések, mind a literálok számára. Egy adott szinten, a mutex kapcsolat akkor érvényes két cselekvés között, ha a következő három feltétel valamelyike fennáll:

- *Inkonzisztens hatások*: egy cselekvés negálja egy másik következményét. Például az *Eszik(Süti)* és a *Van(Süti)* megőrzése inkonzisztens hatású, mert a *Van(Süti)* következmény tekintetében ellentétesek.
- *Interferencia*: egy cselekvés egyik következménye a negálása egy másik cselekvés előfeltételének. Például az *Eszik(Süti)* interferál a *Van(Süti)* megőrzésével, mert negálja az előfeltételét.
- *Versenyhelyzet*: egy cselekvés előfeltétele kölcsönösen kizárá egy másik előfeltételelét. Például a *Süt(Süti)* és az *Eszik(Süti)* mutexek, mert versenyeznek a *Van(Süti)* előfeltétel tekintetében.

Két azonos szineten lévő literál között mutex kapcsolat áll fenn, ha az egyik a másik negálja, vagy bármely lehetséges cselekvéspár, amely a két literált elérheti, egymást kölcsönösen kizárá. Ezt az állapotot *inkonzisztens háttérnek* nevezzük. Például a *Van(Süti)* és a *Megevett(Süti)* kizáráják egymást  $S_1$ -ben, mert az egyetlen mód a *Van(Süti)* teljesítésére egy megőrző cselekvés, ami mutex a *Megevett(Süti)* egyetlen elérési lehetőségével, nevezetesen az *Eszik(Süti)* cselekvéssel. Az  $S_2$  szinten a két literál már nem mutex, mert új elérési lehetőségek vannak hozzájuk, mint a *Süt(Süti)* és a *Megevett(Süti)* megőrző cselekvése, melyek nem zájják ki egymást.

## Tervkészítési gráfok heurisztikus becslésekre

A tervkészítési gráf, ha egyszer már létrehoztuk, nagyon gazdag információforrás a problémáról. Például *egy literál, ami nem található meg az utolsó szinten, egyetlen tervvel sem érhető el*. Ez a megfigyelés a visszafelé keresésben használható a következőképp: bármely elérhetetlen literált tartalmazó állapot költsége  $h(n) = \infty$ . Hasonlóan a részben rendezett tervkészítésnél bármely tervnek a költsége egy elérhetetlen nyitott előfeltétellel,  $h(n) = \infty$ .

Ez az ötlet tovább általánosítható. minden cél literál elérési költségét becsülhetjük azzal a szinttel, ahol először megjelenik a tervkészítési gráfban. Ezt a cél szint költségének (**level cost**) nevezzük. A 11.12. ábrán a *Van(Süti)* szint költsége 0, a *Megevett(Süti)* szint költsége 1. Könnyű megmutatni (lásd 11.9. feladat), hogy ezek a közelítések elfogadhatók az egyedülálló célokra. A becslés azonban lehet, hogy nem túl jó, mert a tervkészítési gráfok több cselekvést megengednek szintenként, míg a heurisztika csak a szintek számát tartalmazza, és nem a cselekvések számát. Ebből kifolyólag a heurisztikák számításában, gyakori a **soros tervkészítési gráf** (*serial planning graph*) használata. A soros gráf szerint egy adott időpillanatban csak egyetlen cselekvés hajtható végre, ami muterekkel érhető el úgy, hogy a megőrző cselekvések kivételével, minden cselekvés közé mutex kapcsolatot teszünk. A soros tervkészítő gráfkból kinyert szintköltségek gyakran egész elfogadható közelítését adják a valós költségeknek.

Három egyszerű megközelítés létezik, hogy célliterálok konjunkciójához költséget becsüljünk. A **maximális szint** (**max-level**) heurisztika egyszerűen a célok közötti maximális szintköltségét veszi, ami elfogadható, de nem feltétlen nagyon pontos. A **szintösszeg** (**level sum**) heurisztika a részcel függetlenségi feltételezésből a célok szintköltségeinek összegét adja. Ez nem elfogadható, de nagyon jól működik olyan gyakorlati problémákra, amelyek nagymértékben részekre bonthatók. Ez sokkal pontosabb, mint a 11.2. alfejezetben bemutatott nem-kielégített-célok-heurisztika. A mi feladatunkban a *Van(Süti)  $\wedge$  Megevett(Süti)* konjunktív célohoz rendelhető heurisztikus becslés  $0 + 1 = 1$ , míg a helyes válasz 2. Mindezeken túl, ha eltávolítjuk a *Süt(Süti)* cselekvést, a becslés még mindig 1, de a konjunktív cél ekkor már elérhetetlen. Végezetül, a **halmazszint** (**set-level**) heurisztika azt a szintet keresi meg, ahol a konjunktív cél összes literálja megjelenik a tervkészítési gráfban, és nincsenek közöttük kölesönösen kizárt párok. A heurisztika a korrekt 2 értéket adja az eredeti feladatra és végtelent a *Süt(Süti)* nélküli feladatra. Ez előnyben van a maximális szint heurisztikával szemben, és kimágaslóan jól működik azon feladatokban, ahol sok kölcsönhatás van a részterek között.

Lévén egy pontos heurisztika készítésére szolgáló eszköz, a tervkészítési gráf tekintető úgy, mint egy relaxált probléma, ami hatékonyan megoldható. Hogy a relaxált probléma természetét megértsük, pontosan meg kell értenünk, hogy mit jelent, ha egy  $g$  literál megjelenik a tervkészítési gráf  $S_i$  szintjén. Ideális esetben garanciát szeretnénk arra, hogy létezik egy terv  $i$  cselekvés szinttel, ami eléri  $g$ -t, illetve ha  $g$  nem jelenik meg, akkor nincs is ilyen terv. Sajnos ezt garantálni majdnem ugyanolyan nehéz, mint megoldani az eredeti tervkészítési problémát. A tervkészítési gráf a garancia második részéről gondoskodik (ha  $g$  nem jelenik meg, akkor nincs terv), de ha  $g$  megjelenik, akkor a tervkészítési gráf csak annyit ígér, hogy van egy terv, ami *várhatóan* eléri  $g$ -t, nincsenek „nyilvánvaló” hibák. Egy nyilvánvaló hiba definíció szerint egy olyan hiba, ami úgy detektálható, hogy két cselekvést vagy két literált tekintünk egyszerre, vagy más



szavakkal a mutex relációkat vizsgáljuk. Lehetnek összetettebb hibák, melyek három, négy vagy több cselekvést tartalmaznak, de a tapasztalatok azt mutatják, hogy nem éri meg ezekkel foglalkozni. Ez hasonló a kényszerkielégítési problémáknál tanultakkal, hogy a megoldás megkeresése előtt gyakran megéri a 2-konzisztencia számítás, de a 3- vagy a magasabb konzisztencia kiszámítása ritkábban kifizetődő (lásd 5.2. alfejezet).

## A GRAPHPLAN algoritmus

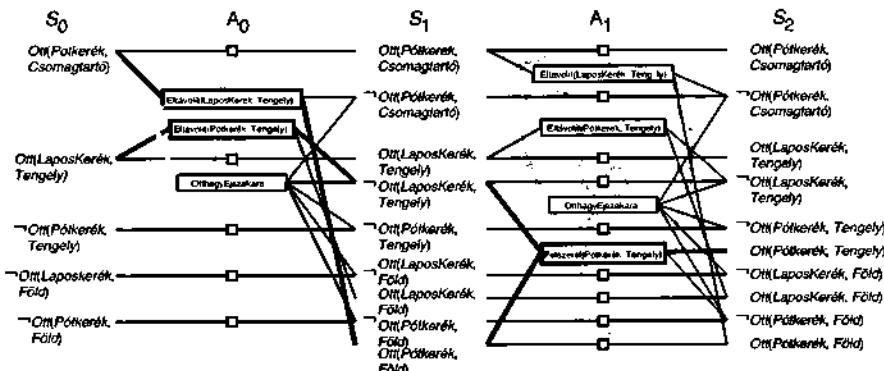
Ez a fejezet bemutatja, hogy hogyan nyerhető ki közvetlenül a terv a tervkészítési gráf-ból, ahelyett hogy azt csak a heurisztikákhoz használjuk. A GRAPHPLAN algoritmusnak (11.13. ábra) két fő lépése van, melyek ciklikusan váltakoznak. Először ellenőrzi, hogy minden célliterát jelen van-e az adott szinten anélkül, hogy mutex kapcsolatok lennének közöttük. Ebben az esetben, az aktuális gráfban létezhet megoldás, így az algoritmus megpróbálja azt kinyerni. Ellenkező esetben bővíti a gráfot úgy, hogy az aktuális szint-hez a cselekvéseket, a következőhöz az állapotliterálokat adjá: ez a folyamat folytatódik addig, amíg megtaláljuk a megoldást, vagy bebizonyosodik, hogy nem létezik megoldás.

```
function GRAPHPLAN(probléma) returns megoldás vagy kudarc
    gráf ← KIINDULÓ-TERVKÉSZÍTÉSI-GRÁF(probléma)
    célok ← CÉLOK[probléma]
    loop do
        if célok minden mutex az utolsó szintjén a gráf-nak then do
            megoldás ← MEGOLDÁS-KINYERÉS(gráf, célok, Hossz(gráf))
            if megoldás ≠ kudarc then return megoldás
            else if NINCS-LEHETSÉGES-MEGOLDÁS(gráf) then return kudarc
        gráf ← GRÁF-BŐVÍTÉS(gráf, probléma)
```

**11.13. ábra.** A GRAPHPLAN algoritmus. A GRAPHPLAN algoritmus egy megoldás kinyerő és egy gráf-bővítő lépés között alternál. A MEGOLDÁS-KINYERÉS a befejezéstől visszafelé keresve ellenőrzi, hogy található-e megoldás. A GRÁF-BŐVÍTÉS adja a cselekvéseket az adott szinthez és az állapotliterálokat a következőhöz.

Kövessük végig a GRAPHPLAN működését a 11.1. alfejezet kerékcseré problémáján! A teljes gráf a 11.14. ábrán látható. A GRAPHPLAN első lépésben inicializálja a tervkészítési gráfot egy egyszintű ( $S_0$  szint) gráfra, ami a kiindulási állapot öt literálját tartalmazza. Az *Ott*(Pótkerék, Tengely) célliterál nincs jelen az  $S_0$ -ban, ezért nem kell meghívunk a MEGOLDÁS-KINYERÉST, hiszen biztosak vagyunk benne, hogy még nincs megoldás. Ehelyett a GRÁF-BŐVÍTÉS hozzáad három cselekvést, melyeknek az előfeltétele már teljesül az  $S_0$  szinten (például az összes cselekvés, kivéve a *Felszerel*(Pótkerék, Tengely) – cselekvést). Hozzáadja továbbá a megőrző-cselekvésetet  $S_0$  összes literáljához. A cselekvések következményeit az  $S_1$  szinthez adjuk hozzá. A GRÁF-BŐVÍTÉS ezek után mutex kapcsolatokat keres, és hozzáadja ezeket a gráphoz.

Az *Ott*(Pótkerék, Tengely) cselekvés még mindig nincs jelen az  $S_1$  állapotban, ezért most sem hívjuk meg a MEGOLDÁS-KINYERÉS lépést. A GRÁF-BŐVÍTÉS hívással a 11.14.



**11.14. ábra.** A kerékszerre probléma tervkészítési gráfja az S<sub>2</sub> szintre bővítés után. A mutex kapcsolatokat szürke vonalak jelölik. Csak néhány fontos mutexet mutatunk, mert az ábra olvashatatlantá válna, ha az összeset jelölnénk. A megoldást megvastagított vonalak és kiemelések jelölik.

ábrán látható tervkészítési gráfot kapjuk. Most, hogy a cselekvések teljes skálája rendelkezésre áll, ideje néhány példát nézni a mutex kapcsolatokra és a hatásaiakra:

- **Nem konzisztens hatások:** az *Eltávolít(Pótkerék, Csomagtartó)* mutex kapcsolatban van az *OttagyÉjszakára* cselekvéssel, mert az egyik következménye az *Ott(Pótkerék, Föld)*, míg a másiké ennek negáltja.
- **Interferencia:** az *Eltávolít(Pótkerék, Tengely)* mutex az *OttagyÉjszakára* cselekvéssel, mert az egyiknek előfeltétele az *Ott(Laposkerék, Tengely)*, míg a másiknak ennek negáltja a következménye.
- **Versenyhelyzet:** a *Felszerel(Pótkerék, Tengely)* mutex az *Eltávolít(Laposkerék, Tengely)* cselekvéssel, mert az egyiknek előfeltétele az *Ott(Laposkerék, Tengely)*, míg a másiké ennek negáltja.
- **Inkonzisztens tartó:** az *Ott(Pótkerék, Tengely)* és az *Ott(Laposkerék, Tengely)* kölcsönösen kizárók az S<sub>2</sub>-ben, mert az *Ott(Pótkerék, Tengely)* csak a *Felszerel(Pótkerék, Tengely)* cselekvéssel érhető el, ami mutex a megőrző cselekvéssel, mely az *Ott(Laposkerék, Tengely)* egyetlen elérése. A mutex kapcsolatok így azonnal érzékelik a konfliktust, ami abból adódik, hogy két objektumot ugyanazon időpontban azonos helyre próbálunk tenni.

Ez alkalommal visszamegyünk a ciklus kezdetére. A cél minden literálja jelen van S<sub>2</sub>-ben, és egyik sem áll kölcsönös kizárában egyetlen másikkal sem. Ez azt jelenti, hogy egy megoldás létezhet, és a MEGOLDÁS-KINYERÉS megtalálhatja. Lényegében a MEGOLDÁS-KINYERÉS egy kéttétekű kényszerkielégítési problémát old meg, melynek változói a szintek cselekvései, és az értékeik pedig azt jelzik, hogy *benne vannak-e* vagy *nincsenek benne* a tervben. Ehhez egy egyszerű kényszerkielégítési algoritmust használhatunk, vagy definiálhatjuk a MEGOLDÁS-KINYERÉS-t mint egy keresési problémát. Itt a keresés minden állapota a kielégítetlen célok egy halmazát tartalmazza, valamint egy mutatót a tervkészítési gráf egy szintjére. Ezt a keresési problémát a következőképpen definíáljuk:

- A kiindulási állapot a tervkészítési gráf utolsó szintje (S<sub>n</sub>) a tervkészítési probléma céljaival egyetemben.

- Az  $S_i$  szint állapotában rendelkezésre álló cselekvések összessége, az  $A_{i-1}$  cselekvéseinak azon konfliktusmentes részhalmaza, melyek következményei elérik az állapotban lévő célokat. Az eredményként előálló állapot szintje  $S_{i-1}$ , céljai pedig a kiválasztott cselekvések előfeltételei. A „konfliktusmentességen” olyan cselekvések halmazát értjük, amelyek között nincs kettő, amelyek kölcsönösen kizárnák egymást, és az előfeltételeik között sem szerepelnek kölcsönösen kizáró párok.
- A cél, hogy elérjünk az  $S_0$  szinten egy állapotot úgy, hogy minden cél teljesüljön.
- minden cselekvés költsége 1.

Ehhez a problémához az  $S_2$  szintről indulunk, az *Ott(Pótkerék, Tengely)* céllal. Az egyetlen választásunk, hogy elérjük ezt a célt, a *Felszerel(Pótkerék, Tengely)* cselekvés. Ez az  $S_1$  keresési állapothoz visz mincket, melynek céljai az *Ott(Pótkerék, Föld)* és az  $\neg$ *Ott(Laposkerék, Tengely)*. Az előbbit az *Eltávolít(Pótkerék, Csomagtartó)* cselekvéssel érhetjük el, míg a későbbit az *Eltávolít(Laposkerék, Tengely)* vagy az *Ott(hagyÉjszakára* cselekvések egyikével. Az *Ott(hagyÉjszakára* kölcsönösen kizáró kapcsolatban van a *Eltávolít(Pótkerék, Csomagtartó)*-val, ezért az egyetlen megoldás, hogy az *Eltávolít(Pótkerék, Csomagtartó)* és az *Eltávolít(Laposkerék, Tengely)* cselekvéseket választjuk. Ez az  $S_0$  keresési állapotra vezet, az *Ott(Pótkerék, Csomagtartó)* és az *Ott(Laposkerék, Tengely)* célokkal. Mindkettő jelen van az állapotban, így megvan a megoldásunk: az *Eltávolít(Pótkerék, Csomagtartó)* cselekvés és az *Eltávolít(Laposkerék, Tengely)* az  $A_0$  szinten, melyet az  $A_1$ -ben a *Felszerel(Pótkerék, Tengely)* követ.

Tudjuk, hogy a tervkészítés polinomiális helyigényű, és hogy a tervkészítési gráf elkészítése polinomiális idejű, ezért tudjuk, hogy a megoldás kinyerése legrosszabb esetben kezelhetetlen. Ebből kifolyólag valamifajta heurisztika segítségére lesz szükségünk, hogy a cselekvések közül válasszunk a visszafelé keresés során. A gyakorlatban jól működik egy mohó algoritmus, mely a literálok között a szintköltség alapján választ. Bárminely célhalmazra a következő sorrendben járunk el:

1. Elsőként a legmagasabb szintköltségű literált választjuk.
2. Hogy teljesítstük ezt a literált, válasszuk először a legegyzszerűbb előfeltételekkel rendelkező cselekvést, azaz válasszuk azt az akciót, melynek előfeltételeire a szintköltségek összege (vagy maximuma) a legkisebb!

## A GRAPHPLAN algoritmus leállása

Eddig átsiklottunk a leállás kérdése fölött. Lehetünk-e biztosak abban, hogy amennyiben egy problémának nincs megoldása, a GRAPHPLAN algoritmus nem kerül végtelen ciklusba, iterációként bővíve a tervkészítési gráfot? A válasz igen, de ennek bizonyítása túlmutat ennek a könyvnek a keretein. Itt csak a fő ötleteket körvonalazzuk, különösen azokat, melyek a tervkészítési gráfok általános tulajdonságaira világítanak rá.

Az első lépés, hogy észrevegyük, hogy a tervkészítési gráfok bizonyos tulajdonságai monoton növekvők vagy csökkenők. „X monoton növekvő” azt jelenti, hogy az X-ek halmaza az  $i$ -edik szinten az  $i + 1$ -edik szint halmazának (nem feltétlenül valódi) részhalmaza. A tulajdonságok a következők:

- *Monoton növekvő literálok.* Ha egy literál megjelenik egy adott szinten, akkor az ezt követő összes szinten megjelenik. Ez a megőrző cselekvések miatt van, azaz, ha egy literál megjelenik, a megőrző cselekvések hatására örökké megmarad.
- *Monoton növekvő cselekvések.* Ha egy cselekvés megjelenik egy adott szinten, akkor az ezt követő összes szinten megjelenik. Ez a literálok növekedésének a következménye, azaz, ha egy cselekvés előfeltételei megjelennek egy szinten, akkor az összes ezt követő szinten rendelkezésre állnak, így a cselekvés is.
- *A kölcsönös kizárások monoton csökkenése.* Ha két cselekvés kölcsönösen kizárja egymást az  $A_i$  szinten, akkor az összes ezt megelőző szinten, ahol mindenki megjelenik, szintén kölcsönösen kizárák egymást. Ugyanez igaz a literálok közötti kölcsönös kizárásokra. Ez az ábrákon nem mindig látható, mert ezek a következő egyszerűsítést használják: nem ábrázolják sem azokat a literálokat, melyek egy adott  $S_i$  szinten nem teljesülnek, sem azokat a cselekvéseket, amelyek egy adott  $A_i$  szinten nem végrehajthatók. Láthatjuk, hogy a kölcsönös kizárások monoton csökkenése igaz, ha figyelembe vesszük, hogy a nem látható literálok és cselekvések minden mással kölcsönösen kizárák egymást.

A bizonyítás egy kicsit összetett, de a következő esetekkel kezelhető: ha az  $A$  és  $B$  cselekvések kölcsönösen kizárák egymást az  $A_i$  szinten, akkor ezt csak a háromféle kizárár egyike okozhatja. Az első kettő – az inkonzisztens hatások, illetve a következtetések – a cselekvések tulajdonságai, így, ha a cselekvések kölcsönösen kizárák egymást az  $A_i$  szinten, akkor minden szinten kizárók lesznek. A harmadik eset a versenyhelyzet, az  $S_i$  szint feltételeitől függ: ennek a szintnek tartalmaznia kell az  $A_i$  olyan előfeltételét, amely kölcsönösen kizárá a  $B_i$  előfeltétellel. Ez a két előfeltétel akkor lehet kölcsönösen kizárá, ha egymás negáltai (amely esetben minden szinten kölcsönösen kizárók), vagy ha minden cselekvés, amely teljesíti az egyiket, kölcsönösen kizárá az összes cselekvéssel, mely a másikat érheti el. Mi már tudjuk, hogy a rendelkezésre álló cselekvések monoton növekednek, így indukcióval bizonyítható, hogy a kölcsönös kizárások csökkenők.

Mivel a cselekvések és a literálok növekednek, a kölcsönös kizárások csökkennek, és mivel csak véges számú cselekvés és literál van, minden tervkészítési gráf szükségszerűen kiegyenlítődik – azaz minden egymást követő szint azonos lesz. Ha hiányzik a probléma egy célja, vagy a célok között vannak kölcsönösen kizárók, miután a gráf kiegyenlítődött, akkor a probléma soha nem oldható meg, így leállíthatjuk a GRAPHPLAN algoritmust, és hibával térhetünk vissza. Ha a gráf kiegyenlítődik, és minden cél jelen van kizárásként nélkül, de a MEGOLDÁS-KINYERÉS sikertelen a megoldás megtalálásában, akkor lehet, hogy véges lépésszámban bővítenünk kell a gráfot, de végezetül megállhatunk. A megállás problémaköre ennél összetettebb, amit itt nem tárgyalunk.

## 11.5. TERVKÉSZÍTÉS ÍTÉLETLOGIKÁVAL

A 10. fejezetben láthattuk, hogy a tervkészítés elvégezhető a szituációkalkulus egy tételének bizonyításával. A téTEL azt mondja, hogy adott kiinduló állapotra és követő állapot-axiómákra, amelyek a cselekvések következményeit írják le, a cél egy adott cselekvéssorból adódó helyzetben igaz lesz. Ezt a megközelítést már 1969-ben sem

találták elégé hatékonynak érdekes tervek megtalálására. Napjaink fejlesztései az ítéletlogikához készített hatékony magyarázatadó algoritmusokra (lásd 7. fejezet) ismét felkeltették az érdeklődést a tervkészítés logikai magyarázatkészítési megközelítésére.

Az ebben a fejezetben tárgyalt megközelítés a tételbizonyítás helyett egy cselekvés-sor **kielégíthetőségenek (satisfiability)** vizsgálatán alapul. Ítéletlogikai mondatok modelljét fogjuk megtalálni, melyek a következőképpen néznek ki:

$$\text{kiinduló állapot} \wedge \text{minden lehetséges cselekvésleírás} \wedge \text{cél}$$

A mondat ítéletlogikai szimbólumokat fog tartalmazni, melyek a lehetséges cselekvés-előfordulások megfelelői. A modell, ami kielégíti a mondatot, igaz értéket rendel azokhoz a cselekvésekhez, amelyek részei a helyes tervnek, és hamis értéket a többihez. Egy olyan célkitűzés, mely egy nem helyes terv megfelelője, nem lesz modell, mivel nem lesz konzisztens azzal a feltételezéssel, hogy a cél igaz. Ha a tervkészítési probléma megoldhatatlan, akkor a mondat sem kielégíthető.

## Tervkészítési problémák ítéletlogikai leírása

A STRIPS problémák ítéletlogikai leírásra való fordítását a tudásreprezentációs ciklus egy iskolapeldáján mutatjuk be: egy megfelelő axiómakészlettel indulunk, és úgy talál-juk, hogy ezek az axiómák hamis, nem várt modellekkel tesznek lehetővé, ezért további axiómákat adunk hozzájuk.

Kezdjük a nagyon egyszerű légi szállítási problémával. A kiinduló állapotban (0. idő-pont) a  $P_1$  repülő az *SFO* repülőterén van, míg a  $P_2$  repülő a *JFK*-n. A cél, hogy a  $P_1$  legyen a *JFK*, a  $P_2$  pedig az *SFO* reptéren, azaz a repülőgépeknek helyet kell cserélniük. Először egymástól eltérő ítéletlogikai szimbólumokra van szükségünk az egyes időlépések kijelentéseirehoz. Az időpillanatok jelöléséhez felső indexeket használunk, mint a 7. fejezetben. Így a kiinduló állapot felírása az

$$\text{Ott}(P_1, \text{SFO})^0 \wedge \text{Ott}(P_2, \text{JFK})^0$$

(Emlékezzünk arra, hogy az  $\text{Ott}(P_1, \text{SFO})^0$  egy atomi szimbólum!) Mivel az ítélet-logika nem használja a zárt világ feltételezést, ezért a kiinduló állapotban definiálnunk kell a *nem* igaz ítéletlogikai állításokat is. Ha néhány állítás nem ismert a kiinduló állapotban, akkor ezek maradhatnak meghatározatlanok (**nyílt világ feltételezés**). Ebben a példában az

$$\neg\text{Ott}(P_1, \text{JFK})^0 \wedge \neg\text{Ott}(P_2, \text{SFO})^0$$

kiinduló állapotot specifikáljuk. A célt magát egy megadott időlépéshez kell kapcsolni. Mivel nem tudjuk *a priori*, hogy hány lépéstre van szükség a cél eléréséhez, megpróbálhatjuk feltételezni, hogy a cél igaz a kiinduló állapotban,  $T = 0$  időpontban. Azaz a következőt állítjuk:  $\text{Ott}(P_1, \text{JFK})^0 \wedge \text{Ott}(P_2, \text{SFO})^0$ . Ha ez *sikertelen*, újra próbálkozunk  $T = 1$ -gyel, és így tovább, amíg a legkisebb megfelelő tervhosszat el nem érjük.  $T$  minden értékére a tudásbázis a 0. időponttól a  $T$ -ig tartó lépésekkel fedő mondatokat fogja tartalmazni. Hogy biztosítsuk a leállást, egy önkényes  $T_{\max}$  felső korlátot kell állítanunk. Ezt az algoritmust mutatja a 11.15. ábra. Egy másik lehetséges megközelítést, mely elkerüli a többszörös megoldási kísérleteket, a 11.17. feladatban tárgyalunk.

```

function SATPLAN(probléma,  $T_{\max}$ ) returns megoldás vagy kudarc
  inputs: probléma, egy tervkészítési probléma
           $T_{\max}$ , a terv maximális hossza

  for  $T = 0$  to  $T_{\max}$  do
    cnf, leképezés  $\leftarrow$  SATRA-FORDÍTÁS(probléma,  $T$ )
    feladat  $\leftarrow$  SAT-MEGOLDÓ(cnf)
    if feladat nem üres then
      return MEGOLDÁS-KINYERÉS(feladat, leképezés)
  return kudarc

```

**11.15. ábra.** A SATPLAN algoritmus. A tervkészítési feladatot egy konjunktív normál formájú mondatra fordítjuk le, amelyben minden cél egy megadott  $T$  időpillanatban teljesül, és  $T$ -ig minden lépésre tartalmazza az axiómákat. (A fordítás részleteit a szövegben írjuk le.) Ha a kielégíthetőségi algoritmus talál modellt, akkor a tervet a modellben az igaz értéket kapott ítéletlogikai szimbólumokhoz tartozó cselekvések kinyerésével kapjuk meg. Ha nem létezik modell, akkor a folyamatot ismételjük, a célt egy lépéssel későbbre mozgatva.

A következő kérdés, hogy hogyan kódoljuk a cselekvés leírásokat az ítéletlogikában. A legegyszerűbb megközelítés, hogy minden cselekvés megjelenésére egy ítéletlogikai szimbólumot vezetünk be. Például a  $Repül(P_1, SFO, JFK)^0$  igaz, ha a  $P_1$  repülőgép az *SFO*-ról a *JFK*-ra repül a 0. időpillanatban. Csakúgy, mint a 7. fejezetben, felírjuk az ítéletlogikai változatait a következő állapotaxiómáknak, melyeket a 10. fejezetben a szituációkalkulushoz fejlesztettünk ki. Adott például az

$$\begin{aligned} Ott(P_1, JFK)^1 \Leftrightarrow & (Ott(P_1, JFK)^0 \wedge \neg(Repül(P_1, JFK, SFO)^0 \wedge Ott(P_1, JFK)^0)) \\ & \vee (Repül(P_1, SFO, JFK)^0 \wedge Ott(P_1, SFO)^0) \end{aligned} \quad (11.1)$$

axióma. Ennek jelentése, hogy a  $P_1$  repülő a *JFK*-n lesz az 1-es időpillanatban, ha a *JFK*-n volt a 0. időpillanatban, és onnan nem repült el, vagy ha az *SFO*-n volt a 0. időpillanatban, és elrepült a *JFK*-ra. Egy ilyen axiómára minden egyes repülőgéphez, repülőtérhez és időpillanathoz szükségünk van. Mindezeken túl minden egyes hozzáadott repülőtér egy új kiinduló, illetve ott végződő útvonalat ad az összes repülőtérhez, ezért további diszjunkciókat ad az összes axióma jobb oldalához.

Ha ezek az axiómák megvannak, akkor egy kielégíthetőségi algoritmust futtathatunk a terv megtalálására. Kell, hogy legyen olyan terv, ami a célt a  $T = 1$  időpillanatban eléri, nevezetesen az a terv, melyben a két repülőgép helyet cserél. Tegyük fel, hogy a tudásbázis a következő:

$$kiinduló állapot \wedge követő állapot axiómák \wedge cél^1 \quad (11.2)$$

mely szerint a cél a  $T = 1$  időpillanatban igaz. Ellenőrizhető, hogy az eljárás, melyben a  $Repül(P_1, SFO, JFK)^0$  és a  $Repül(P_2, JFK, SFO)^0$

igaz, és minden más cselekvésszimbólum hamis, modellje a tudásbázisnak. Eddig itt minden rendben. Vannak más lehetséges modellek, melyeket a kielégíthetőségi algoritmus visszaadhat? Valójában igen. Ezen modellek mindegyike kielégítő terv? Sajnos nem. Vegyük például a

$$Repül(P_1, SFO, JFK)^0, a Repül(P_1, JFK, SFO)^0 \text{ és a } Repül(P_2, JFK, SFO)^0$$

cselekvésszimbólumok által specifikált meglehetősen butácska tervet. Ez a terv buta, mert a  $P_1$  repülő az *SFO*-ról indul, ezért a  $\text{Repül}(P_1, \text{JFK}, \text{SFO})^0$  cselekvés végrehajthatatlan. Mindazonáltal a terv modellje a (11.2) egyenlet mondhatának! Ez annyit jelent, hogy megfelel mindennek, amit eddig a problémáról állítottunk. Hogy megértsük miért, egy kicsit közelebbről meg kell vizsgálnunk, hogy a következő állapot axiómák mit mondanak el a cselekvésekéről, melyeknek az előfeltételei nem teljesülnek (mint a (11.1) egyenletben). Az axiómák helyesen írják le, hogy semmi sem történik, ha egy ilyen cselekvést végrehajtunk (lásd 11.15. feladat), de nem állítják, hogy egy ilyen cselekvés nem hajtható végre! Hogy elkerüljük az illegális cselekvések tartalmazó tervek készítését, további előfeltétel axiómákat (**precondition axioms**) kell felvennünk, melyek biztosítják, hogy egy cselekvés megjelenéséhez az előfeltételeknek teljesülni kell.<sup>6</sup>

Például szükségünk van az

$$\text{Repül}(P_1, \text{JFK}, \text{SFO})^0 \Rightarrow \text{Ott}(P_1, \text{JFK})^0$$

axiómára, mert az  $\text{Ott}(P_1, \text{JFK})^0$  hamis a kiinduló állapotban, így ez az axióma biztosítja, hogy a  $\text{Repül}(P_1, \text{JFK}, \text{SFO})^0$  minden modellben szintén hamis legyen. Az előfeltétel axiómák hozzáadásával már csak pontosan egy modell van, ami teljesíti az összes axiómát, amikor a cél az 1-es időpillanatban érjük el, nevezetesen az a modell, melyben a  $P_1$  repülőgép a *JFK*-ra, a  $P_2$  repülőgép pedig az *SFO*-ra repül. Vegyük észre, hogy ez a megoldás két párhuzamos cselekvést tartalmaz! Csakúgy, mint a GRAPHPLAN vagy a részben rendezett tervkészítés esetén.

További meglepetésekre számíthatunk, amikor egy új repülőteret illesztünk be, a *LAX*-ot (Los Angeles repülőtere). Most minden repülőgéphez minden állapotban két megengedett cselekvés is tartozik. Amikor egy kielégíthetőségi algoritmust futtatunk, úgy találjuk, hogy a  $\text{Repül}(P_1, \text{SFO}, \text{JFK})^0$ , a  $\text{Repül}(P_2, \text{JFK}, \text{SFO})^0$  és a  $\text{Repül}(P_2, \text{JFK}, \text{LAX})^0$ -ból álló modell kielégíti az axiómákat. Azaz a követő állapot axiómák és az előfeltétel axiómák megengedik, hogy egy repülőgép egyszerre két célállomásra is repülhessen! A kiinduló állapot a  $P_2$  minden repülőútjának előfeltételeit teljesíti, így a követő állapot axiómák szerint a  $P_2$  az *SFO*-n és a *LAX*-on található az 1-es időpillanatban, azaz a cél teljesül. Nyilvánvaló, hogy további axiómákat kell felvennünk, hogy kizárnak ezeket a hibás megoldásokat. Az egyik megközelítés, hogy **cselekvéskizáró axiómákat** (**action exclusion axioms**) veszünk fel, melyek megtagolják az egyidejű cselekvéseket. Például teljes kizárást érhetünk el, ha az összes

$$\neg(\text{Repül}(P_2, \text{JFK}, \text{SFO})^0 \wedge \text{Repül}(P_2, \text{JFK}, \text{LAX})^0)$$

alakú axiómát felvesszük. Ezek az axiómák biztosítják, hogy semelyik két cselekvés ne eshessen azonos időpontra. Ezek kizáják a hibás terveket, hatásukra minden terv teljesen rendezett lesz. Ez elveszíti a részben rendezett tervek rugalmasságát, és azáltal, hogy megnöveli a terv lépésszámát, a számítási idő is meghosszabbodhat.

A teljes kizáráshoz helyett alkalmazhatunk csupán részleges kizárást, amikor az egyidejű cselekvéseket csak akkor zárnak ki, ha kölcsönhatásba kerülnek. A feltételek a mutex cselekvések megfelelői: két cselekvés nem hajtható végre egyidejűleg, ha az egyik negálja a másik egy előfeltételét vagy következményét. Például a  $\text{Repül}(P_2, \text{JFK}, \text{LAX})^0$

<sup>6</sup> Figyeljük meg, hogy az előfeltétel axiómák felvétele azt jelenti, hogy nem kell a követő állapot axiómánál a cselekvésekhez előfeltételeket felvennünk.

és a  $\text{Repül}(P_2, JFK, SFO)^0$  együtt nem következhet be, mert mindenkető negálja a másik előfeltételét. Másrészről azonban a  $\text{Repül}(P_2, JFK, SFO)^0$  és a  $\text{Repül}(P_2, JFK, SFO)^0$  együtt is bekövetkezhet, hiszen a járatok nem zavarják egymást. A részleges kizárást a teljes rendezés kényszere nélkül kiküszöböli a hibás terveket.

A kizárási axiómák néha nagyon buta eszköznek tűnnek. Ahelyett hogy kikötönénk, hogy egy gép nem repülhet egyszerre két reptérre, mondhatnánk egyszerűen, hogy egyetlen tárgy sem lehet egyszerre két helyen:

$$\forall p, x, y, t \ x \neq y \Rightarrow \neg(Ott(p, x)^t \wedge Ott(p, y)^t)$$

Ez a tény a követő állapotot axiómákkal kombinálva, azt *implikálja*, hogy egy repülő nem repülhet egyszerre két reptérre. Az ilyen tényeket **állapotkorlátozásoknak (state constraints)** nevezünk. Az ítéletlogikában természetesen meg kell adnunk minden egyes állapotkorlátozás összes alappéldányát. A repülés feladatra az állapotkorlátozás elegendő az összes hibás terv kizáráshoz. Az állapotmegkötések gyakran sokkal tömörebbek, mint a cselekvéskizárási axiómák, ezek azonban a probléma eredeti STRIPS-rezentációjából nem mindig vezethetők le könnyedén.

Összefoglalva tehát a kielégíthetőségi tervkészítés, a kiinduló állapotot, a célt, a követő állapot axiómákat, az előfeltételeket, valamint a cselekvéskizárási vagy az állapotaxiomák egyikét tartalmazó mondathoz tartozó modellek megtalálását jelenti. Megmutatható, hogy az axiómák ezen halmaza elégsges, abban az értelemben, hogy a továbbiakban nincsenek hibás „megoldások”. Az ítéletlogikai mondatot kielégítő bármely modell megfelelő megoldása az eredeti problémának, azaz a terv minden sorba rendezése egy megengedett cselekvéssor, ami a célhoz vezet.

## Az ítéletlogikai leírás bonyolultsága

Az ítéletlogikai megközelítés legnagyobb hátránya az eredeti problémából generált tudásbázis mérete. A  $\text{Repül}(p, a_1, a_2)$  cselekvésséma például  $T \times |\text{Repülők}| \times |\text{Repülőterek}|^2$  számú különböző állításlogikai szimbólumot eredményez. Általánosságban a cselekvésszimbólumok számának korlátja a  $T \times |Act| \times |O|^P$ , ahol az  $|Act|$  a cselekvéssémák száma,  $|O|$  a problémakör objektumainak száma, valamint  $P$  a cselekvés maximális aritása (paramétereinek száma). A klózok száma még nagyobb, 10 időlépés, 12 repülőgép és 30 repülőtér esetén a teljes cselekvéskizárási axiómában 583 millió klóz van.

Mivel a cselekvésszimbólumok száma a cselekvéssémák aritásával exponenciálisan nő, megoldás lehet az aritás csökkentése. Ezt a szemantikus hálóktól (lásd 10. fejezet) kölcsönvett ötlet segítségével tehetjük meg. A szemantikus hálók csak bináris predikátumokat használnak, az ennél több argumentummal rendelkező predikátumokat egy bináris predikátumhalmazra képezzük le, ami külön ad meg minden predikátumot. Ezt az ötletet a  $\text{Repül}(P_1, SFO, JFK)^0$  cselekvésszimbólumra alkalmazva, három új szimbólumot kapunk:

- $\text{Repül}_1(P_1)^0$ : a  $P_1$  repülő a 0. időpontban repült
- $\text{Repül}_2(SFO)^0$ : a járat kiinduló állomása az *SFO*
- $\text{Repül}_3(JFK)^0$ : a járat célállomása a *JFK*

Ez a szimbólumfelosztásnak (*symbol splitting*) nevezett folyamat kiküszöböli az exponenciális számú szimbólum igényét. Így most csak  $T \times |Act| \times P \times |O|$  szimbólumra van szükség.

A szimbólum felosztás önmagában lecsökkenti ugyan a szimbólumok számát, de a tudásbázisban szereplő axiómák számát nem csökkenti automatikusan. Azaz, ha minden klózban szereplő cselekvésszimbólumot egyszerűen csak három szimbólum konjunkciójára cserélénk, a tudásbázis mérete nagyjából azonos maradna. A szimbólumfelosztás valóban csökkenti a tudásbázis méretét, mivel néhány szétbontott szimbólum feleslegessé válik bizonyos axiómákban, és elhagyható. Vegyük például a (11.1) egyenletben szereplő követő állapot axiómát, módosítva, hogy tartalmazza *LAX*-ot, és elhagyva a cselekvés előfeltételeket (melyeket külön előfeltétel axiómákkal fedünk le):

$$\begin{aligned} Ott(P_1, JFK)^1 \Leftrightarrow & (Ott(P_1, JFK)^0 \wedge \neg Repül(P_1, JFK, SFO)^0 \wedge \neg Repül(P_1, JFK, LAX)^0) \\ & \vee Repül(P_1, SFO, JFK)^0 \vee Repül(P_1, LAX, JFK)^0 \end{aligned}$$

Az első feltétel szerint a  $P_1$  a *JFK*-n lesz, ha a 0. időpontban ott volt, és nem repült el egyetlen másik városba sem. A második azt mondja, hogy ott lesz, ha mindegy, hogy honnan, de a *JFK*-ra repül. A felosztási szimbólumok használatával, egyszerűen elhagyhatjuk azon argumentumokat, melyek értéke nem számít:

$$\begin{aligned} Ott(P_1, JFK)^1 \Leftrightarrow & (Ott(P_1, JFK)^0 \wedge \neg Repül_1(P_1)^0 \wedge Repül_2(JFK)^0) \vee Repül_1(P_1)^0 \\ & \wedge Repül_3(JFK)^0 \end{aligned}$$

Vegyük észre, hogy az *SFO* és a *LAX* már nem szerepel az axiómában. Általánosabban a felosztási szimbólumoknak köszönhetően a követő állapot axiómák mérete függetlenné válik a repülőterek számától. Hasonló redukció történik az előfeltétel és a cselekvéskizárási axiómákkal (lásd 11.16. feladat). A korábban bemutatott 10 időlépés, 12 repülő és 30 reptér esetében a teljes cselekvéskizárási axóma mérete 583 millió klózról 9360 klózra csökken.

A módszernek egyetlen hátrólítője van: a felosztási szimbólum reprezentáció nem engedi meg a párhuzamos cselekvéseket. Vegyük a  $Repül(P_1, SFO, JFK)^0$  és a  $Repül(P_2, JFK, SFO)^0$  párhuzamos cselekvéseket. A felosztott megadásra alakítva a

$$\begin{aligned} & Repül_1(P_1)^0 \wedge Repül_2(SFO)^0 \wedge Repül_3(JFK)^0 \wedge \\ & Repül_1(P_2)^0 \wedge Repül_2(JFK)^0 \wedge Repül_3(SFO)^0 \end{aligned}$$

kifejezést kapjuk. Innentől már nem állapítható meg mi történt! Tudjuk, hogy  $P_1$  és  $P_2$  repült, de nem tudjuk azonosítani a járatok kiinduló és célállomásait. Ez annyit tesz, hogy egy teljes cselekvéskizárási axiómát kell használni, annak korábban jelzett hátrányaival.

A kielégíthetőségen alapuló tervkészítők nagyon nagy tervkészítési problémákat képesek kezelni – például a többtucatnyi dobozból álló kockavilág probléma 30 lépéses tervét. A propozíciós kódolás mérete, valamint a megoldás költsége nagymértékben problémafüggő, de a legtöbb esetben az ítéletlogikai axiómák tárolásához szükséges memóriaméret a szűk keresztmetszet. Ennek a kutatásnak egy érdekes eredménye, hogy a visszalépéses algoritmusok (mint a *DPLL*) gyakran hatékonyabbak voltak a tervkészítési problémák megoldásában, mint a lokális keresési algoritmusok (például a *WALKSAT*). Ennek oka, hogy az állításlogikai axiómák többsége Horn-klóz, ami egység terjesztéstechnikákkal kezelhető hatékonyan. Ez a megfigyelés hibrid algoritmusok kifejlesztéséhez vezetett, melyek a véletlen keresési, a hiba-visszaterjesztési és az egységterjesztési módszereket kombinálják.

## 11.6. A TERVKÉSZÍTÉSI MÓDSZEREK ELEMZÉSE

A mesterséges intelligencia területén belül, a tervkészítés jelenleg nagy figyelemnek örvend. Ennek egyik oka, hogy egyesíti az eddig bemutatott két nagy területet, a *keresést* és a *logikát*. A tervkészítő tekinthető egy programnak, ami vagy egy megoldást keres, vagy (konstruktívan) bizonyítja egy megoldás létezését. A két terület alapötleteinek keresztezése az utóbbi évtizedben több nagyságrendnyi javulást hozott a hatékonyság tekintetében, és egyúttal megnövelte a tervkészítők ipari alkalmazásokban való felhasználhatóságát. Sajnos még mindig nincs tiszta képünk arról, hogy mely problémákra mely megoldások a legalkalmasabbak. Később valószínűleg új technikák kerülnek majd előtérbe, melyek dominálnak majd a meglévők felett.

A tervkészítési feladat elsősorban a kombinatorikus robbanás kézbentartásáról, kontrollásról szól. Egy  $p$  elemi állítást tartalmazó problématerben  $2^p$  állapot lehetséges. Összetett problémakörökben  $p$  nagyon nagy lehet. Vegyük azt, hogy a problémater objektumainak tulajdonságai (*Pozíció, Szín* stb.) és relációi (*Ott, Rajta, Közte* stb.) vannak. Az állapotok száma,  $d$  objektummal és hármás relációkkal a problématerben  $2^{d^3}$ . Végekövetkeztetésként – a legrosszabb esetet feltételezve – megállapíthatjuk, hogy a tervkészítés reménytelen.

Az ilyen pessimizmus ellen az „oszd meg és uralkodj” elv hatásos fegyver lehet. A legjobb esetben – amennyiben a probléma teljesen dekomponálható – az „oszd meg és uralkodj” elv exponenciális gyorsulást hozhat. A dekomponálhatóságot azonban a cselekvések negatív kölcsönhatásai megszüntetik. A részben rendezett tervkészítő ezt egy erős reprezentácos megközelítéssel, az okozati kapcsolatokkal kezelte, de sajnos a konfliktusok mindegyike csak egy választással oldható fel (ami a konfliktusban álló cselekvést vagy a kapcsolat elé, vagy mögé sorolja) és ezen választások száma exponenciálisan nő. A GRAPHPLAN algoritmus a gráf felépítésekor kikerüli ezeket a döntéseket úgy, hogy az ütközéseket mutex kapcsolatokkal rögzíti anélkül, hogy a feloldás módjáról döntene. A SATPLAN algoritmus a mutex kapcsolatok hasonló területét fedi le, de ezt egy specifikus adatstruktúra felhasználása helyett az általános CNF alak felhasználásával teszi. Az, hogy ez mennyire jól működik, a felhasznált SAT megoldótól függ.

Néha a probléma hatékonyan megoldható, ha észrevesszük, hogy a negatív kölcsönhatások kizáthatók. Azt mondjuk, hogy egy feladatnak **sorba rendezhető részceljai** (*serializable subgoals*) vannak, ha létezik a részcéloknak egy olyan rendezése, hogy a tervkészítő sorrendben elérheti őket anélkül, hogy a korábban már elért részcélokat felszámolná. A kockavilágban például, ha a cél egy torony megépítése (például az  $A$  a  $B$ -n, ami pedig az asztalon elhelyezkedő  $C$ -n van), akkor a részcélok felülről lefelé sorrendezhetők: ha először elérjük, hogy  $C$  az asztalon van, akkor a továbbiakban – mielőtt a további részcélokat teljesítjük – ennek megszüntetésére soha nem lesz szükség. Egy tervkészítő, ami az alulról felfelé trükköt alkalmazza, a kockavilág bármely problémáját képes megoldani visszalépés nélkül (bár nem minden a legrövidebb tervet adja).

Egy összetettebb példaként, a NASA Deep Space 1 űrhajóját irányító Távirányító Ágens tervkészítő esetén elhatározták, hogy az űrhajót irányító állítások legyenek sorba rendezhetők. Ez talán nem is annyira meglepő, hiszen az űrhajót a mérnökök úgy terveztek, hogy minél könnyebb legyen irányítani (más megkötések figyelembevételével). A célok sorrendezhetőségét kihasználva a Távirányító Ágens tervkészítő a keresés nagy részét megspórolhatta, így megfelelően gyors volt az űrhajó valós idejű irányítására, amit korábban lehetetlennek tartottak.

A kombinatorikus robbanás kordában tartására több lehetőség is van. Az 5. fejezetben láthattuk, hogy a kényszerkielégíthetőségi problémák (CSP-k) visszalépési számának kontrollálásához számos megoldás létezik, mint például a függősegírányított visszalépés. Ezen technikák mindegyike alkalmazható a tervkészítéshez is. Például a megoldás kinyerése egy tervkészítési gráfból kezelhető egy kétértékű CSP-ként, melynek változói azt jelzik, hogy egy adott cselekvés bekövetkezik-e egy adott időpontban, vagy sem. A CSP megoldható az 5. fejezet bármely algoritmusával, például a min-konfliktusok algoritmussal. Egy közeli rokon, a BLACKBOX rendszerben használt megoldás, a tervkészítési gráfot CNF kifejezésre fordítja, melyből ezután egy SAT megoldóval nyeri ki a tervet. Ez a megközelítés jobb, mint a SATPLAN algoritmus, feltételezhetően azért, mert a tervkészítési gráf már számos lehetetlen állapotot és cselekvést eltávolított a feladatból. A GRAPHPLAN algoritmusnál is jobban működik, valószínűleg azért, mert a kielégíthetőségi keresés, mint a WALKSAT, nagyobb rugalmasságú a GRAPHPLAN algoritmus által használt szigorú visszalépéses keresésnél.

Nem kétséges, hogy a GRAPHPLAN-hoz, a SATPLAN-hoz és a BLACKBOX-hoz hasonló tervkészítők előremozdították a tervkészítés fejlődését azáltal, hogy megnövelték a tervkészítő rendszerek teljesítményét, és tisztázták a kapcsolódó reprezentációs és komplexitási kérdéseket. Ezek a megoldások azonban eredendően ítéletlogikai megoldások, így korlátozott a problémakör, amelyet képesek kifejezni. (Például néhány tucat objektumot és helyet tartalmazó logisztikai problémának megfelelő CNF kifejezésekhez gigabájtnyi kapacitásra van szükség.) Bár a tervkészítési gráphoz hasonló struktúrák továbbra is hasznosak lesznek, mint a heurisztikák forrása, valószínűnek tűnik, hogy elsőrendű reprezentációkra és algoritmusakra lesz szükség a további fejlődéshez.

## 11.7. ÖSSZEFoglalás

Ebben a fejezetben a tervkészítési feladatot determinisztikus és teljesen megfigyelhető környezetek esetén definiáltuk. Bernutattuk a tervkészítési feladatok leírására használt főbb reprezentációkat, valamint számos algoritmikus megközelítést a megoldásukra. Idézzük fel a fontosabb állításokat:

- A tervkészítő rendszerek olyan problémamegoldó algoritmusok, melyek állapotok és cselekvések explicit ítéletlogikai (vagy elsőrendű) reprezentációin működnek. Ezek a leírások lehetővé teszik hatékony heurisztikák származtatását, valamint erős és rugalmas algoritmusok kifejlesztését a problémamegoldáshoz.
- A STRIPS nyelv az előfeltételek és következmények segítségével írja le a cselekvéseket, valamint a kiindulási és célállapotokat pozitív literálok konjunkciójaként adj meg. Az ADL nyelv lazít ezeken a megkötéseken, megengedi a diszjunkciót, a negálást és a kvantorokat.
- Az állappotr-keresés történet előrefelé: **progreszív (progression)** vagy hátrafelé: **regresszív (regression)**. Hatékony heurisztikák származthatók a részcélok függetlenségét feltételezve és a tervkészítési feladat különböző gyengítéseivel.
- A részben rendezett tervkészítő (RRT) algoritmusok a tervek terében keresnek anélkül, hogy cselekvések teljesen rendezett sorozatára jutnának. A céltól visszafelé működnek a részcélok eléréséhez szükséges cselekvéseknek a tervez való hozzá-

adásával. Ezek az algoritmusok különösen hatékonyak az „oszd meg és uralkodj” megközelítéssel kezelhető problémákra.

- Egy tervkészítési gráf (**planning graph**) megalkotható inkrementálisan a kiinduló állapotból elindulva. minden egyes réteg az adott időpillanatban végrehajtható összes cselekvés és literál halmazt tartalmazza, rögzíti a kölcsönös kizárásiakat vagy mutexeket, azaz olyan literálok és cselekvések közötti relációkat, melyek egyszerre nem következhetnek be. A tervkészítési gráfok hasznos heurisztikákra vezetnek az állapottér és a részben rendezett tervkészítők esetén, és közvetlenül felhasználhatók a GRAPHPLAN algoritmusban.
- A GRAPHPLAN algoritmus a tervkészítési gráfot dolgozza fel visszafelé keresést alkalmazva a terv kinyeréséhez. Lehetőséget biztosít továbbá még a cselekvések között bizonyos részben rendezésekre.
- A SATPLAN algoritmus a tervkészítési problémát ítéletlogikai axiómákra fordítja, és egy kielégíthetőségi algoritmust használ a helyes tervnek megfelelő modell megtalálására. Számos különböző ítéletlogikai reprezentációt fejlesztettek ki különböző mértékű tömörséggel és hatékonysággal.
- A tervkészítés fő megközelítései mindegyikének vannak nehézségei, és egyelőre nincs egyetértés abban, hogy melyik a legjobb. A versengés és a módszerek ötvözése komoly előrehaladást eredményezett a tervkészítő rendszerek hatékonyságában.

## Irodalmi és történeti megjegyzések

A mesterséges intelligenciában a tervezés az állapottér-keresések vizsgálatából, tételebizonyításból, szabályozástechnikából, illetve a robotika, ütemezés és más területek gyakorlati szükségleteiből ered. Az első számottevő tervkészítő rendszer a STRIPS (Fikes and Nilsson, 1971), ezen hatások kölcsönhatását példázza. A STRIPS-et, az SRI-ben zajló Shakey robot projekt szoftverének tervkészítő komponenseként fejlesztették ki. Ennek teljes szabályozási struktúráját egy eszköz-cél analízist használó állapottér-kereső rendszer, a GPS, azaz az általános problémamegoldó (general problem solver) (Newell és Simon, 1961) mintájára készítették. A STRIPS a QA3 tételebizonyító rendszer egy verzióját (Green, 1969b) használta fel a cselekvések előfeltételeinek teljesítésére. Lifschitz a STRIPS precíz definícióit és elemzéseit adja meg (Lifschitz, 1986). Bylander bizonyította, hogy az egyszerű STRIPS tervkészítés PSPACE-teljes (Bylander, 1992). Fikes és Nilsson történelmi áttekintést adnak a STRIPS projektről, és feltáriják ennek kapcsolatait az újabb tervkészítési törekvésekkel (Fikes és Nilsson, 1993).

A STRIPS-ben használt cselekvésreprezentációnak sokkal nagyobb hatása volt, mint algoritmikus megközelítésének. Ezt követően majdnem minden tervkészítő rendszer használta a STRIPS nyelv ilyen vagy olyan módosításait. Sajnos a különböző változatok elburjánzása szükségtelenül megnehezítette az összehasonlításokat. Az idő műlásaval jobban megértettük az egyes formalizmusok korlátait és hátrányait. A cselekvésleíró nyelv (Action Description Language – ADL) (Pednault, 1986) lazította a STRIPS nyelv néhány megkötését, és lehetővé tette a valósághoz közelibb problémák leírását. Nebel az ADL leírást STRIPS leírásra fordító sémákat vizsgálja (Nebel, 2000). A problémater leíró nyelvet (Problem Domain Description Language – PDDL) (Ghallab és társai, 1998) egy számítógép által érthető szabványos szintaxisként vezették be a STRIPS, az

ADL és egyéb nyelvek leírására. A PDDL-t 1998-tól az AIPS konferencián zajló tervkészítési versenyek szabványos nyelveként használták.

Az 1970-es évek elejének tervkészítői általában teljesen rendezett cselekvéssorokkal dolgoztak. A problémadekompozíciót úgy érték el, hogy minden részcélhoz egy résztervet készítettek, majd ezeket valamilyen sorrendben összefűzték. Hamarosan felfedezték, hogy ez a megközelítést, melyet Sacerdoti **lineáris tervkészítésnek** (*linear planning*) nevezett (Sacerdoti, 1975), nem teljes. Nem képes megoldani néhány nagyon egyszerű problémát, mint például a Sussman-anomáliát (lásd 11.11. feladat), amit Allen Brown a HACKER rendszerrel végzett kutatásai során fedezett fel (Sussman, 1975). Egy teljes tervkészítőnek meg kell engedni egy mondatban a különböző résztervekből származó cselekvések összefésülését (*interleaving*). A sorrendezhető részcélok alapötlete (Korf, 1987) pontosan megfeleltethető azon problémák halmazának, melyekre az összefésülésre nem alkalmas tervkészítők teljesek.

Az összefésülési probléma egyik megoldása a célregressziós tervkészítés volt, ami egy olyan technika, melyben egy teljesen rendezett terv lépései a részcélok közötti konfliktusok elkerülése érdekében újrarendezzük. Ezt Waldinger vezette be (Waldinger, 1975), és Warren WARPLAN rendszerében (Warren, 1974) szintén felhasználták. A WARPLAN-nel kapcsolatban szintén kiemelendő, hogy ez volt az első tervkészítő, melyet logikai programozási nyelven (Prolog) írtak, valamint az egyik legjobb példa arra a kiemelkedő gazdaságosságra, ami a logikai programozással nyerhető: a WARPLAN összesen száz kódosor, azaz csak töredéke az abban az időben ismert hasonló tervkészítők méreteinek. Az INTERPLAN (Tate, 1975a; 1975b) a Sussman-anomália és hasonló problémák elkerülése érdekében, szintén megengedte a tervlépések tetszőleges összefésülését.

A részben rendezett tervkészítés alapötletei tartalmazzák az ütközések érzékelését (Tate, 1975a) és az elérő feltételek kölcsönhatásuktól való védelmét (Sussman, 1975). A részben rendezett tervek (melyeket akkor **feladathálóknak** – *task network* – neveztek) készítését a NOAH tervkészítő (Sacerdoti, 1975; 1977) és Tate NONLIN rendszere vezette be (Tate, 1975b; 1977).<sup>7</sup>

A következő húsz év kutatásaiban a részben rendezett tervkészítés dominált, mialatt azonban a problémakört széles körben nem értették. A TWEAK (Chapman, 1987) ezen időszak tervkészítési munkájának logikai reprezentációja és egyszerűsítése volt. Chapman felírása elégőt tisztta volt ahhoz, hogy bizonyítható legyen a tervkészítési problémák különböző megfogalmazásainak teljessége és követhetetlensége (NP-nehézsége és -eldönthetetlensége). Chapman munkája egy teljes részben rendezett tervkészítő első egyszerű és olvasható leírására vezetett (McAllester és Rosenblitt, 1991). A McAllester és Rosenblitt SNLP-nek (Soderland és Weld, 1991) nevezett algoritmusának implementációja széles körben elterjedt, és elsőként tette számos kutató számára lehetővé a részben rendezett tervkészítők megértését és kutatását. Az ebben fejezetben korábban bemutatott RRT algoritmus az SNLP-n alapul.

Weld csapatja fejlesztette ki az UCPOP-ot is (Penberthy és Weld, 1992), az első tervkészítőt ADL-ben kifejezett problémákra. Az UCPOP a kielégítetlen-célok-száma heurisztikát alkalmazta. Valamivel gyorsabban futott, mint az SNLP, de nagyon ritkán volt képes néhány tucat lépésnél többet tartalmazó tervek megtalálására. Bár javított

<sup>7</sup> A terminológia kissé zavaros. Sok szerző a **nemlineáris (nonlinear)** jelzést a „részben rendezett” ételemben használja. Ez kissé különbözik Sacerdotti eredeti szóhasználatától, aki ezt az összefésült tervekre használta.

heurisztikákat is kifejlesztettek az UCPOP módszerhez (Joslin és Pollack, 1994); (Gerevini és Schubert, 1996) az 1990-es években, a gyorsabb módszerek megjelenésével a részben rendezett tervkészítők háttérbe szorultak. Guyen és Kambhampati a módszer visszatérését javasolták (Guyen és Kambhampati, 2001); a tervkészítési gráfból megfelelő heurisztikák származtatásával a REPOP tervkészítőjük jobban skálázható, mint a GRAPHPLAN, továbbá ez vetélytársa lett a leggyorsabb állapottér-tervkészítőknek is.

Avrim Blum és Merrick Furst felpezsdítette a tervkészítés területét GRAPHPLAN rendszerével (Blum és Furst, 1995; 1997), ami több nagyságrenddel gyorsabb volt az akkori részben rendezett tervkészítőknél. Más gráftervező rendszerek, mint az IPP (Koehler és társai, 1997), a STAN (Fox és Long, 1998) és az SGP (Weld és társai, 1998) hamarosan követték. Egy kicsivel korábban Ghallab és Laurelle a tervkészítő gráfhoz nagyon hasonló adatstruktúrát fejlesztettek ki (Ghallab és Laurelle, 1994). Az IXTET részben rendező tervkészítőjük az adatstruktúrát a keresés irányítására szolgáló heurisztikák kinyerésére alkalmazta. Guyen és társai a tervkészítő gráfokból származtatott heurisztikák nagyon alapos áttekintését adták (Guyen és társai, 2001). A mi tervkészítő gráfokról szóló leírásunk részben ezen, részben pedig Subbarao Kambhampati jegyzetein alapul. Ahogyan már a fejezetben említettük, többféle módon használható a tervezési gráf a megoldás keresésének irányítására. A 2002-es AIPS tervkészítő győztese, az LPG (Gerevini és Serina, 2002), a WALKSAT által ösztönözve lokális keresési technikát használt a tervkészítési gráf keresésére.

A tervkészítés mint kielégíthetőség és a SATPLAN algoritmus Kautz és Selman javaslata, akiket a kielégíthetőségi problémákra alkalmazott mohó lokális keresés meglepő sikere inspirált (Kautz és Selman, 1992) (lásd 7. fejezet). Kautz és társai szintén kutatták a STRIPS axiómák ítéletlogikai reprezentációjának különböző módjait, és azt találták, hogy a legtömörebb alakok nem feltétlenül vezetnek a leggyorsabb megoldásokra (Kautz és társai, 1996). Ernst és társai szisztematikus elemzést hajtottak végre (Ernst és társai, 1997), valamint kifejlesztettek egy automatikus fordítót PDDL problémák ítéletlogikai megadásának generálására. A Kautz és Selman által kifejlesztett BLACKBOX tervkészítő (Kautz és Selman, 1998) a GRAPHPLAN és SATPLAN alapötleteit kombinálja.

Az állapottér-tervkészítőkkel kapcsolatos érdeklődés újjáéledésének úttörője Drew McDermott UNPOP programja (McDermott, 1996), amely elsőként javasolta a törlési listát figyelmen kívül hagyó egyszerűsített problémán alapuló távolság heurisztikát. Az UNPOP név a részben rendezett tervkészítőket illető túlzott figyelem egyik reakciója volt; McDermott szerint más megközelítések nem kapták meg a megérdemelt figyelmet. Bonet és Geffner heurisztikus kereső tervkészítője (Heuristic Search Planner – HSP) és ennek későbbi leszármazottjai (Bonet és Geffner, 1999) elsőként tették az állapottér-keresést alkalmazhatóvá nagyméretű tervkészítési feladatokra. A mai napig legsikeresebb állapottér-kereső Hoffmann FASTFORWARD vagy FF keresője, az AIPS 2000 tervkészítő versenyének győztese (Hoffmann, 2000). Az FF egyszerűsített tervkészítési gráf heurisztikát használ egy nagyon gyors, az előrefelé és a lokális keresést újszerűen ötvöző algoritmussal.

Napjainkban a tervezők bináris döntési diagram (binary decision diagram) alakú reprezentációja hódít, ami egy véges automata tömör leírási módja, melyet a hardver-verifikációval foglalkozó közösség részletesen tanulmányozott (Clarke és Grumberg, 1987; McMillan, 1993). A bináris döntési diagramok jellemzőinek mint egy tervkészítési probléma megoldásának való megfelelés tulajdonságának bizonyítására számos

technika elérhető. Cimatti és társai egy ezen a megközelítésen alapuló tervkészítőt mutattak be (Cimatti és társai, 1998). Más reprezentációkat szintén felhasználtak; például Vossen és társai az egész programozás tervkészítésre való felhasználhatóságát vizsgálja (Vossen és társai, 2001).

A döntőbizottság még nem döntött, de a különböző tervkészítési algoritmusok néhány nagyon érdekes összevetése már elérhető. Helmert a tervkészítési problémák számos osztályát vizsgálja, és megmutatja, hogy az NP-nehéz problémakörökben a megkötés-alapú megközelítések, mint a GRAPHPLAN és a SATPLAN bizonyulnak a legjobbnak, míg a keresésalapú megközelítések azon problémakörökben hasznosak, ahol visszalépések nélkül elérhető egy megfelelő megoldás (Helmert, 2001). A GRAPHPLAN és SATPLAN algoritmusoknak nehézséget okoznak a sok objektumot tartalmazó problémák, mert ezek miatt sok cselekvésre van szükségük. Az esetek egy részében a probléma elkerülhető vagy elodázható, ha a cselekvések bizonyos részét csak szükség esetén generáljuk le dinamikusan, ahelyett hogy már a keresés előtt példányosítanánk mindegyiket. Weld a modern tervkészítő algoritmusok két kitűnő áttekintését adja (Weld, 1994; 1999). Érdekes látni az eltelt öt év változásait az áttekintésekben: az első a részben rendezett tervkészítésre fókuszál, míg a második a GRAPHPLAN és SATPLAN algoritmusokra. A *Readings in Planning* (Allen és társai, 1990) a terület legjobb korábbi cikkeinek minden részletre kiterjedő antológiája, beleértve néhány áttekintést is. Yang a részben rendezett tervkészítők könyv méretű áttekintését adja (Yang, 1997).

A tervkészítés kutatása a megjelenésétől kezdve központi kérdése a mesterséges intelligencia kutatásának, így a tervkészítésről született cikkek is számottevő részét képezik a folyóiratok és konferenciák anyagának. Specializált konferenciákat is rendeznek, mint az International Conference on AI Planning Systems (AIPS), az International Workshop on Planning and Scheduling for Space vagy a European Conference on Planning.

## Feladatok

- 11.1. Írja le a különbségeket és hasonlóságokat a problémamegoldás és a tervkészítés között.
- 11.2. Adottak a 11.2. ábra axiómái. Melyek az alkalmazható konkrét példányai a *Repül(p, honnan, hova)* cselekvésnek a következő állapotban:

$$\begin{aligned} & \text{Ott}(P_1, \text{JFK}) \wedge \text{Ott}(P_2, \text{SFO}) \wedge \text{Repülő}(P_1) \wedge \text{Repülő}(P_1) \\ & \quad \wedge \text{Repülőtér}(\text{JFK}) \wedge \text{Repülőtér}(\text{SFO}) \end{aligned}$$

- 11.3. Nézzük meg, hogyan fordíthatunk le egy STRIPS sémahalmazt a szituáció-kalkulus követő állapot axiómáira (lásd 10. fejezet).
  - Vegyük a *Repül(p, honnan, hová)* sémat. Írja le a *RepülElőfeltétel(p, honnan, hová, s)* predikátum logikai definícióját, ami igaz, ha a *Repül(p, honnan, hová)* előfeltételei teljesülnek az s szituációban.
  - Ezután írjuk fel a *Ott(p, x, s)* követő állapot axiómáját, ami a cselekvéssé-mával azonos információt tartalmazza, feltételezve, hogy a *Repül(p, honnan, hová)* az ágens számára rendelkezésre álló egyetlen cselekvésséma.

- Most feltételezzük, hogy van egy további utazási lehetőség: *Teleportálás(p, honnan, hová)*. Ennek egy további előfeltétele a  $\neg Torzult(p)$ , extra következménye pedig a *Torzult(p)*. Magyarázza meg, hogyan kell módosítani a szituációkalkulus tudásbázisát.
  - Végezetül fejlesszen ki egy általános és pontosan specifikált megoldást egy STRIPS sémahalmaz követő állapot axióma halmazra való lefordításához.
- 11.4.** Egy majom a „majom-és-banán” feladattal találkozik egy laboratóriumban, ahol elérhetetlen banánfűrtök lógnak a plafonról. Van egy doboz, ami lehetővé teszi a majom számára a banán elérését, amennyiben felmászik rá. Kiindulásképp a majom az *A*-ban van, a banánok a *B* helyen és a doboz a *C*-n. A majom és a doboz magassága *Alacsony*, de ha a majom a dobozra mászik, magassága *Magas* lesz, megegyezik a banánok magasságával. A majom számára elérhető cselekvések tartalmazzák a *Menj* egyik helyről a másikra, a *Tol* egy tárgyat egyik helyről a másikra, a *Felmászik* egy tárgyra, illetve *Lemászik* egy tárgyról, és a *Megfog*, illetve *Elenged* egy tárgyat cselekvéseket. A megmarkolás a tárgy megfogását eredményezi, ha a majom és a tárgy ugyanazon a helyen és ugyanabban a magasságban vannak.
- Írja fel a kiinduló állapotot.
  - Írja le a hat cselekvés definícióját STRIPS alakban.
  - Tegyük fel, hogy a majom meg akarja tréfálni a tudósokat, akik teázni mentek, azáltal hogy megragadja a banánokat, de a dobozt az eredeti helyén hagyja. Írja fel ezt mint általános célt (például nem feltételezve, hogy a doboz szükségszerűen a *C* helyen van) a szituációkalkulus nyelvén. Megoldható ez egy STRIPS-szerű rendszerrel?
  - A tolási axiómája valószínűleg helytelen, mert a tárgy túl nehéz, így a pozíció azonos marad, ha a *Tol* operátort alkalmazzuk. Ez elágazási vagy értékelési feladatra ad példát? Javítsa ki a feladat leírását, hogy figyelembe vegye a nehéz tárgyakat.
- 11.5.** Magyarázza meg, hogy a hátrafelé keresésben a megelőző állapotok generálásának folyamata miért nem igényli, hogy a cselekvés negatív következményeinek megfelelő literálokat is szerepeljessük.
- 11.6.** Magyarázza meg, miért vezet relaxált problémára a negatív következmények eldobása egy STRIPS probléma minden cselekvés sémájából.
- 11.7.** Vizsgálja meg a 3. fejezetben szereplő kétirányú keresés (**bidirectional search**) definícióját.
  - A kétirányú állapottér-keresés használata jó ötlet lenne tervkészítéshez?
  - És a részben rendezett tervezek terében történő kétirányú keresés?
  - Tervezze meg a részben rendezett tervkészítő azon verzióját, melyben egy cselekvés akkor adható hozzá a tervhez, ha az előfeltételei elérhetők a tervben már szereplő cselekvések következményein keresztül. Magyarázza meg, hogyan kezeljük a konfliktusokat és a rendezési megkötéseket. Szükségszerűen azonos ez az algoritmus az állapottérben történő előrefelé keréssel?

- (d) Vegyünk egy részben rendezett tervkészítőt, ami kombinálja a c) részfeladat módszerét azzal az általános módszerrel, ahol a nyitott feltételek eléréséhez cselekvéseket adunk a tervhez. Azonos lesz az így előálló algoritmus a b) részfeladattal?

**11.8.** Készítse el a 11.2. ábrán szereplő probléma tervkészítési gráfjának 0., 1. és 2. szintjét.

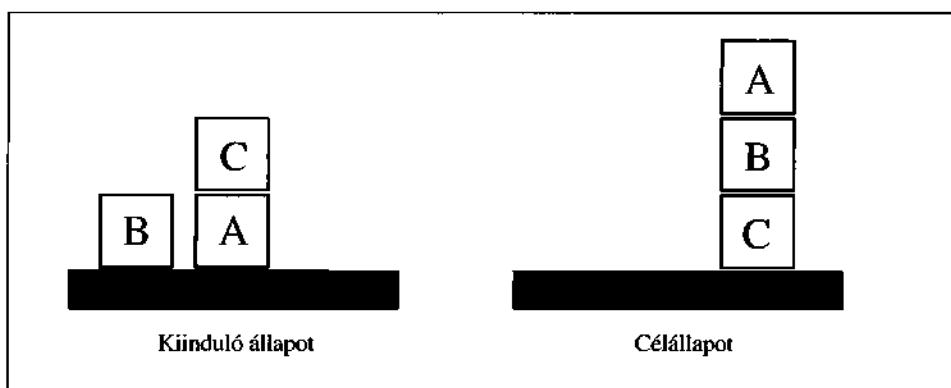
**11.9.** Bizonyítsa be az alábbi tervkészítő gráfokra vonatkozó állításokat:

- Az a literál, ami nem jelenik meg a gráf utolsó szintjén, nem érhető el.
- Egy soros gráfban szereplő literál szintköltsége nem lehet nagyobb, mint az őt elérő optimális terv költsége.

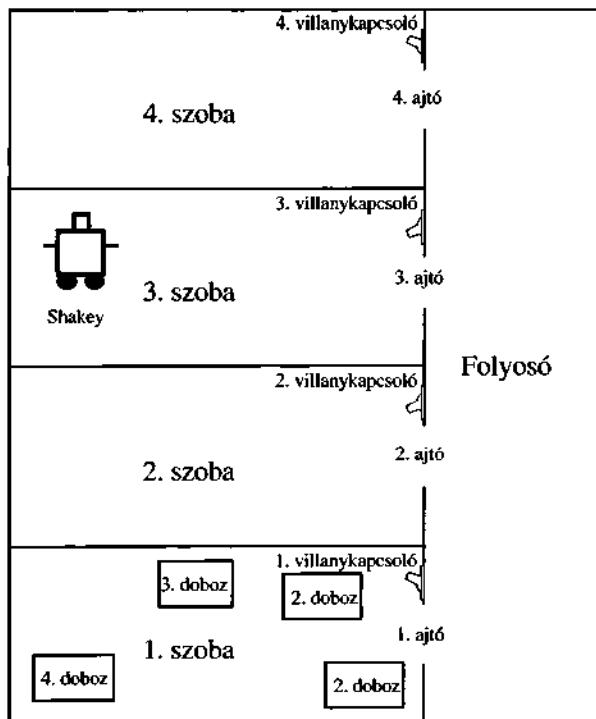
**11.10.** Összevetettük az állapottérben előre- és hátrafelé kereső tervkészítőket a részben rendezett tervkészítőkkel úgy, hogy az utóbbi egy tervtérkeresőnek vettük. Magyarázza meg, hogyan tekinthető az előre- és hátrafelé történő állapottérkeresés tervtérkeresőnek, és mondja meg, mik a tervfinomító operátorok.

**11.11.** A 11.16. ábra a kockavilág problémát mutatja, ami **Sussman-anomáliaként** (**Sussman anomaly**) ismert. A problémát azért tekintették anomáliának, mert az 1970-es évek nem összefűlös tervkészítői nem tudták megoldani. Írjuk fel a probléma definícióját STRIPS alakban, és oldjuk meg, kézzel vagy egy tervkészítő programmal. A nemösszefűlös tervkészítő egy olyan tervkészítő, ami ha két  $G_1$  és  $G_2$  részcélt kap, akkor vagy egy  $G_1$  elérésére szolgáló tervet fűz hozzá egy  $G_2$  elérésére szolgáló tervhez, vagy fordítva. Magyarázza meg, hogy egy nem összefűlös tervkészítő miért nem tudja megoldani a kockavilág problémát.

**11.12.** Vegyük a cipő és zokni felvételének problémáját, ahogy a 11.3. alfejezetben definiáltuk. Alkalmazzuk a GRAPHPLAN algoritmust erre a problémára, és mutassuk meg, hogyan áll elő a megoldás. Adjuk hozzá a kabát és a kalap felvételére szolgáló cselekvéseket. Adjuk meg azt a részben rendezett tervet, ami megoldás.



**11.16. ábra.** A Sussman-anomália kockavilág tervkészítési probléma



**11.17. ábra.** Shakey világa. Shakey képes egy szobán belül mozogni, át tud menni a szobák közötti ajtókon, fel tud mászni tárgyakra, el tudja tolni a mozdítható tárgyakat, valamint a villányt tudja kapcsolni.

és mutassuk meg, hogy ennek 180 különböző sorrendezése van. Mi a 180 különböző sorrendezés reprezentációjához szükséges különböző tervkészítő gráf megoldások minimális száma?

**11.13.** Az eredeti STRIPS programot a Shakey robot irányítására készítették. A 11.17. ábra a Shakey világának egy változatát mutatja, melyben négy szoba sorakozik egy folyosó mentén, melyek mindeneknek van egy ajtaja és van benne egy villanykapcsoló.

Shakey világának cselekvései tartalmazzák az egyik helyről a másikra való mozgást, mozgatható tárgyak (mint dobozok) eltolását, szilárd testekre (mint dobozokra) való fel- és lemászást, és villanykapcsolók fel-, illetve lekapcsolását. A robot önmagában soha nem volt elég ügyes ahhoz, hogy egy dobozra másszon vagy egy kapcsolót átkapcsoljon, de a STRIPS tervkészítő képes a robot képességein túlmutató tervezet találni és megadni. Shakey hat cselekvése a következő:

- A *Megy(x, y)*, ami feltételezi, hogy Shakey x-ben van, valamint az x és y helyek ugyanazon szobában vannak. Konvenció alapján a két szoba közötti ajtó mindenben benne van.
- A *b doboz eltolása x pozícióból ugyanazon szoba y pozíciójába: Tol(p, x, y)*. Szükségünk lesz a *Doboz* predikátumra és konstansokra a dobozokhoz.

- Dobozra felmászás: *Felmászik(b)*; dobozról lemászás: *Lemászik(b)*. Szükségünk lesz a *Rajta* predikátumra és a *Padló* konstansra.
- A villany felkapcsolása: *Felgyűjt(s)*; lekapcsolása: *Leolt(s)*. A lámpa fel-, illetve lekapcsolásához Shakey-nek egy dobozon kell állnia a villanykapcsoló mellett.

A 11.17. ábra alapján írja fel Shakey hat cselekvését és a kiinduló állapotot STRIPS jelölésekkel. Készítsen tervet Shakey számára, hogy a *Doboz<sub>2</sub>* dobozt a *Szoba<sub>2</sub>* szobába mozgassa.

- 11.14.** Láttuk, hogy a tervkészítési gráfok, csak ítéletlogikai cselekvéseket képesek kezelni. Mi van, hogyha a tervkészítő gráfokat a célban változókat is tartalmazó problémákra kívánjuk alkalmazni, mint az  $Ott(P_1, x) \wedge Ott(P_2, x)$ , ahol  $x$  egy véges terület pozícióból kerül ki? Hogyan tudna egy ilyen problémát átkódolni, hogy tervkészítő gráfokkal dolgozhasson? (Segítség: emlékezzen a részben rendezett tervkészítés Vége cselekvésére. Ehhez milyen előfeltételeknek kellene tartozni?)

- 11.15.** Egészen eddig feltételeztük, hogy a cselekvések csak a megfelelő helyzetekben kerülnek végrehajtásra. Nézzük, hogy az ítéletlogikai követő állapot axiómák (mint a 11.1 egyenlet axiómája) mit mondanak az olyan cselekvések röör, melyek előfeltételei nem teljesülnek.
- Mutassa meg, hogy az axiómák azt jósolják, hogy semmi sem történik, amikor egy cselekvést olyan állapotban hajtunk végre, melyben az előfeltételei nem teljesülnek.
  - Vegyük egy  $p$  tervet, ami tartalmazza a cél eléréséhez szükséges cselekvéseket és illegális cselekvéseket is. Igaz, hogy ebben az esetben

$$kiinduló\ állapot \wedge követő\ állapot\ axiómák \wedge p \models cél$$

- Egy szituációkalkulusban elsőrendű követő állapot axiómák esetén (mint a 10. fejezetben) lehetséges bizonyítani, hogy egy illegális cselekvéseket tartalmazó terv elérje a célt?

- 11.16.** A repülőtér problémakörből adott példákkal magyarázza meg, hogy a szimbólum-szötválasztás hogyan csökkenti az előfeltétel és a cselekvéskizárási axiómák méretét. Származtasson egy általános képletet, mely az axiómahalmazok méretét adja meg az időlépések száma, a cselekvéssémák száma, ezek aritása, valamint a tárgyak száma függvényében.

- 11.17.** A 11.15. ábrán szereplő SATPLAN algoritmusban a minden kielégíthetőségi algoritmus hívás egy további  $g^T$  célt eredményez, ahol a  $T$  értékkészlete  $0 \dots T_{\max}$ . Tegyük fel, hogy ehelyett a kielégíthetőségi algoritmust csak egyszer hívjuk meg a  $g^0 \vee g^1 \vee \dots \vee g^{T_{\max}}$  céllal.
- Ez mindig ad vissza tervet, amennyiben létezik egy  $T_{\max}$  hosszúságú vagy annál rövidebb terv?
  - Eredményez ez a módszer új hibás megoldásokat?
  - Vizsgáljuk meg, hogy hogyan lehetne módosítani egy kielégíthetőségi algoritmust, például a WALKSAT algoritmust, hogy az (amennyiben létezik) rövid megoldásokat találjon, amennyiben hasonló diszjunktív célt kap.

# 12. TERVKÉSZÍTÉS ÉS CSELEKVÉS A VALÓ VILÁGBAN

*Ebben a fejezetben láthatjuk, hogy a kifejezőbb reprezentáció és az interaktívabb ágens-architektúrák hogyan vezetnek a valós világban is használható tervkészítőkhöz.*

Az előző fejezet a tervkészítés legalapvetőbb fogalmait, reprezentációt és algoritmusait vezette be. A való világ feladataihoz, mint a Hubble-űrteleszkóp megfigyeléseinek ütemezésére, gyárak működtetésére vagy hadműveletek logisztikai kezelésére jóval összetebben tervkészítőket használnak. Ezek kiterjesztik az alaptípusokat mind a reprezentációs nyelv, mind pedig a tervkészítés és a környezet együttműködésének szempontjából. Ez a fejezet bemutatja hogyan. A 12.1. alfejezet az idő- és erőforráskorláttal rendelkező tervkészítést mutatja be. A 12.2. alfejezet az előre definiált részterekhez történő tervkészítést írja le. A 12.3–12.6. alfejezet olyan ágensarchitektúrákat mutat be, melyeket a bizonytalan környezet kezelésére terveztek. A 12.7. alfejezet bemutatja, hogyan tervezünk, amikor a környezet más ágenseket is tartalmaz.

## 12.1. IDŐ, ÜTEMEZÉS ÉS ERŐFORRÁSOK

A STRIPS reprezentáció azt írja le, hogy *mit* csinálnak a cselekvések, de mivel a reprezentáció a szituációkalkuluson alapul, nem tartalmazza, hogy *milyen hosszú* egy cselekvés, sem azt, hogy *mikor* következik be, kivéve azt, hogy egy másik cselekvés előtt vagy után jön. Néhány feladatkörben azt szeretnénk tudni, hogy a cselekvések mikor kezdődnek és végződnek. A teherszállítási problémákban például, azt szeretnénk tudni, hogy egy adott csomagot hordozó repülőgép mikor érkezik, nem csak azt, hogy megérkezik, amikor a repülésnek vége.

Az alkalmazások egy általános családjának, az **ütemezési feladatok** (*job shop scheduling*) családjának a lényege az idő. Az ilyen feladatok munkák elvégzését igénylik, melyek mindegyike cselekvések sorozatából áll, ahol minden cselekvés adott időtartamú és bizonyos erőforrásokat igényelhet. A probléma, hogy egy olyan ütemezést határozzunk meg, ami – az erőforráskorlátok figyelembevétele mellett – minimalizálja az összes feladat elvégzéséhez szükséges összidőt.

A 12.1. ábrán egy ütemezési feladatra láthatunk példát. Ez egy nagyon leegyszerűsített autó-összeszerelési feladat. Két munkánk van, a  $C_1$  és a  $C_2$  autó összeszerelése. minden munka három cselekvésből áll: a motor beszerelése, a kerekek felszerelése, valamint az eredmény végső ellenőrzése. Először a motort kell berakni (mivel az első kerekek beszerelése megakadályozná, hogy hozzáférjünk a motortérhez), a vizsgálatot pedig értelemszerűen utoljára kell végrehajtani.

*Kiindulás(Karosszéria(C<sub>1</sub>)  $\wedge$  Karosszéria(C<sub>2</sub>)  
 $\wedge$  Motor(E<sub>1</sub>, C<sub>1</sub>, 30)  $\wedge$  Motor(E<sub>2</sub>, C<sub>2</sub>, 60)  
 $\wedge$  Kerekek(W<sub>1</sub>, C<sub>1</sub>, 30)  $\wedge$  Kerekek(W<sub>2</sub>, C<sub>2</sub>, 15))  
 Cél(Kész(C<sub>1</sub>)  $\wedge$  Kész(C<sub>2</sub>))*

*Cselekvés(MotorBeszerel(e, c),*

*ELŐFELTÉTEL: Motor(e, c, d)  $\wedge$  Karosszéria(c)  $\wedge$   $\neg$ MotorBenne(c)*

*KÖVETKEZMÉNY: MotorBenne(c)  $\wedge$  Időtartam(d))*

*Cselekvés(KerekekFelszerel(w, c),*

*ELŐFELTÉTEL: Kerekek(w, c, d)  $\wedge$  Karosszéria(c)  $\wedge$  MotorBenne(c)*

*KÖVETKEZMÉNY: KerekekRajta(c)  $\wedge$  Időtartam(d))*

*Cselekvés(Átvizsgáll(c),*

*ELŐFELTÉTEL: MotorBenne(c)  $\wedge$  KerekekRajta(c)  $\wedge$  Karosszéria(c)*

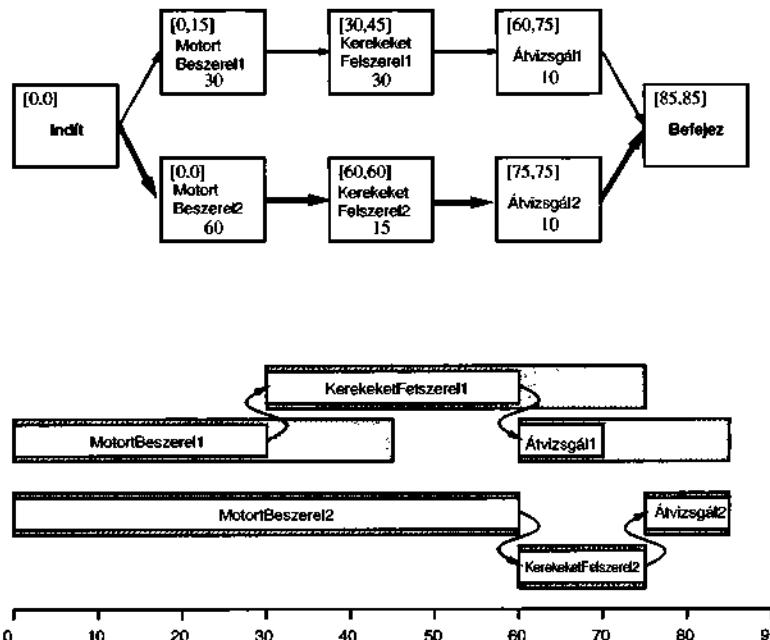
*KÖVETKEZMÉNY: Kész(c)  $\wedge$  Időtartam(10))*

**12.1. ábra.** A két autó összeszerelését tartalmazó ütemezési feladat. A jelölésben az *Időtartam(d)* jelentése, hogy egy cselekvés *d* percet vesz igénybe. A *Motor(E<sub>1</sub>, C<sub>1</sub>, 60)* jelentése az, hogy az E<sub>1</sub> egy motor, ami a C<sub>1</sub> alvázba illeszthető és 60 perc alatt szerelhető be.

A 12.1. ábrán látható probléma bármely eddig látott tervkészítővel megoldható. A 12.2. ábra (ha a számokkal nem törödünk) a részben rendezett tervkészítő megoldását mutatja. Hogy az inkább egy ütemezési, mintsem egy tervkészítési feladat legyen, minden cselekvéshez meg kell határoznunk, hogy mikor kezdődjön és mikor végződjön. Ez azt jelenti, hogy figyelnünk kell minden cselekvés hosszára, csakúgy, mint a sorrendjére. Az *Időtartam(d)* jelölés egy cselekvés kapcsán (ahol *d* csak egy számmal helyettesíthető be) azt jelenti, hogy egy cselekvés elvégzéséhez *d* percre van szükség.

Ha adott a cselekvések egy részleges rendezése az időtartamokkal, ahogy a 12.2. ábra is mutatja, akkor a **kritikus útvonal módszer (critical path method – CPM)** felhasználható az egyes cselekvések lehetséges kezdési és befejezési időpontjainak meghatározására. Egy részben rendezett tervben található útvonal, cselekvések egy lineárisan rendezett sorozata, ami az *Indít*-ból indul és a *Befejez*-ben fejeződik be (például a 12.2. ábra részben rendezett tervében két út található).

A **kritikus útvonal (critical path)** az az út, amelynek a teljes ideje a leghosszabb; az útvonal „kritikus”, mert meghatározza a teljes terv hosszát. Más utak rövidítése nem rövidíti a terv egészét, de a kritikus útvonalon lévő bármely cselekvés kezdetének elhalasztása a teljes tervet lassítja. Az ábrán a kritikus útvonal vonalát vastag vonallal jelöljük. Hogy a teljes tervet minimális idő alatt hajtsuk végre, a kritikus útvonal cselekvéseit úgy kell végrehajtani, hogy köztük ne legyen késleltetés. A kritikus útvonalon kívül eső cselekvéseknek van valamennyi mozgásterük – egy időablak, amelyben végrehajthatók. Ezt az időablakot a kezdeti időpont lehető legkorábbi (*ES*) és a lehető legkésőbbi (*LS*) értékével definiáljuk. Az *LS – ES* mérőszámot, a cselekvés mozgásterének (*slack*) nevezünk. A 12.2. ábrán láthatjuk, hogy a teljes terv 85 percet vesz igénybe, a kritikus útvonal minden cselekvésének a mozgástere 0 (ez mindenig igaz), és a C<sub>1</sub> összeszerelésének minden cselekvése egy 15 perces ablakban indítható. A cselekvések *ES* és *LS* idői a probléma **ütemtervét (schedule)** adják.



**12.2. ábra.** A 12.1. ábrán szereplő ütemezési probléma megoldása. Az ábra tetején a részben rendezett terv megoldása látható. minden cselekvés hozza a téglalap alján látható, a legkorábbi és a legkésőbbi kezdési időkkel ( $ES, LS$ ) a bal felső sarokban. A két szám között különbség a cselekvés mozgástere, a 0 mozgásterű cselekvések kritikus útvonalon vannak, melyeket vastag vonallal jelölünk. Az ábra alján ugyanezen megoldás időrendjét láthatjuk. A szürke téglalapok azt az intervallumot jelölik, ami alatt a cselekvés végrehajtható, feltételezve, hogy a sorrendezési megkötések betartjuk. A szürke téglalapok nem felhasznált, nem kitöltött területei a mozgásteret jelölik.

A következő kifejezések az  $ES$  és  $LS$  definícióiként szolgálnak, és egyúttal a kiszámításukra szolgáló dinamikus programozási algoritmust is körvonalazzák:

$$ES(Indít) = 0$$

$$ES(B) = \max_{A \prec B} ES(A) + Időtartam(A)$$

$$LS(Befejez) = ES(Befejez)$$

$$LS(A) = \min_{A \prec B} LS(B) - Időtartam(A)$$

Az ötlet az, hogy az  $ES(Indít)$  értékének 0-ra állításával indítunk. Ezután, amint elérünk egy olyan  $B$  cselekvést, hogy minden cselekvés, ami közvetlenül  $B$  előtt jön, már rendelkezik  $ES$  értékkal, az  $ES(B)$  értéket a közvetlenül megelőző cselekvések közül a legkorábbi befejezési idő maximumára állítjuk be, ahol a legkorábbi befejezési időt úgy definiáljuk, mint a cselekvés kezdési időpontja plusz a cselekvés időtartama. A folyamat addig folytatódik, amíg minden cselekvéshez egy  $ES$  értéket rendelünk. Az  $LS$  értékeket hasonlóképpen számítjuk ki a  $Befejez$  cselekvéstől visszafelé haladva. A résleteket egy későbbi feladat tartalmazza.

A kritikus útvonal algoritmus komplexitása csak  $O(Nb)$ , ahol  $N$  a cselekvések száma és  $b$  az egy cselekvésbe be-, illetve kimenő maximális elágazások száma. (Ennek megértéséhez vegyük észre, hogy az  $LS$  és  $ES$  számításokat minden cselekvésre csak egy-

szer hajtjuk végre, és minden számítás legfeljebb  $b$  másik cselekvésen meg végig.) Ezért, ha adott a cselekvések egy részben rendezése, a minimális időtartamú ütemezés megtalálása elég könnyedén elvégezhető.

## Ütemezés erőforráskorlátokkal

A valós ütemezési problémákat az **erőforrásokra** (**resources**) vonatkozó korlátozások jelenléte megbonyolítja. Például egy motornak az autóba szereléséhez egy motoremelőre van szükség. Ha csak egyetlen emelőnk van, akkor nem tudjuk egyszerre beszerelni az  $E_1$  motort, a  $C_1$  és az  $E_2$  motort a  $C_2$  autóba, így a 12.2. ábrán bemutatott ütemezés végrehajthatatlan. A motoremelő példa egy **újrahasznosítható erőforrásra** ( **reusable resource**). Az erőforrás foglalt, amíg a cselekvés tart, de újra használhatóvá válik annak befejezése után. Vegyük észre, hogy az újrahasznosítható erőforrásokat nem tudjuk a hagyományos előfeltétel és következmény alakú cselekvésleírásokkal kezelni, mert a rendelkezésre álló erőforrás-mennyisége nem változik, miután egy cselekvést lezártunk.<sup>1</sup> Ezért kiterjesztjük a leírásunkat, hogy egy ERŐFORRÁS:  $R(k)$ -t is tartalmazzon, amelynek jelentése, hogy  $k$  egység  $R$  típusú erőforrásra van szükség a cselekvéshez. Az erőforrás-szükséglet mind előfeltétel (a cselekvés nem hajtható végre, ha az erőforrás nem érhető el), mind pedig **átmeneti** következmény, abban az értelemben, hogy az  $R$  erőforrás rendelkezésre állása  $k$ -val csökken a cselekvés időtartama alatt. A 12.3. ábra mutatja, hogyan lehet a motorbeszerelés problémáját kiterjeszteni, hogy az három erőforrást is tartalmazzon: egy motoremelőt a motorok beszereléséhez, egy kerékállomást a kerekek felszereléséhez és két vizsgálóbiztost. A 12.4. ábra a legrövidebb szerelési idejű megoldást mutatja, ez 115 percet igényel. Ez hosszabb, mint az erőforrás-megkötések nélküli 85 perces ütemezés. Vegyük észre, hogy nincs olyan időpillanat, amikor minden vizsgálóbiztosra szükség van, ezért az egyik vizsgálóbiztost azonnal egy termelékenyebb posztra helyezhetjük.

Az erőforrások numerikus mennyiségekként történő leírása mint a *Vizsgálóbiztosok(2)*, a megnevezett entitások használata helyett, mint a *Vizsgálóbiztos( $I_1$ )* és a *Vizsgálóbiztos( $I_2$ )*, jó példa a nagyon általánosan használt aggregációs (**aggregation**) technikára. Az aggregáció központi ötlete, hogy egyedi objektumokat csoporthoz kötünk mennyiségekké, amikor maguk az objektumok nem megkülönböztethetők a cél szempontjából. Az összeszerelési problémánkban nem számít, hogy *melyik* vizsgálóbiztos vizsgálja az autót, ezért nincs szükség különbségtételre. (Ugyanez az ötlet működik a 3.9. feladat misszionáriusok és kannibálok problémájára.) Az aggregáció a komplexitás csökkentéséhez elengedhetetlen. Vizsgáljuk meg, mi történik, ha egy ütemezés 10 konkurrens *Vizsgálat* cselekvést tartalmaz, de csak 9 vizsgálóbiztos áll rendelkezésre. Ha a vizsgálóbiztosokat mint mennyiségeket reprezentáljuk, a bukást az algoritmus azonnal detektálja, és visszalép, hogy egy másik ütemezéssel próbálkozzon. Ha a vizsgálóbiztosokat önálló személyekként reprezentáljuk, akkor az algoritmus feleslegesen minden a  $10!$  vizsgálóbiztos-kiosztás kipróbálásához visszalép.

<sup>1</sup> Ezzel szemben a **fogyóeszköz-erőforrások** (**consumable resources**), mint az autó-összeszereléshez használt csavarok, az eredeti keretrendszeren belül kezelhetők; lásd 12.2. feladat.

*Kiindulás(Karosszéria(C<sub>1</sub>)  $\wedge$  Karosszéria(C<sub>2</sub>))*

- $\wedge$  Motor(E<sub>1</sub>, C<sub>1</sub>, 30)  $\wedge$  Motor(E<sub>2</sub>, C<sub>2</sub>, 60)
- $\wedge$  Kerekek(W<sub>1</sub>, C<sub>1</sub>, 30)  $\wedge$  Kerekek(W<sub>2</sub>, C<sub>2</sub>, 15)
- $\wedge$  MotorEmelő(1)  $\wedge$  KerékcsereÁllomás(1)  $\wedge$  Vizsgálóbiztos(2))

*Cél(Kész(C<sub>1</sub>)  $\wedge$  Kész(C<sub>2</sub>))*

*Cselekvés(MotortBeszerel(e, c, m),*

- ELŐFELTÉTEL:** Motor(e, c, d)  $\wedge$  Karosszéria(c)  $\wedge$   $\neg$  MotorBenne(c)
- KÖVETKEZMÉNY:** MotorBenne(c)  $\wedge$  Időtartam(d)
- ERŐFORRÁS:** MotorEmelő(1))

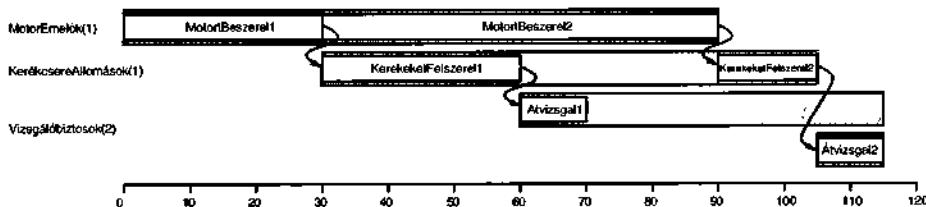
*Cselekvés(KerekeketFelszerel(w, c),*

- ELŐFELTÉTEL:** Kerekek(w, c, d)  $\wedge$  Karosszéria(c)  $\wedge$  MotorBenne(c)
- KÖVETKEZMÉNY:** KerekekRajta(c)  $\wedge$  Időtartam(d)
- ERŐFORRÁS:** KerékcsereÁllomás(1))

*Cselekvés(Átvízgátl(c),*

- ELŐFELTÉTEL:** MotorBenne(c)  $\wedge$  KerekekRajta(c)
- KÖVETKEZMÉNY:** Kész(c)  $\wedge$  Időtartam(10)
- ERŐFORRÁS:** Vizsgálóbiztos(1))

**12.3. ábra.** Két autó összeszerelésének ütemezési feladata erőforrásokkal. Egy összeszerelő állomás, egy kerékszerelő-állomás és két vizsgálóbiztos az összes elérhető erőforrás. Az **ERŐFORRÁS: r** jelölés jelentése, hogy az r erőforrást egy cselekvés végrehajtása alatt használjuk, de a cselekvés befejezése után újra szabad.



**12.4. ábra.** A 12.3. ábra erőforrásokat is tartalmazó ütemezési feladatának megoldása. Az ábra bal széle az erőforrásokat sorolja fel, a cselekvések az általuk használt erőforrásokkal egy sorban jelennek meg. Attól függően, hogy melyik összeszerelés használja a motorszerelő-állomást először, két lehetséges ütemezés létezik. Mi az optimális megoldást ábrázoltuk, ami 115 percet vesz igénybe.

Az előnyök ellenére, az erőforrás-megkötések nagyon megbonyolítják az ütemezési feladatokat azáltal, hogy a cselekvések közé további kölcsönhatásokat vezetnek be. Habár a megkötések nélküli ütemezés a kritikus útvonal módszer felhasználásával könnyű, a legkorábbi befejezési idejű erőforrás-megkötésekkel tartalmazó ütemezés megtalálása NP-nehéz. Ez a komplexitás gyakran minden gyakorlatban, minden az elméletben látható. Az 1963-ban közzétett nagy kihívást jelentő feladat – egy optimális ütemezés megtalálása egy 10 gépet, 10 munkát és 100 cselekvést tartalmazó problémához – 23 évig megoldatlan maradt (Lawler és társai, 1993). Sok megközelítést kipróbáltak, köztük az elágazós megkötést, a szimulált lehűtést, a tabu keresést, a kényszerkielégítést és számos más, a II. részben szereplő technikát. A legegyszerűbb, de népszerű heurisztika a **minimális tartalék (minimum slack)** algoritmus. Ez a cselekvésekkel mohó jelleggel ütemezi. minden iterációban megnézi azokat a cselekvéseket, amelyeknek az összes

előzményét már ütemeztük, és azt ütemezi, amelyiknek a legkevesebb tartalék ideje van a legkorábbi lehetséges kezdéshez. Ezután frissít az *ES* és *LS* időértékeket minden egyes érintett cselekvésre, és ezt ismétli.

A heurisztika ugyanazon az ötleten alapul, mint a legjobban korlátozott változó heurisztika a kényszerkielégítésnél. Ez a gyakorlatban gyakran jól működik, de a mi összeszerelési problémánkra egy 130 perces megoldást ad, nem pedig a 12.4. ábrán bemutatott 115 perceset.

Az ebben a fejezetben bemutatott megközelítés a „tervezünk először, ütemezünk később” megközelítés: azaz a teljes problémát egy *tervkészítési* fázisra és egy *ütemezési* fázisra bontjuk. A tervezési fázisban a cselekvéseket részben rendezzük, hogy elérjük a probléma céljait, az ütemezési fázisban pedig az időinformációkat illeszljük a tervhez, ezzel biztosítva, hogy a terv kielégítse az erőforrás- és a határidők korlátokat. Ez a megközelítés nagyon gyakori a valós idejű gyártási és logisztikai problémákban, ahol a tervkészítési fázist leggyakrabban emberi szakértők végzik. Ellenben, amikor sok erőforrás-megkötés van, adódhat, hogy néhány érvényes terv sokkal jobb ütemezéshez vezet, mint mások. Ebben az esetben, a tervkészítési és ütemezési fázist célszerű *integrálni* úgy, hogy a részben rendezett tervkészítés során figyelembe vesszük a cselekvések időtartamait és átfedéseit. A 11. fejezetben bemutatott számos tervkészítő algoritmus kiterjeszhető úgy, hogy ezt az információt is kezelje. Például a részben rendezett tervkészítők az okozati kapcsolatok konfliktusainak detektálásához hasonlóan az erőforráskorlát megsértéseit is detektálni tudják. A heurisztikák módosíthatók, hogy a cselekvések teljes költségén túl, a tervek teljes végrehajtási idejét is becsüljék. Ez napjainkban a kutatás egy aktív területe.

## 12.2. HIERARCHIKUS FELADATHÁLÓ TERVKÉSZÍTÉS

Az egyik legelterjedtebb módszer a komplexitás kezelésére a **hierarchikus dekompozíció** (*hierarchical decomposition*). Az összetett szoftvereket szubrutinok vagy objektum osztályok hierarchiájából építik fel, a hadseregek különböző egységek hierarchiái, a kormányok és cégek igazgatóságokból, osztályokból és alosztályokból állnak. A hierarchikus struktúra fő előnye, hogy a hierarchia minden szintjén egy számítási feladat, egy hadművelet vagy egy adminisztratív feladat az eggyel alatta lévő szint néhány cselekvésre épül, így a cselekvések megfelelő elrendezése a magasabb szinten lévő feladat megoldásához *kis* számítási költséggel jár. Másrészről a nem hierarchikus módszerek a feladatot *nagyszámú*, független cselekvésre bontják fel, ami nagy léptékű feladatok esetén egyáltalán nem praktikus. A legjobb esetben – amikor a magas szintű megoldásokhoz minden kielégítő alacsony szintű megvalósítás tartozik – a hierarchikus megoldások az exponenciális idejű tervkészítő algoritmusokkal szemben lineáris időre vezetnek.

Ez az alfejezet a **hierarchikus feladathálókon** vagy **HFH-kon** (*hierarchical task networks* – HTN) alapuló tervkészítési módszert mutatja be. A megközelítésünk ötvözi a részben rendezett tervkészítés (11.3. alfejezet) alapötleteit, illetve a „HFH-tervkészítés” területét. A HFH-tervkészítésben a kiinduló problémát, amely a feladatot írja le, a végrehajtandó feladat egy nagyon magas szintű leírásának tekintjük, például: építsünk egy házat. A terveket **cselekvésdekompozíciókkal** (*action decompositions*) finomítjuk. minden cselekvésdekompozíció a magas szintű cselekvést alacsonyabb szintű csele-

vések részben rendezett halmazára bontja. A cselekvésdekompozíció ezért a cselekvések megvalósítására vonatkozó ismereteket testesít meg. Például egy ház felépítése az engedélyek megszerzésére, a kivitelező megbízására, az építkezés elvégzésére és a kivitelező kifizetésére redukálható. (A 12.5. ábra egy ilyen dekompozícióra mutat példát.) A folyamat addig folytatódik, amíg csak az **egyszerű cselekvések (primitive actions)** maradnak a tervben. Az egyszerű cselekvések tipikusan azok a cselekvések, amelyeket az ágens automatikusan végre tud hajtani. Egy általános kivitelezőre a „kertépítés” egy egyszerű cselekvés lehet, mert egyszerűen csak egy kertépítő bevonását jelenti. Egy kertépítő számára azonban az olyan cselekvések tekinthetők egyszerűnek, mint az „ültessen rododendront ide”.

A „tiszta” HFH-tervkészítésben a terveket *csak* egymást követő cselekvésdekompozíciókkal állítjuk elő. A HFH ezért a tervkészítést a cselekvésleírások *konkretizálásának* tekinti, szemben az üres cselekvésből kiinduló cselekvésleírás *elkészítésének* folyamatával (ami az állapottér-keresésre és a részben rendezett tervkészítésre is igaz). Végezetül kiderül, hogy minden STRIPS cselekvésleírás egy cselekvésdekompozícióra írható át (lásd 12.6. feladat), és a részben rendezett tervkészítés a tiszta HFH-tervkészítés egy speciális esetének tekinthető. Bizonyos feladatokra azonban – különösen „szokatlan” konjunktív célokra – a tiszta HFH-nézőpont eléggé természetellenes, így egy *hibrid* megközelítést részesítünk előnyben, ahol a cselekvésdekompozíciókat mint a részben rendezett tervkészítés terfinomításait használjuk fel, a nyitott előfeltételek teljesítése és az ütközés feloldásra szolgáló rendezési megkölösek hozzáadása mellett. (Annak, hogy a HFH-tervkészítést a részben rendezett tervkészítés kiterjesztésének tekintjük, további előnye, hogy egy teljesen új jelölésrendszer bevezetése helyett ugyanaz használható.) Kezdetként a cselekvések dekomponálását mutatjuk be részletesebben, majd elmagyarázzuk, hogy a részben rendezett tervkészítést hogyan kell módosítani a dekompozíciók kezeléséhez. Végezetül a teljesség, a komplexitás és a használhatóság kérdéseit tárgyaljuk.

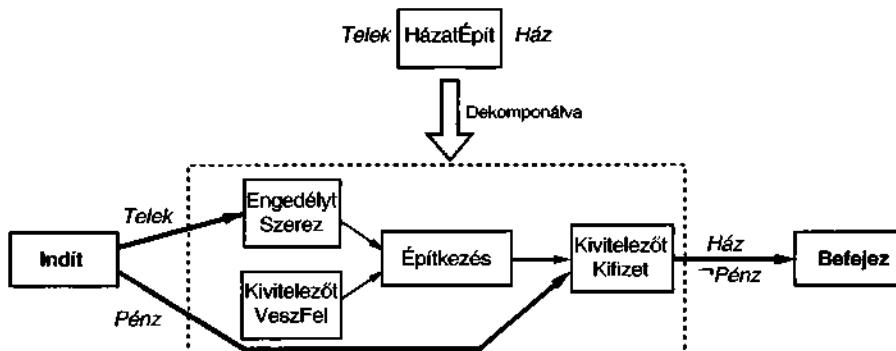
## A cselekvésdekompozíciók reprezentációja

A cselekvésdekompozíciós módszerek általános leírásait egy **tervkönyvtárban (plan library)** tároljuk, ahonnan kinyerhetők és a készülő terv igényeinek megfelelően felhasználhatók. minden módszer egy *Dekomponál(a, d)* formájú kifejezés, amelynek jelentése, hogy egy *a* cselekvés dekomponálható a *d* tervbe, mely egy – a 11.3. alfejezetben leírtaknak megfelelő – részben rendezett tervként van megadva.

A házépítés egy szép, konkrét példa, ezért ezt használjuk fel a cselekvésdekompozíció bemutatására. A 12.5. ábra a *HázatÉpít* cselekvés egy lehetséges dekompozíóját mutatja négy alacsonyabb szintű cselekvésre. A 12.6. ábra néhány cselekvés leírását tartalmazza erre a feladatkörre, valamint a házépítés dekompozíóját, ahogyan az a tervkönyvtárban megjelenne. A könyvtárban más lehetséges dekompozíciók is szerepelhetnek.

A dekompozíció *Indít* cselekvése szolgáltatja a terv cselekvéseinek összes olyan előfeltételét, melyet más cselekvés nem állít elő. Ezeket *külső előfeltételeknek (external preconditions)* nevezzük. Példánkban a dekompozíció külső előfeltételei a *Telek* és a *Pénz*. Hasonlóképp a *Befejez* előfeltételei a *külső következmények (external effects)*. Ezek a terv cselekvéseinek összes olyan következményét jelentik, melyeket más cselekvések nem negálnak. Példánkban a *HázatÉpít* külső következményei a *Ház* és a *¬Pénz*.

Néhány HFH-tervkészítő szintén különbözetet tesz az **elsődleges következmények** (primary effects), mint a *Ház*, és a **másodlagos következmények** (secondary effects), mint a  $\neg Pénz$  között. Mivel minden két típusú következmény felhasználása ütközésekhez vezethet más cselekvésekkel, csak az elsődleges következményeket használhatjuk a célok elérésére, ami nagymértékben csökkenti a keresési teret.<sup>2</sup>



12.5. ábra. A *HázatÉpít* cselekvés egy lehetséges dekompozíciója

A dekompozíciónak a cselekvés egy *korrekt* megvalósításának kell lennie. A *d* terv egy *a* cselekvést korrekten valósít meg, ha *d* egy teljes és konzisztens részben rendezett terv arra a problémára, ahol *a* előfeltételeiből *a* következményeit kell elérnünk. Ha a dekompozíció egy helyesen működő részben rendezett tervkészítő futásának eredménye, akkor nyilvánvalóan korrekt.

Bármely magas szintű cselekvéshez a tervkönyvtár számos dekompozíciót tartalmazhat, például a *HázatÉpít*-nek lehet egy másik dekompozíciója, amely a folyamatot úgy írja le, hogy a házat az ágens saját kezűleg építi kövekből és malterből. minden dekompozícióban egy korrekt tervnek kell lennie, de a magas szintű cselekvésleírásban szereplőkön túl további előfeltételeket és következményeket is tartalmazhat. Például a *HázatÉpít* 12.5. ábrán látható dekompozíciója a *Telek* mellett a *Pénz*-t is megköveteli, és következménye a  $\neg Pénz$ . Másról a saját kezű építés nem igényel pénzt, de szükséges hozzá egy felhasználásra váró *Kő* és *Malter* készlet, valamint eredménye lehet egy *FájósHáz*.

Mivel egy magas szintű cselekvésnek, mint amilyen a *HázatÉpít*, számos lehetséges dekompozíciója lehet, elkerülhetetlen, hogy a STRIPS cselekvésleírása elrejtse ezen dekompozíciók néhány előfeltételét vagy következményét. A magas szintű cselekvés előfeltételeit a dekompozícióban szereplő külső előfeltételek *metszete*, míg a következményt a dekompozíciók külső következményeinek metszete adja. Másképpen, a magas szintű előfeltételek és következmények garantáltan részhalmazai minden egyszerű implementáció valós előfeltételeinek és következményeinek.

<sup>2</sup> Ez néhány nem várt terv előállítását is megakadályozhatja. Például egy csődeljárás előtt álló személy minden ingó vagyonát eltüntetheti (a  $\neg Pénz$  elérésével) egy ház megvásárlásával vagy megépítésével. Ez a terv hasznos, mert az aktuális törvény kizárája az elsődleges lakóhely hitelezők általi lefoglalását (legalábbis az USA-ban – a szerk.).

Cselekvés(*TelketVásárol*, ELŐFELTÉTEL: *Pénz*, KÖVETKEZMÉNY: *Telek*  $\wedge$   $\neg$ *Pénz*)  
 Cselekvés(*KölcsöntVeszFel*, ELŐFELTÉTEL: *Hitel*, KÖVETKEZMÉNY: *Pénz*  $\wedge$  *Adósság*)  
 Cselekvés(*HázatÉpít*, ELŐFELTÉTEL: *Telek*, KÖVETKEZMÉNY: *Ház*)

Cselekvés(*EngedélytSzerez*, ELŐFELTÉTEL: *Telek*, KÖVETKEZMÉNY: *Engedély*)

Cselekvés(*KivitelezőtVeszFel*, ELŐFELTÉTEL: *Szerződés*)

Cselekvés(*Építkezés*, ELŐFELTÉTEL: *Engedély*  $\wedge$  *Szerződés*,  
 KÖVETKEZMÉNY: *HázFelépítve*  $\wedge$   $\neg$ *Engedély*)

Cselekvés(*KivitelezőtKifizet*, ELŐFELTÉTEL: *Pénz*  $\wedge$  *HázFelépítve*,  
 KÖVETKEZMÉNY:  $\neg$ *Pénz*  $\wedge$  *Ház*  $\wedge$   $\neg$ *Szerződés*)

Dekomponál(*HázatÉpít*,

*Terv*(LÉPÉSEK: { $S_1$ : *EngedélytSzerez*,  $S_2$ : *KivitelezőtVeszFel*,

$S_3$ : *Építkezés*,  $S_4$ : *KivitelezőtKifizet*})

RENDEZÉSEK: {*Indít*  $\prec$   $S_1$   $\prec$   $S_2$   $\prec$   $S_3$   $\prec$   $S_4$   $\prec$  *Befejez*, *Indít*  $\prec$   $S_2$   $\prec$   $S_3$ },

KAPCSOLATOK:  $\{Indít \xrightarrow[\text{Telek}]{\quad} S_1, Indít \xrightarrow[\text{Pénz}]{\quad} S_4,$

$S_1 \xrightarrow[\text{Engedély}]{\quad} S_3, S_2 \xrightarrow[\text{Szerződés}]{\quad} S_3, S_3 \xrightarrow[\text{HázFelépítve}]{\quad} S_4,$

$S_4 \xrightarrow[\text{Ház}]{\quad} Befejez, S_4 \xrightarrow[\text{\neg Pénz}]{\quad} Befejez\}))$

12.6. ábra. A házépítési probléma cselekvéseinek leírása és a *HázatÉpít* cselekvés részletes dekompozíciója. A leírások a pénzzel kapcsolatban egy egyszerűsített, az építőkkel kapcsolatban egy optimista nézetet alkalmaznak.

Az információ elrejtésének két másik formáját is meg kell említenünk. Először, a magas szintű leírás teljes mértékben figyelmen kívül hagyja a dekompozíciók összes belső következményét (internal effects). Például a *HázatÉpít* általunk javasolt dekompozíciójának ideiglenes belső következménye az *Engedély* és a *Szerződés*.<sup>3</sup> Másodszor, a magas szintű leírás nem specifikálja a cselekvésen belüli időintervallumot, ami alatt a magas szintű előfeltételeknek és következményeknek teljesülniük kell. Például a *Telek* előfeltételnek csak addig kell teljesülnie (a közelítő modellünkben), amíg az *EngedélytSzerez* nem hajtódi végre, és a *Ház* csak a *KivitelezőtKifizet* végrehajtása után igaz.

Az információ elrejtésének ez a módja szükségszerű, ha a hierachikus tervezés célja a komplexitás csökkenése. Szükséges, hogy a magas szintű cselekvések röviden anélkül dönthessünk, hogy a számtalan implementációs részlet miatt aggódnánk. Ennek azonban ára van. Konfliktusok léphetnek fel, például egy magas szintű cselekvés belső feltételei, valamint egy másik magas szintű cselekvés belső cselekvései között, miközben a magas szintű leírások alapján nincs mód ezek detektálására. Ennek a problémának komoly hatásai vannak a HFH-tervkészítő algoritmusokra. Dióhéjban, míg a tervkészítő algoritmus az atomi cselekvéseket, mint pontszerű eseményeket kezelheti, a magas szintű cselekvéseknek időbeli kiterjedése van, ami alatt minden mehet tovább.

<sup>3</sup> Az *Építkezés* negálja az *Engedély*-t, egyébként ugyanazon engedély több ház megépítéséhez is használható lenne. Sajnos az *Építkezés* nem zárja le a *Szerződés*-t, mert előbb a *KivitelezőtKifizet*-et kell végrehajtanunk.

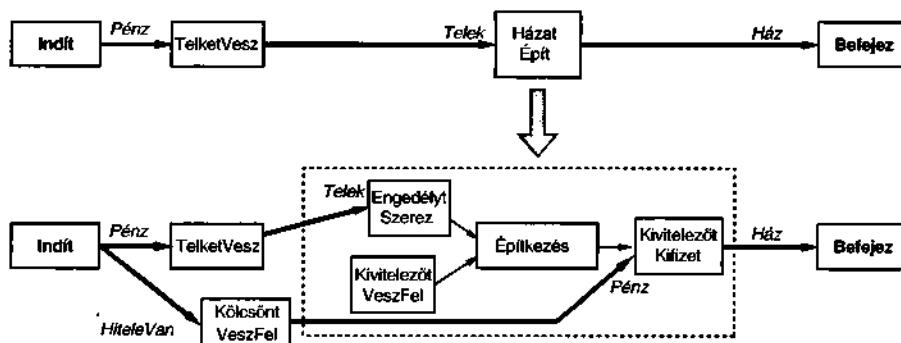
## A tervkészítő módosítása a dekompozíciók kezeléséhez

Most megmutatjuk, hogyan módosítható a részben rendezett tervkészítő, hogy egy HFH-tervkészítő is tartalmazzon. Ehhez a részben rendezett tervkészítő állapotátmennet-függvényét (lásd 468. oldal) módosítsuk úgy, hogy lehetővé tegye a  $T$  aktuális részleges terven a dekompozíciós módszerek alkalmazását. Az új következő terveket úgy alakítjuk ki, hogy először kiválasztunk néhány nem-atomi  $a'$  cselekvést a  $T$ -ból, majd a tervkönyvtár minden *Dekomponál*( $a, d$ ) metódusára, amelyben a  $\theta$  behelyettesítéssel az  $a$  és  $a'$  egyenlővé válik, az  $a'$ -t behelyettesítjük a  $d' = \text{SUBST}(\theta, d)$ -vel.

A 12.7. ábrán egy példa látható. Az ábra tetején egy ház megszerzésére vezető  $T$  terv található. Az  $a' = \text{HázatÉpít}$  magas szintű cselekvést választottuk a dekompozícióra. A 12.5. ábráról a  $d$  dekompozíciót választottuk, és a *HázatÉpít* cselekvést ezzel helyettesítettük. A dekompozíciós lépés által létrehozott Pénz nyitott feltétel teljesítésére a *KölcsöntVeszFel* járulékos lépést vezettük be. Egy cselekvés helyettesítése annak dekompozíójával egy kicsit hasonlít a szerváltultetéses műtétekhez. Az új résztervet ki kell bontanunk a csomagolásából (az *Indít* és *Befejez* lépések közül), be kell illesztenünk és minden megfelelően le kell zárnunk. Ezt többféleképpen is megtehetjük. Hogy pontosabban legyünk, minden lehetséges  $d'$  dekompozícióhoz az alábbiakat kell megtennünk:

1. Először is az  $a'$  cselekvést el kell távolítani a  $T$  tervből. Ezután a  $d'$  minden s lépésére ki kell választanunk egy cselekvést, amely kitölți az  $s$  szerepét, és hozzá kell adnunk a tervhez. Ez lehet az  $s$  egy új példányosítása vagy egy meglévő  $s'$  lépés a  $T$ -ból, ami  $s$ -sel azonos. A *Borkészítés* cselekvésdekompozíciója például megkövetelheti a *TelketVesz* cselekvést, de várhatóan használhatjuk ugyanazt a *TelketVesz* cselekvést, mely már szerepel a tervben. Ezt részfeladat-megosztásnak (*subtask sharing*) nevezzük.

A 12.7. ábrán nincsenek megosztási lehetőségek, ezért új cselekvéspéldányokat hoztunk létre. Ha egy cselekvést kiválasztottunk,  $d'$  minden belső előfeltételét átmásoljuk. Például az *EngedélytSzerez* cselekvés, az építkezés elő rendelt, és létezik egy okozati kapcsolat ezen lépések között, mely szerint az *Engedély* előfeltétele az *Építkezés-nek*. Ez lezárja az  $a'$  helyettesítését a  $d\theta$  példányosításával.



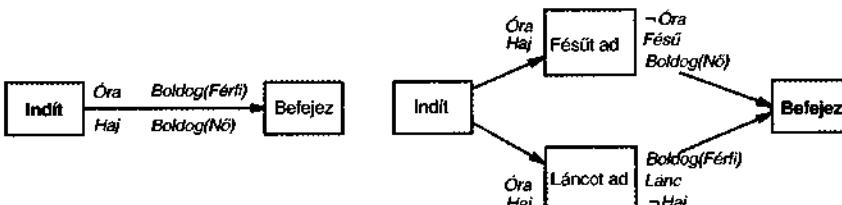
**12.7. ábra.** Egy magas szintű cselekvés dekompozíciója egy létező tervben. A *HázatÉpít* cselekvést a 12.5. ábra dekompozíciójával helyettesítjük. A *Telek* külső előfeltételt a már létező *TelketVesz*-ből kiinduló okozati kapcsolat biztosítja. A *Pénz* külső előfeltétel nyitott marad a dekompozíciós lépés után, ezért a *KölcsöntVeszFel* cselekvést szűrjuk be.

2. A következő lépés, hogy az eredeti tervben szereplő  $a'$  sorrendezési megkötéseit megfelelően átvezessük a  $d'$  lépéseihez. Először vegyük  $T B \prec a'$  alakú sorrendezési megkötéseit. Hogyan kellene  $B$ -t rendezni a  $d'$  lépéseinak megfelelően? A legkézenfekvőbb megoldás, hogy  $B$  az összes  $d'$ -ben szereplő lépés előtt jöjjön, amit úgy érhetünk el, hogy  $d'$  minden *Indít*  $\prec s$  alakú megkötését a  $B \prec s$  megkötéssel helyettesítjük. Másrészt ez a megközelítés túlzottan szigorú lehet! Például a *TelketVesz* a *HázatÉpít* előtt kell következzen, de nincsen szükség arra, hogy a *TelketVesz* a *KivitelezőtVeszFel* előtt jöjjön a kiterjesztett tervben. Egy túlzottan szigorú rendezés felállítása lehetetlenne teheti néhány megoldás megtalálását. Ezért minden rendezési megkötésre a legjobb megoldás, hogy rögzítsük a megkötés *okát*. Ha ezután egy magas szintű cselekvést kibontunk, az új rendezési megkötések a lehető legláborra vehetők az eredeti megkötés okával összhangban. Ugyanezek a megfontolások alkalmazhatók, amikor az  $a' \prec C$  alakú megkötéseket helyettesítjük.
3. Az utolsó lépés, hogy az okozati kapcsolatokat átvezessük. Ha a  $B \xrightarrow{r} d'$  az eredeti terv egy okozati kapcsolata volt, helyettesítsük azt egy okozati kapcsolathalmazzal, amely  $B$ -ból  $d'$  összes olyan lépéséhez vezet, amelynek előfeltétele a  $d$  dekompozíció *Indít* lépése által előállított  $p$  (például  $d'$  összes lépése, melyekre  $p$  egy külső előfeltétel). A példában a *TelketVesz*  $\xrightarrow{\text{Telk}} \text{HázatÉpít}$  okozati kapcsolatot a *TelketVesz*  $\xrightarrow{\text{Telk}} \text{Engedély}$  kapcsolattal helyettesítjük. (A dekompozícióban szereplő *KivitelezőtKifizet*, *Pénz* előfeltétele nyitott feltétellel válik, mert az eredeti tervben nincs olyan cselekvés, ami a *Pénz*-t biztosítja a *HázatÉpít*-hez.) Hasonlóképpen a terv összes  $a' \xrightarrow{r} C$  okozati kapcsolatát helyettesítsük  $d'$  bármely, a  $d$  dekompozíció *Vége* lépéseihez  $p$ -t szolgáltató lépésből  $C$ -be mutató okozati kapcsolathalmazzal (például  $d'$   $p$ -t külső következményként tartalmazó lépésből). Példánkban a *HázatÉpít*  $\xrightarrow{\text{Ház}} \text{Befejez}$  kapcsolatot a *KivitelezőtKifizet*  $\xrightarrow{\text{Ház}} \text{Befejez}$  kapcsolattal helyettesítjük.

Ez lezárja a részben rendezett tervkészítőknél alkalmazott, a dekompozíciók elkészítéséhez szükséges kiterjesztések.<sup>4</sup>

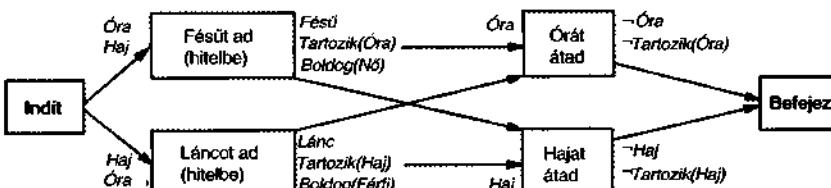
A részben rendezett tervkészítő algoritmushoz további módosítások szükségesek, mert a magas szintű cselekvések *elrejtik az információt* a végső elemi megvalósításukról. Nevezetesen az eredeti részben rendezett tervkészítő algoritmus hibával lép vissza, ha az aktuális terv feloldhatatlan ütközést tartalmaz, vagyis ha egy cselekvés ütközik egy okozati kapcsolattal, de nem helyezhető sem elé, sem pedig mögé. (Erre a 11.9. ábra mutat példát.) Másrészt, a magas szintű cselekvések esetén a feloldhatatlan ütközések néha feloldhatók az ütköző cselekvések *dekompozíciójával* és lépéseik összrendezésével. Erre a 12.8. ábra mutat be egy példát. Így előfordulhat olyan eset, ahol dekompozícióval egy teljes és konzisztens alapterv nyerhető, még akkor is, hogyha nem létezik teljes és konzisztensen magas szintű terv. Ez a lehetőség azt jelenti, hogy egy teljes HFH-tervkészítőnek végig kell tekintenie az eredeti részben rendezett tervkészítő-höz található számos metszési lehetőséget. Egyébként használhatjuk bármely metszési eljárást, remélve, hogy nem hagyunk figyelmen kívül lehetséges megoldást.

<sup>4</sup> Vannak további apró módosítások a magas szintű cselekvések konfliktusfeloldásának kezelésére. Az érdeklődő olvasó ezeknek a fejezet végén hivatkozott irodalmakban nézhet utána.



(a) Kiindulási feladat

(b) Absztrakt inkonzisztens terv



(c) A (b) dekompozíciója egy konzisztens megoldásra

**12.8. ábra.** Az O. Henrik történetből kiragadott *Mágusok ajándéka* probléma egy inkonzisztens absztrakt terveit mutat, ami azonban dekomponálható egy konzisztens megoldásra. Az (a) ábra a problémát mutatja be: egy szegény házaspának csak két értékes tulajdona van. A férfinek egy aranyórája, a nőnek pedig a gyönyörű hosszú haja. Mindketten azt tervezik, hogy ajándékot vásárolnak a másiknak, hogy az boldog legyen. A férfi úgy dönt, hogy az óráját egy ezüstfésüre cseréli be, míg a nő eladja a haját, hogy aranyláncot vegyen az órához. (Feltételezzük, hogy a „Fésüt ad” cselekvés előfeltétele a *Haj*, mivel ha a feleségnek nincs hosszú haja, a cselekvés nem éri el a kívánt hatást, hogy boldoggá tegye; és hasonlóképpen a „Láncot ad” cselekvésre.) A (b) ábrán szereplő részleges terv inkonzisztens, mert a „Fésüt ad” és „Láncot ad” absztrakt lépések nem sorrendezhetők konfliktus nélkül. (c) Dekomponáljuk a „Fésüt ad” lépést egy „beilleszt terv” metódussal. A dekompozíció első lépésében a férfi megszerzi a fésüt, és odaadja feleségének, miközben az órái egy későbbi időpontban adja oda fizetéséig. A második lépésben az órát átadja, és a kötelezettséget teljesíti. Egy hasonló módszer dekomponálja a „Láncot ad” lépést. Amíg mindenki odaadó lépést a szállítási lépés előre sorrendezzük, ez a dekompozíció megoldja a feladatot. (Vegyük észre, hogy ez azon múlik, hogy a lánc használata az órához vagy a fésü használata a hajhoz boldogságot okoz még akkor is, ha a tulajdonjogot már elveszítették.)

## Elemzés

Kezdjük a rossz hírrrel: a tiszta HFH-tervkészítés (ahol az egyetlen megengedett tervfinomítás a dekompozíció), a véges állapottér ellenére is, eldönthetetlen! Ez nagyon elszomorítónak tűnhet, mivel a HFH-tervkészítés lényege, hogy növeljük a hatékonyságot. Ez a nehézség azért lép fel, mert a cselekvésdekompozíciók **rekurzívak** (*recursive*) (például a sétálás implementálható egy lépés megtételével és utána egy sétálással), így a HFH-tervek tetszőlegesen hosszúra nyúlhatnak. Nevezetesen a legrövidebb HFH-megoldás is tetszőlegesen hosszú lehet, így nincs arra lehetőség, hogy a keresést egy megadott idő után leállítsuk. Ennek ellenére azonban legalább három okunk van a bizakodásra:

1. Kizáráhatjuk a rekurziót, amit csak nagyon kevés tervkészítési feladat igényel meg. Ebben az esetben az összes HFH-terv véges vagy megszámítható hosszságú.

2. Korlátozhatjuk azon megoldások hosszát, amelyekre kíváncsiak vagyunk. Mivel az állapottér véges, az olyan terv, amelynek több lépése van, mint az állapottér állapotainak száma, mindenkorban tartalmaz ciklust, ami ugyanabba az állapotba többször lép be. Az ilyen HFH-megoldások kizárással nagyon picit vesztünk, de uraljuk a keresést.
3. Létrehozhatunk egy hibrid megközelítést, ami a részben rendezett és a HFH-tervkészítőket kombinálja. A részben rendezett tervkészítés önmagában elegendő, hogy eldöntsük, létezik-e terv, így a hibrid feladat is nyilvánvalóan eldönthető.

A harmadik megoldással egy kicsit óvatosan kell bánnunk. A részben rendezett tervkészítő különböző módokon tud elemi cselekvéseket összehuzalozni, így olyan megoldásokkal találhatjuk magunkat szembe, amelyek nagyon nehezen érhetők, és amelyeknek nincs meg a HFH-tervek szép hierarchikus rendezése. Egy megfelelő kompromisszum, hogy a hibrid keresést úgy szabályozzuk, hogy a cselekvésdekompozíciót előnyben részesítjük az új cselekvések hozzáadása előtt, de nem olyan mértékben, hogy tetszőlegesen hosszú HFH-tervek jöjjenek létre, mielőtt egy elemi cselekvést hozzáadnánk. A megvalósítás egyik lehetséges módja egy költségfüggvény alkalmazása, amely engedményt ad a dekompozíció által bevezetett cselekvésekre. Minél nagyobb az engedmény, a keresés annál inkább a tisza HFH-tervkészítésre hasonlít, és a megoldás annál inkább hierarchikus. A hierarchikus tervek rendszerint sokkal könnyebben végrehajthatók a valós helyzetekben, és könnyebben javíthatók, ha valami elromlik.

A HFH-tervek egy másik fontos tulajdonsága a részfeladat-megosztás lehetősége. Emlékezzünk vissza, hogy a részfeladat-megosztás azt jelenti, hogy ugyanazt a cselekvést használjuk fel a tervdekompozícióban szereplő két különböző lépéshez. Ha megtiltjuk az alfeladat megosztását, akkor a  $d'$  dekompozíció minden példányosítása csak egyetlen módon hajtható végre, nem pedig sokféleképp, így jelentősen szűkítjük a keresési teret. Rendszerint ez a szűkítés időt spórol, és a legrosszabb esetben is az optimálisnál csak kicsivel hosszadalmasabb megoldásra vezet. Néhány esetben azonban több problémát is okozhat. Vegyük például az „élvezzük a mézesheteket, és neveljünk fel egy családot” célt. A tervkönyvtár a „házasodj és menj Hawaiiira” tervvel állhat elő az első részrétra és a „házasodj és legyenek gyermekid” tervvel a másodikra. A részfeladat megosztás nélkül a terv két különálló házasodás cselekvést tartalmaz, ami nagymértékben kerülendő.

Egy érdekes példa a részfeladat-megosztás költségeire és előnyeire a fordítók optimalizálásánál fordul elő. Vegyük a  $\tan(x) - \sin(x)$  kifejezés fordításának problémáját. A legtöbb fordító ezt két külön szubrutin triviális összeolvasztásával éri el: a  $\tan$  lépések a  $\sin$  lépések előtt következnek. Vegyük azonban a  $\sin$  és  $\tan$  alábbi Taylor soros közelítéseit:

$$\tan(x) \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} \quad \sin(x) \approx x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040}$$

Egy részfeladat-megosztást tartalmazó HFH-tervkészítő sokkal hatékonyabb megoldásokat készíthet, mert a  $\sin$  kiszámításának számos lépésére választhatja a  $\tan$  meglévő lépései. A legtöbb fordító nem alkalmazza ezt a fajta procedúrák közötti megosztást, mert az összes lehetséges megosztott terv figyelembevétele túl sok időt venne igénybe. Ehezlett a legtöbb fordító minden résztervet függetlenül készít el, majd esetleg módosítja az eredményt egy optimalizáló használatával.

A cselekvésdekompozíció által bevezetett járulékos bonyolítások tudatában miért hisszük mégis, hogy a HFH-tervkészítés hatékony lehet? A komplexitás valós forrásai a gyakorlatban nehezen analizálhatók, ezért vegyük egy idealizált esetet. Tegyük fel például, hogy egy  $n$  cselekvésből álló tervet akarunk létrehozni. Egy nemhierarchikus előrefelé haladó állapottér-tervező költsége, minden állapotban  $b$  megengedett cselekvéssel,  $O(b^n)$ . A HFH-tervkészítő esetén feltételezzünk egy nagyon általános dekompozíciós formát: minden nem elemi cselekvésnek,  $d$  lehetséges dekompozíciója van, mindegyik a következő alacsonyabb szinten  $k$  cselekvésből áll. Tudni szeretnénk, hogy ebben a struktúrában hány különböző dekompozíciós fa létezik. Ha a kiindulási szinten  $n$  cselekvés van, akkor a gyökér alatti szintek száma  $\log_k n$ , így a belső dekompozíciós csomópontok száma  $1 + k + k^2 + \dots + k^{\log_k n - 1} = (n - 1)/(k - 1)$ . minden belső csomópontnak  $d$  lehetséges dekompozíciója van, így összesen  $d^{(n-1)/(k-1)}$  lehetséges dekompozíciós fa alkotható. Ezt a képletet megvizsgálva láthatjuk, hogy  $d$  alacsonyan tartása nagy  $k$  mellett óriási megtakarításokat eredményezhet. Ha  $b$  és  $d$  összemérhető, akkor gyakorlatilag a nemhierarchikus költség  $k$ -adik gyökét vesszük. Másrészt azonban egy kisszámú, de hosszú dekompozíciót tartalmazó tervkönyvtár megalkotása, bár lehetővé teszi bármely probléma megoldását, nem minden lehetséges. Másképp mondva, a hosszú makrók, amelyek problémák széles körén alkalmazhatók, kiemelten értékesek.

Egy másik és talán jobb ok arra, hogy a HFH-tervkészítő hatékonyságában higgyünk, az, hogy a gyakorlatban működik. A nagyméretű alkalmazásokhoz használt tervkészítők majdnem mindegyike HFH-tervkészítő, mert a HFH-tervkészítés lehetőséget ad az emberi szakértő számára, hogy a komplex feladatok végrehajtásához szükséges elengedhetetlen tudást átadhassák, hogy a nagy tervek kicsi számítási ráfordítással elkészíthetők legyenek. Például a HFH-tervkészítési az ütemezéssel ötvözött O-PLAN-I (Bell és Tate, 1985) arra használták, hogy a Hitachi számára készítsen gyártási tervezet. Egy tipikus gyártási sor probléma 350 különböző terméket, 35 összeszerelő gépet és több mint 2000 különböző műveletet tartalmaz. A tervkészítő egy 30 napos ütemezést készít, naponta három 8 órás műszakkal és több millió lépéssel.

A HFH-tervkészítés kulcsa így egy tervkönyvtár megalkotása, amely a komplex magas szintű cselekvések megvalósításához tartalmazza az ismert módszereket. A könyvtár megalkotásának egyik módja, hogy a problémamegoldások tapasztalataiból tanulunk módszereket. Egy terv elkészítésének gyötrelmes tapasztalatai után az ágens elmentheti a tervet a könyvtárba mint egy a feladat (task) által definiált magas szintű cselekvés megvalósítási módját. Ily módon az ágens egyre kifinomultabbá válik, ahogy a régi módszerekre alapozva új módszerek épülnek. Ennek a tanulási folyamatnak egyik fontos szempontja, hogy az elkészített módszereket általánosítani tudja, a tervből csak a kulcslépéseket megtartva, azaz a probléma példányainak specifikus részleteit elhagyva (például az építő nevénk és címének elhagyása a tervezkről). Az ilyen általánosításra alkalmas módszereket a 19. fejezetben mutatjuk be. Számunkra elközelhetetlen, hogy hasonló mechanizmusok nélkül az emberek olyan kompetensek lehetnek, mint amilyenek.

## 12.3. TERVKÉSZÍTÉS ÉS CSELEKVÉS NEMDETERMINISZTIKUS PROBLÉMAKÖRÖKBEN

Eddig csak klasszikus tervkészítési (*classical planning*) környezeteket vettünk figyelembe, amelyek teljesen megfigyelhetők, statikusak és determinisztikusak. Mindezeken túl feltételeztük, hogy a cselekvésleírások megfelelők és teljesek. Ilyen körfüggvények között egy ágens először tervez, majd „becsukott szemmel” végrehajtja a tervet. Egy bizonytalan környezetben másrészről az ágensnek az érzékeit is fel kell használnia, hogy kövesse a történéseket a terv végrehajtása alatt, valamint szükség esetén módosítsa vagy helyettesítse a tervet, amennyiben valami váratlan dolog történik.

Az ágensnek meg kell küzdenie mind a *nem teljes, hiányos (incomplete)*, mind pedig a *nem megfelelő, hibás (incorrect)* információkkal. A nem teljesség abból következik, hogy a világ vagy csak részben megfigyelhető, vagy nemdeterminisztikus, vagy minden kettő. Például az irodaszerszékreny ajtaja vagy be van zárva, vagy nincs. Az egyik kulcsom vagy kinyitja az ajtót, vagy nem, már amennyiben be van zárva. Ráadásul ismereteim e hiányosságainak vagy tudatában vagyok, vagy nem. Ebből kifolyólag a vilagról alkotott modellem gyenge, de helyes. Másrészről a nem helyesség azért lép fel, mert a világ nem szükségszerűen felel meg a róla alkotott modelleknek. Például én *hihetem*, hogy a kulcsom kinyitja az irodaszerszékrent, de ebben tévedhetek is, ha a zárákat kicseréltek. A helytelen információ kezelési képességének hiányában az ágens olyan buta lehet, mint egy ganajtúró bogár (71. oldal), ami még az után is megpróbálja a ganajgolyót bedugni a fészkébe, hogy a golyót eltávolították a fogásából.

A teljes és korrekt ismeretek megszerzésének lehetősége attól függ, hogy mennyi nemdeterminisztikusság van a világban. **Korlátos nemdeterminisztikusság (bounded indeterminacy)** esetén a cselekvéseknek megjósolhatatlan következményei lehetnek, de a lehetséges következményeket felsorolhatjuk a cselekvésleíró axiómákban. Például amikor feldobunk egy pénzt, jogosan feltételezhetjük, hogy a kimenetele vagy fej, vagy írás lesz. Egy ágens kezelheti a korlátos nemdeterminisztikusságot úgy, hogy a terveket az összes lehetséges körfüggvényre felkészíti. **Nem korlátos nemdeterminisztikusság (unbounded indeterminacy)** esetén azonban az előfeltételek vagy hatások halma-za vagy nem ismert, vagy túl nagy ahhoz, hogy kimerítően felsoroljuk. Ez az eset áll fenn a nagyon összetett vagy dinamikus problémakörökben, mint amilyenek például az autóvezetés, a gazdaságos tervezés vagy a hadműveletek. Egy ágens csak úgy kezelheti a nem korlátos nemdeterminisztikusságot, ha felkészítjük arra, hogy átdolgozza a terveit és/vagy a tudásbázisát. A nem korlátos nemdeterminisztikusság közeli rokonságban van a 10. fejezetben tárgyalta **kvalifikációs problémával (qualification problem)** (az összes előfeltétel felsorolásának lehetetlenségével, amire egy valódivilág-beli cselekvésnek szüksége van, hogy a szándékolt hatást elérje).

A nemdeterminisztikusság kezelésére négy tervkészítő módszer létezik. Az első kettő a korlátos nemdeterminisztikusság, a második kettő pedig a nem korlátos nemdeterminisztikusság esetén használható:

- **Érzékelőmentes tervkészítés (sensorless planning):** ez a más néven alkalmazkodó tervkészítésnek (*conformant planning*) nevezett módszer normál, szekvenciális terveket készít, melyeket érzékelés nélkül kell végrehajtani. Az érzékelésmentes terv-

készítő algoritmusnak biztosítania kell, hogy a terv a célt minden lehetséges körülmeny között elérje függetlenül a valós induló állapottól és a cselekvések aktuális kimeneteleitől. Az érzékelésmentes tervkészítés a **kényszerítésen** (**coercion**) alapul, alapgondolata, hogy a világ erőszakkal egy megadott állapotba vihető még akkor is, ha az ágensnek csak részleges információi vannak az aktuális állapotról. A kényszerítés nem minden lehetséges, így az érzékelésmentes tervkészítés gyakran alkalmazhatatlan. Az érzékelőmentes feladatmegoldást, amely magában foglalja a hiedelem állappontban történő keresést, a 3. fejezetben mutattuk be.

- **Feltételes tervkészítés (conditional planning):** a más néven eshetőségi tervkészítés (**contingency planning**) módszere a korlátos nemdeterminizmust kezeli úgy, hogy egy feltételest tervet készít, amely a különböző eshetőségekhez különböző ágakat tartalmaz. Csakúgy, mint a klasszikus tervkészítésben, az ágens először tervez, majd végrehajtja az elkészített tervet. Az ágens érzékelő cselekvéseket épít a tervbé, hogy ellenőrizze a megfelelő feltételeket, így meghatározhatja, hogy a terv mely részét hajtsa végre. A légi szállítási feladatkörben például olyan terveink lehetnek, amelyek a következőket mondják: „ellenőrizd, hogy az SFO repülőtér használható-e. Ha igen, repülj oda; egyébként repülj Oaklandba”. A feltételeles tervkészítéssel a 12.4. alfejezet foglalkozik.
- **Végrehajtás monitorozás és újratervezés (execution monitoring and replanning):** ebben a megközelítésben az ágens bármely ezt megelőző tervkészítő technikát felhasználhat (klasszikus, érzékelőmentes vagy feltételes), de használ egy végrehajtás monitorozást (**execution monitoring**), amely eldönti, hogy a terv használható-e az aktuális állapotban, vagy újra kell gondolni. **Újratervezés (replanning)** akkor történik, amikor valami hiba lép fel. Ily módon az ágens a nem korlátos nemdeterministikuságot is kezelní képes. Például ha az újratervező ágens nem láttá előre az SFO lezárásának lehetőségét, észreveheti ezt a szituációt, amikor fellép, és a tervkészítő által új útvonalat találhat a célohoz. Az újratervező ágenseket a 12.5. alfejezetben tárgyaljuk.
- **Folytonos tervkészítés (continuous planning):** minden eddig látott tervkészítő arra készült, hogy elérje a célt, és megálljon. A folytonos tervkészítő egy életen keresztüli működésre van tervezve. Kezelní képes a környezetben fellépő nem várt körülményeket még akkor is, ha azok az ágenst egy tervkészítés közepén érik. A **célújraformálás (goal formulation)** által képes kezelní a célok elhagyását, illetve új célok keletkezését is. A folytonos tervkészítést a 12.6. alfejezetben tárgyaljuk.

Vegyük egy példát, hogy megvilágítsuk a különböző ágensek közötti különbségeket. A probléma a következő: a kiinduló állapotban adott egy szék, egy asztal és néhány doboz festék. A színe semminek sem ismert, és olyan állapotot kell elérni, ahol az asztal és a szék színe azonos.

A **klasszikus tervkészítő** ágens nem tudja kezelní ezt a problémát, mert a kiinduló állapot nem teljesen specifikált, azaz nem tudjuk, hogy milyen színűek a bútorok.

Az **érzékelőmentes tervkészítő** ágensek egy olyan tervet kell találni, ami érzékelés nélkül működik a terv végrehajtása során. A megoldás, hogy bármelyik festékesdobozt nyitjuk ki, a festéket mind az asztalra, mind a székre alkalmazzuk. Így **kényszerítjük (coercing)** őket, hogy azonos színűek legyenek (még akkor is, ha az ágens nem tudja, hogy ez milyen szín). A kényszerítés akkor helyénvaló, ha az előzetes információk

gyűjtése drága vagy lehetetlen. Például az orvosok gyakran egy széles spektrumú antibiotikumot használnak ahelyett, hogy egy feltételes tervet készítenék, ami egy vérvizsgálatot hajt végre, megvárja az eredményeket, és utána egy specifikus antibiotikumot alkalmaz. Ezt az indokolja, hogy a vérteszt rendszerint nagy költséggel és késedelemmel jár.

A **feltételes tervkészítő** ágens jobb tervet készíthet: először megvizsgálja az asztal és a szék színét, majd ha ezek már eleve azonosak, a terv kész. Ha nem, megnézi a festékesdobozok címkéit, ha ezek között talál olyan színűt, mint amilyen a színe bármelyik bútordarabnak, akkor ezt a festéket alkalmazza a másikon. Egyébként befesti minden darabot bármelyik színnel.

Az **újratervező** ágens a feltételes tervkészítővel azonos tervet készíthet vagy elsőre egy jóval kevesebb ágat tartalmazót, majd amennyiben szükséges, a végrehajtási időben tölti fel a többöt. A cselekvésleírások nem megfelelőségét szintén kezelni tudja. Például tegyük fel, hogy a *Fest(tárgy, szín)* cselekvésnek a *Szín(tárgy, szín)* deterministikus hatást tulajdonítjuk. Egy feltételes tervkészítő feltételezi, hogy a következmény teljesül, ha a cselekvést végrehajtottuk, de az újratervező ágens ellenőrizheti a hatást és ha az nem igaz (talán mivel az ágens figyelmetlen volt és kihagyott egy részt), újra tervezhet és újra festhet egy részt. Ehhez a példához az 520–521. oldalon visszatérünk.

A **folytonos tervkészítő** ágens a nem várt események kezelésén túl újra is tervezhet, ha mondjuk a „legyen ebéd az asztalon” új célt illesztjük be, azaz a festési tervet el kell halasztani.

A valódi világban az ágensek a különböző megközelítések kombinációit használják. Az autógyártók pótkereket és tartalék légszákokat árulnak, amelyek fizikai megtestesítői a feltételes tervágaknak, melyeket a defektek vagy az ütközések kezelésére készítettek. Másrészről a legtöbb autóvezető soha nem veszi figyelembe ezeket a lehetőségeket, így a defektekre vagy az ütközésekre mint újratervező ágensek reagálnak. Általánosságban az ágensek csak azokra az eshetőségekre készítenek feltételes tervet, amelyeknek fontos hatásai vannak, és nem elhanyagolható eséllyel okoznak hibát. Így egy autóvezetőnek, aki a Szaharán keresztül kíván utazni, igencsak figyelembe kell vennie a lerobbanás lehetőségét, míg egy áruházba vezető út jóval kevesebb megelőző tervezést igényel.

Az ebben a fejezetben bemutatott ágensek a nemdeterministikusság kezelésére készültek, de nem alkalmasak arra, hogy mérlegeljenek a sikeres valószínűsége és a terv elkészítésének költsége tekintetében. A 16. fejezet további eszközöket biztosít az ilyen problémák kezelésére.

## 12.4. FELTÉTELES TERVKÉSZÍTÉS

Feltételes tervkészítés a bizonytalanság kezelésének egy módja. Módszere, hogy a terv előre meghatározott pontjain ellenőrzi, hogy valójában mi is történik a környezetben. A feltételes tervkészítés bemutatása a teljesen megfigyelhető környezetek esetén a legegyszerűbb, így ezzel az esettel kezdünk. A részlegesen megfigyelhető eset jóval nehezebb, de egyben jóval érdekesebb is.

## Feltételes tervkészítés teljesen megfigyelhető környezetekben

A teljes megfigyelhetőség jelentése, hogy az ágens mindenkorban ismeri az aktuális állapotot. Nem determinisztikus környezet esetén azonban, az ágens nem képes megjósolni a cselekvéseinak *kimenetelét*. A feltételes tervkészítő ágens a nem determinisztikusságot úgy kezeli, hogy a tervekbe feltételeket épít be (tervkészítési időben), amelyek ellenőrzik a környezet állapotát (futási időben), hogy a továbbiakról dönthessen. A kérdezés tehát, hogyan készíthetők el az ilyen feltételes tervezetek.

Példaként a **porszívóvilág** (*vacuum word*) problémakört használjuk, amelynek állapotterét a determinisztikus esetre a 103. oldalon fektettük le. Emlékezzünk vissza, hogy a *Balra*, a *Jobbra* és a *Szív* a rendelkezésre álló cselekvések. Szükségünk lesz néhány propozícióra, hogy definiáljuk az állapotokat: legyen az *OttBal* (*OttJobb*) igaz, ha az ágens a bal (jobb) állapotban van, és legyen a *TisztaBal* (*TisztaJobb*) igaz, ha a bal (jobb) állapot tiszta.<sup>5</sup> Az első feladatunk a STRIPS nyelv kiterjesztése, hogy megengedje a nem determinisztikusságot. Ennek érdekében megengedjük a cselekvésekben a **diszjunktív következményeket** (*disjunctive effects*), ami azt jelenti, hogy egy cselekvést bármikor végrehajtva annak kettő vagy több különböző kimenetele is lehet. Tegyük fel például, hogy a *Balra* lépés néha sikertelen. Ekkor a

*Cselekvés(Balra, Előfeltétel: OttJobb, Következmény: OttBal  $\wedge$   $\neg$ OttJobb)*  
normál cselekvésleírást módosítanunk kell, hogy diszjunktív következményt is tartalmazzon:

*Cselekvés(Balra, Előfeltétel: OttJobb, Következmény: OttBal  $\vee$  OttJobb)* (12.1)  
Szintén hasznosnak találjuk a **feltételes következményeket** (*conditional effects*), amikor is egy cselekmény következménye függ attól az állapottól, amelyben végrehajtjuk. A feltételes következmények a cselekvés KÖVETKEZMÉNY részében jelennek meg a „when <feltétel>: <következmény>” szintaxisával. Például a *Szív* cselekvés modellezésére a

*Cselekvés(Szív, Előfeltétel:, Következmény:(when OttBal: TisztaBal)  
 $\wedge$  (when OttJobb: TisztaJobb))]*

Kifejezést írnánk fel. A feltételes következmények nem vezetik be a nem determinisztikusságot, de segítséget nyújtanak annak modellezésében. Tegyük fel például, hogy egy körmönfont porszívónk van, ami néha, ha mozog, piszkot szór a célnégyzetre, de csak akkor, ha az tiszta. Ez a

*Cselekvés(Balra, Előfeltétel: OttJobb, Következmény: OttBal  
 $\vee$  (OttBal  $\wedge$  (when TisztaBal:  $\neg$ TisztaBal)))*

leírással modellezhető, ami mindenkorban diszjunktív, mindenkorban pedig feltételes.<sup>6</sup> Hogy feltételes tervezet készíthessünk, **feltételes lépésekre** (*conditional steps*) van szükségünk. Ezeket az „if <teszt> then terv\_A else terv\_B” szintaxis használatával írjuk le, ahol a <teszt> egy

<sup>5</sup> Nyilvánvalóan az *OttJobb* akkor és csak akkor igaz, ha a  $\neg$ OttBal igaz, és fordítva. A két állítás használatanak oka főként az olvashatóság javítása.

<sup>6</sup> A when *TisztaBal:  $\neg$ TisztaBal* feltételes következmény egy kicsit furcsának tűnhet. Emlékezzünk vissza azonban, hogy itt a *TisztaBal* a cselekvés előtti, míg a  $\neg$ *TisztaBal* a cselekvés végrehajtása utáni helyzetre vonatkozik.

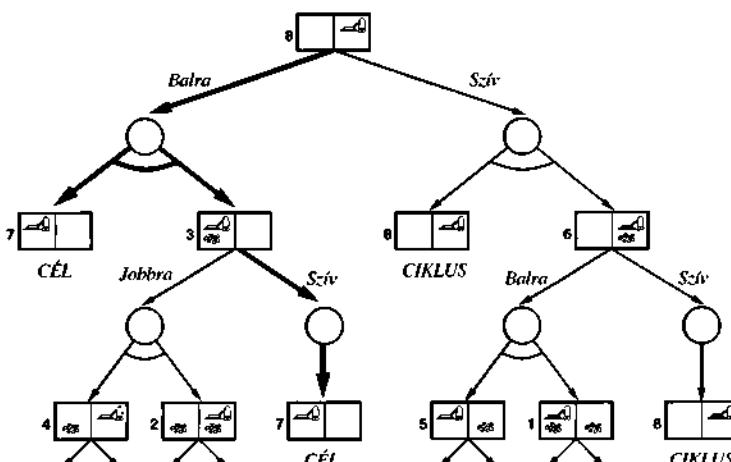
kétertétekű függvénye az állapotváltozóknak. Az „if *OttBal*  $\wedge$  *TisztaBal* then *Jobbra* else *Szív*” például a porszívóvilág egy feltételes lépése lehet. Egy ilyen lépés végrehajtása a készítőnek módosításra szorosan kötött. A feltételes lépések egymásba ágyazásával a tervez fák lesznek.

A feltételes tervektől elvárjuk, hogy működjenek, függetlenül attól, hogy valójában a cselekvés mely kimenetele következik be. Ezzel a problémával egy másik köntösbe bujtatva már találkoztunk korábban. A kétszemélyes játékokban (lásd 6. fejezet) olyan lépéseket szeretnénk, amelyek az ellenfél lépéseihez függetlenül győzelemhez vezetnek. A nemdeterminisztikus tervkészítési problémákat ezért gyakran természet elleni játékoknak (*games against nature*) nevezik.

Vegyük a porszívóvilág egy speciális példáját. A kiinduló állapotban a robot a tiszta világ jobb oldali négyzetén van. Mivel a környezet teljesen megfigyelhető, az agens ismeri a teljes *OttJobb*  $\wedge$  *TisztaBal*  $\wedge$  *TisztaJobb* állapotleírást. A célállapotban a robot a tiszta világ bal oldali négyzetén van. Ez a feladat elég triviális lenne, ha nem a „dupla-Murphy” porszívóval lenne dolgunk, amely néha piszkot hagy maga után, amikor egy tiszta célnégyzetre lép, és néha bepiszkolja a tiszta négyzetet, ha a Szívás cselekvés végrehajtódik.

Ennek a környezetnek a „játékfáját” a 12.9. ábrán mutatjuk be. A cselekvéseket a robot a fa „állapot” csomópontjaiban hajtja végre, majd a körrel jelölt „valószínűségi” csomópontokban a természet dönt a cselekvés kimeneteléről. A megoldás egy részfa, mely (1) minden levelében egy cél csomópontot tartalmaz, (2) minden „állapot” csomóponthoz egy cselekvést specifikál, és (3) minden „valószínűségi” csomópontban tartalmazza az összes kimenetelhez tartozó ágat. Az ábrán a megoldást vastag vonallal jelöltük, ami a [*Balra*, if *OttBal*  $\wedge$  *TisztaBal*  $\wedge$  *TisztaJobb* then [] else *Szív*] tervnek felel meg. (Mivel állappoter-tervkészítőt használunk, a feltételes lépésekben használt tesztek egyelőre teljes állapot leírások.)

A játékok pontos megoldásához a **minimax algoritmust** használjuk (lásd 6.3. ábra). Ehhez a feltételes tervkészítésben tipikusan két módosítás tartozik. Először is, a MAX és



**12.9. ábra.** A „dupla-Murphy” porszívóvilág keresési fájának első két szintje. Az állapotcsomópontokban és a VAGY csomópontokban cselekvéseket kell választani. A valószínűségi csomópontok, amelyeket körökkel jelöltünk és csomópontok, ahol, ahogy azt a kimenő ágakon szereplő ív is jelöli, minden kimenetelt kezelni kell. A megoldást vastag vonallal jelöltük.

a MIN csomópontok VAGY és És csomópontokká válhatnak. Nyilvánvalóan a tervnek minden elérő állapotban választania kell *valamelyen* cselekvést, de kezelnie kell ezen cselekvés minden kimenetelét. Másodsor, az algoritmusnak nemcsak egy lépést kell megadnia, hanem egy feltételes tervet kell készítenie. Egy VAGY csomópontban a terv egyszerűen a választott cselekvés, amelyet bármír követhet. Egy És csomópontban a terv *if-then-else* lépések egymásba ágyazott sorozata, melyek minden lehetséges kimenetelhez egy résztervet adnak meg. Ezen lépésekben a feltétel vizsgálatokban teljes állapotleírások szerepelnek.<sup>7</sup>

```

function És-VAGY-GRÁF-KERESÉS(feladat) returns egy feltételes terv vagy kudarc
  VAGY-KERESÉS(KINDULÓ-ÁLLAPOT[feladat], feladat, [ ])
```

---

```

function VAGY-KERESÉS(állapot, [feladat], feladat, [ ]) returns egy feltételes terv vagy kudarc
  if CÉL-TESZT[feladat](állapot) then return üres terv
  if állapot az útvonal-on then return kudarc
  for each cselekvés, állapot_halmaz in KÖVETŐÁLLAPOTOK[feladat](állapot) do
    terv  $\leftarrow$  És-KERESÉS(állapot_halmaz, feladat, [állapotútvonal])
    if terv  $\neq$  kudarc then return [cselekvés|terv]
  return kudarc
```

---

```

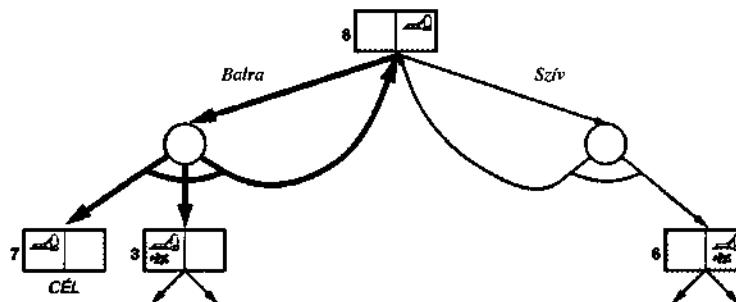
function És-KERESÉS(állapot_halmaz, feladat, terv) returns egy feltételes terv vagy kudarc
  for each si in állapot_halmaz do
    tervi  $\leftarrow$  VAGY-KERESÉS(si, feladat, terv)
    if terv = kudarc then return kudarc
  return [if s1 then terv1 else if s2 then terv2 else ... if sn-1 then tervn-1 else tervn]
```

**12.10. ábra.** Egy algoritmus a nemdeterminisztikus környezetek által generált És-VAGY gráfok keresésére. Feltételezzük, hogy az ÁLLAPOTÁTMENET függvény a cselekvések egy listáját adja vissza, melyek minden egyike egy lehetséges kimenetel halmazhoz tartozik. A cél egy feltételes terv megtalálása, ami bármilyen körülmenye között elér egy célállapotot.

Formálisan az eddig definiált keresési tér egy És-VAGY gráf. Az És-VAGY gráfok korábban a 7. fejezetben az ítéletlogikai Horn-kláz következtetésben jelentek meg. Itt az ágak logikai következtető lépések helyett cselekvések, de az algoritmus azonos. A 12.10. ábra az És-VAGY gráfok keresésére egy rekurzív, mélyiségi kereső algoritmust ad meg.

A bemutatott algoritmusban kulcsfontosságú a nemdeterminisztikus tervkészítési problémákban gyakran felmerülő ciklusok kezelésének módja (például ha egy cselekvésnek néha nincs következménye vagy egy helytelen következmény kijavítható). Ha az aktuális állapot azonos a gyökértől idáig vezető útvonal egy állapotával, akkor hibával tér vissza. Ez nem jelenti, hogy *nincs* megoldás az aktuális állapotból, egyszerűen annyit jelent, hogy *van* egy nemciklikus megoldás, ami elérhető az aktuális állapot korábbi előfordulásából, így az állapot újabb bekövetkezése kihagyható. Ezzel az ellenőrzéssel biztosítjuk, hogy az algoritmus minden véges állapottér esetén leálljon, mivel minden útvonal célit ér, zsákutcába jut, vagy egy állapot ismétlése. Vegyük észre, hogy az algoritmus nem ellenőrzi, hogy az aktuális állapot egy másik útvonalon szereplő állapot ismétlése-e. A 12.15. feladat ezt a kérdést járja körül.

<sup>7</sup> Az ilyen terveket case szerkezettel is leírhatnánk.



**12.11. ábra.** A „tripla-Murphy” porszívóvilág keresési gráfjának első szintje, ahol a ciklusokat explicit megjelöltük. A probléma összes megoldása ciklikus terv.

Az ÉS-VAGY-GRÁF-KERESÉS által visszaadott tervek feltételeket tartalmazznak, melyek a teljes állapotleírást megvizsgálják, hogy egy agról döntsenek. A legtöbb esetben ennél jóval kevésbé kimerítő ellenőrzésekkel is megúszhatjuk. Például a 12.9. ábrán látható megoldás a [Balra, if TisztaBal then [] else Szív] megadással egyszerűen leírható. Ennek oka, hogy a TisztaBal teszt elegendő, hogy az ÉS csomópont állapotait két egyelemű halmazba sorolja úgy, hogy a tesztek után az ágens pontosan ismerje az állapotát. Valójában az egy változós if-then-else tesztek sorozata minden elegendő, hogy állapotok egy halmazát egyelemű halmazokra ossza, *feltéve*, hogy az állapot teljesen megfigyelhető. Ezért az általanosság teljes megőrzése mellett a teszteket egy változós tesztekre szűkíthetjük.

Az utolsó nehézség, ami gyakran felmerül a nemdeterminisztikus feladatkörökben, a következő: a dolgok nem mindenkor működnek előre, így újra kell próbálkozni. Vagyük például a „tripla-Murphy” porszívó példáját, mely (a korábban bemutatott szokások mellett) néha nem mozdul az utasítás ellenére. Például csakúgy, mint a (12.1) egyenletben, a Balra cselekvés tartalmazhatja a OttBal  $\vee$  OttJobb diszjunktív hatást. Ekkor a [Balra, if TisztaBal then [] else Szív] terv már nem garantált, hogy működik. A 12.11. ábra a keresési gráf egy részletét mutatja. Tisztán látható, hogy a továbbiakban nincsenek ciklusmentes megoldások, és az ÉS-VAGY-GRÁF-KERESÉS hibával térne vissza. Létezik azonban egy ciklikus megoldás (*cyclic solution*), ami addig próbálhatja a Balra lépést, míg nem egyszer működik. Ez a megoldás könnyebben kifejezhető, ha a terv egy részét egy címkelvel (*label*) jelöljük meg, és a terv ismételgetése helyett erre a címkeré hivatkozunk. Így a ciklikus megoldásunk

[ $L_1 : \text{Balra, if OttJobb then } L_1 \text{ else if TisztaBal then [] else Szív}$ ]

alakú. (A „while OttJobb do Balra” kifejezés egy jobb szintaxis a terv ciklikus részére.) Az ÉS-VAGY-GRÁF-KERESÉS-ben szükséges módosításokat a 12.16. feladat dolgozza fel. A megvalósítás kulcsa, hogy az állapottérben egy  $L$  állapotba visszalépő hurok a tervben egy arra a pontra viaszumatót hurkot jelent, ahol az  $L$  állapotba vezető résztervet végrehajtjuk.

Így már képesek vagyunk feltételeket és ciklusokat tartalmazó programokhoz hasonlatos összetett tervek létrehozására. Sajnos ezek a ciklusok végtelen ciklusok lehetnek. Például a tripla-Murphy világ cselekvés reprezentációjában a semmi jelentése, hogy

a *Balra* szükségszerűen sikeres. A ciklikus tervek ezért kevésbé előnyösek, mint a ciklus nélküliek, de megoldásnak tekinthetők, amennyiben minden levél egy célállapot, és a terv minden pontjából elérhető egy levél.

## Feltételes tervkészítés részlegesen megfigyelhető környezetekben

Az előző alfejezet teljesen megfigyelhető környezetekkel foglalkozott, amelyek előnye, hogy a feltételes ellenőrzések bármit kérdezhetnek, és biztosak lehetnek a választ kapnak. A valódi világban a részleges megfigyelhetőség jóval gyakoribb. Egy részlegesen megfigyelhető tervkészítési feladat kiinduló állapotában az ágens csak bizonyos dolgokat tud az aktuális állapotról. Ennek a helyzetnek a legegyszerűbb modellezése, ha a kiinduló állapotról annyit mondunk, hogy egy **állapothalmazba** tartozik. Az állapothalmaz az ágens kiinduló **hiedelmi állapotát** (*belief state*) írja le.<sup>8</sup>

Tegyük fel, hogy a porszívóvilág ágens tudja, hogy a jobb oldali négyzeten van, és az tiszta, de nem tudja érzékelni a piszok jelenlétét vagy hiányát a többi négyzeten. Ekkor, *legjobb ismeretei szerint* két állapotban lehet: a bal oldali négyzet vagy piszkos, vagy tiszta. Ezt a hiedelmi állapotot a 12.12. ábrán *A*-val jelöltük. Az ábra a „váltakozó dupla-Murphy” porszívóvilág ÉS-VAGY gráfjának egy részét mutatja be, melyben a tiszta négyzetet elhagyó ágens piszkot hagyhat maga után.<sup>9</sup> Ha a világ teljesen megfigyelhető volna, az ágens egy „Mozogj balra és jobbra, és szívd fel a piszkot, amennyiben találsz, amíg minden kocka tiszta nem lesz, és a bal oldali kockán vagyok” formájú ciklikus megoldást készíthetne (lásd 12.16. feladat). Sajnos csak lokális szemétérzékeléssel ez a terv végrehajthatlan, hiszen a „mindkét kocka tiszta” teszt igazságértéke nem határozható meg.

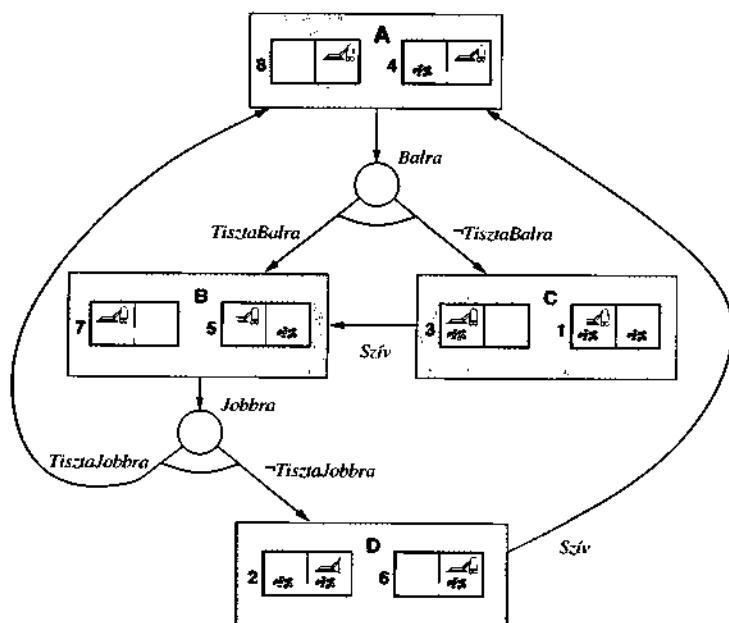
Figyeljük meg az ÉS-VAGY gráf felépítését. Az *A* hiedelmi állapotból a *Balra* mozgás kimenetelét mutatjuk. (A többi cselekvésnek nincs értelme.) Mivel az ágens piszkot hagyhat maga után, a két kiinduló világban négy lehetséges világ adódhat, ahogy az a *B* és *C* állapotokban látható. A rendelkezésre álló érzékelő információk alapján a világ két különálló hiedelmi állapotra osztható.<sup>10</sup> A *B*-ben, az ágens tudása a *TisztaBal*, míg *C*-ben a *¬TisztaBal*. A piszok feltakarítása *C*-ben az ágenst a *B*-be mozgatja. A *B*-ből a jobbra mozgás vagy hagy piszkot maga után, vagy nem, így az ágensnek azon tudása alapján, hogy *TisztaJobb* igaz (vissza az *A*-ra) vagy hamis (*D* hiedelmi állapot), ismét négy lehetséges világ adódik.

Összegezve a nemdeterminisztikus, részben megfigyelhető környezetek a hiedelmi állapotok egy ÉS-VAGY gráfját eredményezik. Ebből adódóan feltételes tervezek találhatók pontosan ugyanazon algoritmusokkal, mint a teljesen megfigyelhető esetben, nevezetesen az ÉS-VAGY-GRÁF-KERESÉS-sel. Ez abból is könnyen megérhető, ha belátjuk, hogy az ágens *hiedelmi* állapota *mindig teljesen megfigyelhető*, azaz *mindig* tudja, hogy mit tud. A „hagyományos” teljesen megfigyelhető problémamegoldás csak egy speciális eset,

<sup>8</sup> Ezeket a fogalmakat a 3.6. alfejezetben vezettük be, így az olvasó ezt átismerheti a továbblépés előtt.

<sup>9</sup> A kisgyermekes szülők jól ismerhetik ezt a jelenséget. Tisztelet a kivételeknek.

<sup>10</sup> Vegyük észre, hogy ezek nem aszerint kerülnek osztályzásra, hogy az ágens piszkot hagy-e maga után, amikor mozdul. A hiedelmi állappotterbeli elágazásokat a különböző tudáslehetőségek, és nem a különböző fizikai kimenetek okozzák.



**12.12. ábra.** A „váltakozó dupla-Murphy” porszívóvilág ÉS-VAGY gráfjának egy részét mutatja be, melyben a tiszta négyzetet elhagyó ágens piszkot hagyhat maga után. Az ágens nem tudja érzékelni a más négyzetben levő piszkot.

melyben minden hiedelmi állapot egy egyelemű halmaz, pontosan egy fizikai állapottal.

Ezzel készen vagyunk? Nem egészen! Még meg kell határoznunk, hogy hogyan reprezentáljuk a hiedelmi állapotokat, hogyan működik az érzékelés, és hogy ebben az új helyzetben hogyan írjuk le a cselekvéseket.

A hiedelmi állapotok esetén alapvetően három választásunk van:

1. Teljes állapotleírások halmazai. Például a 12.12. ábra kiinduló hiedelmi állapota az

$$\{(OttJobb \wedge TisztaJobb \wedge TisztaBal), (OttJobb \wedge TisztaJobb \wedge \neg TisztaBal)\}$$

Ezzel a leírással könnyű dolgozni, de nagyon költséges: ha  $n$  kétértékű ítéletállítás definiál egy állapotot, akkor a hiedelmi állapot  $O(2^n)$  fizikai állapotleírást tartalmazhat, melyek mindegyike  $O(n)$  méretű. Ha az ágens az ítéletállításoknak csak egy töredékét ismeri, exponenciálisan nagy hiedelmi állapotok adódnak – minél kevesebbet tud, annál több lehetséges állapotban lehet.

2. Logikai mondatok, melyek pontosan leírják a hiedelmi állapotban lehetséges világok halmazát. Például a kiinduló állapot az

$$OttJobb \wedge TisztaJobb$$

formában adható meg. Nyilvánvaló, hogy bármely hiedelmi állapot pontosan befoglalható egyetlen logikai mondatba. Ha akarjuk, az összes konjunktív állapotleírás diszjunkcióját vehetjük, de a példánk mutatja, hogy ennél tömörebb mondatok létezhetnek.

Az általános logikai mondatok egyik hátulütője, hogy mivel sok ekvivalens, de különböző logikai mondat írhatja le ugyanazt a hiedelmi állapotot, az ismétlődő állapotok ellenőrzése általános tételbizonyító képességeket vár el a gráfszerkesztő algoritmustól. Ezért a mondatok egy *kanonikus* reprezentációját szeretnénk, amelyben minden hiedelmi állapot pontosan ugyanannak a mondatnak felel meg.<sup>11</sup> Egy ilyen reprezentáció, azaz ítéletállítások nevei alapján rendezett literálok konjukcióját használja, melynek egy példája a  $\neg OttJobb \wedge TisztaJobb$ . Ez a 11. fejezet nyílt világ feltételezésében (*open-world assumption*) egy egyszerű állapotleírás. Nem minden logikai mondat írható fel ilyen alakban (például nincs mód az *OttBal*  $\vee$  *TisztaJobb* felírására), de számos probléma kezelhető.

3. **Tudás ítéletállítások (knowledge propositions)**, amelyek az ágens ismereteit írják le. (Ugyanezt lásd a 7.7. alfejezetben.) A kiinduló állapotunk:

$$K(OttJobb) \wedge K(TisztaJobb)$$

ahol  $K$  jelentése „tudja” és  $K(P)$  jelentése, hogy az ágens tudja, hogy  $P$  igaz.<sup>12</sup> A tudás ítéletállításokkal zárt világ feltételezést használunk, azaz ha egy állítás nem jelenik meg a listában, akkor hamisnak feltételezzük. Például a  $\neg K(TisztaBal)$  és a  $\neg K(\neg TisztaBal)$  implicit szerepelnek a fenti mondatban, így az rögzít a tényt, hogy az ágens érzéketlen a *TisztaBal* igazságéértékére.

Kimutatható, hogy a második és harmadik lehetőségek durván azonosak, de mi a harmadik tudás, az ítéletállítás lehetőséget használjuk, mert ez az érzékelés egy erősebb leírását adja, és mert már tudjuk, hogy hogyan írhatunk a zárt világ feltételezés (*closed-world assumption*) mellett STRIPS kifejezéseket.

Mindkét esetben, minden ítéletszimbólum háromféleképpen jelenhet meg: lehet pozitív, negált vagy ismeretlen. Ezért így pontosan  $3^n$  lehetséges hiedelmi állapot adható meg. A hiedelmi állapotok halmaza így a hatványhalmaza (az összes részhalmaz halmaza) a fizikai állapotoknak. Összesen  $2^n$  fizikai állapot van, ezért  $2^n$  hiedelmi állapot, ami sokkal több, mint  $3^n$ , így a 2. és 3. választás eléggé korlátozottan alkalmas a hiedelmi állapotok leírására. Ez jelenleg használhatatlannak tűnik, hiszen bármely séma, ami képes minden lehetséges hiedelmi állapot reprezentálására,  $O(\log_2(2^n)) = O(2^n)$  bitet igényel, hogy legrosszabb esetben mindegyiket leírhatssa. A mi egyszerű sémánk csak  $O(n)$  bitet igényel a hiedelmi állapotok leírásához, mert a kifejezőképességet a tömörségre cserélteük. Nevezetesen, ha egy cselekvés megjelenik, amelynek az előfeltételei ismeretlenek, akkor az eredményként kapott hiedelmi állapot nem lesz pontosan reprezentálhazó, és a cselekvés kimenetele ismeretlen lesz.

Most arról kell döntenünk, hogy az érzékelés hogyan működik. Itt két választásunk van. Használhatunk automatikus érzékelést (*automatic sensing*), ami annyit tesz, hogy az ágens minden időlépésben az összes elérhető érzetet megkapja. A 12.12. ábrán látható példa a helyzet és a helyi tisztaság meghatározására automatikus érzékelést fel-

<sup>11</sup> Az általános ítéletlogikai mondatok legjobb kanonikus reprezentációja a bináris döntési diagram (binary decision diagram) vagy BDD (Bryant, 1992).

<sup>12</sup> Ez a jelölés ugyanaz, mint amit a 7. fejezetben az áramkörökkel használtunk. Néhány szerző ezt a „tudja, hogy  $P$  igaz-e” értelmeiben használja. A két értelmezés közötti fordítás kézenfekvő.

tételez. Használhatunk ellenben aktív érzékelést (active sensing), ami annyit jelent, hogy az érzékelő információk csak megadott érzékelési cselekvések (sensory actions), (mint *TisztaságEllenőrzés* és *PozícióEllenőrzés*) végrehajtásával nyerhetők. Az érzékelési típusokat sorban tárgyalni fogjuk.

Most használunk tudás állításokat a cselekvések leírásához. Tegyük fel, hogy az automatikus helyi tisztaságérzékeléssel felruházott váltakozó-dupla-Murphy világ problémában az ágens *Balra* lép. Az erre a világra vonatkozó szabályoknak megfelelően az ágens hagy, vagy talán nem hagy piszkot maga után, ha a négyzet tiszta volt. Mint fizikai következmény, ez *diszjunktív* lenne, de mint *tudás* következmény ez egyszerűen törli az ágens *TisztaJobb* tudását. Emellett az ágens a helyi piszkérzékelés miatt így vagy úgy tudni fogja, hogy vajon a *TisztaBal* igaz-e, és tudni fogja, hogy a pozíciója *OttBal*:

*Cselekvés(Balra, Előfeltétel: OttJobb,*  
*KÖVETKEZMÉNY: K(OttBal)  $\wedge$   $\neg K(OttJobb) \wedge$*   
*when TisztaJobb:  $\neg K(TisztaJobb) \wedge$*   
*when TisztaBal: K(TisztaBal)  $\wedge$*   
*when  $\neg$ TisztaBal: K( $\neg$ TisztaBal)* (12.2)

Vegyük észre, hogy az előfeltételek és a *when* feltételek egyszerű állítások és nem tudás állítások. Ez úgy van, ahogy lennie kell, hiszen a cselekvések kimenetele függ az aktuális világtól, de hogyan ellenőrizhetnénk ezen feltételek igazságértékét csupán a hiedelmi állapot alapján? Ha az ágens az aktuális hiedelmi állapotban *tudja* mondjuk a *K(OttJobb)* állítást, akkor az állításnak igaznak kell lennie az aktuális fizikai állapotban, így egyben a cselekvés is alkalmazható. Ha az ágens nem ismeri az állítást (például *TisztaBal* az *if* feltételel), akkor a hiedelmi állapotnak tartalmaznia kell olyan világokat, melyekben a *TisztaBal* igaz, és olyan világokat, amelyekben a *TisztaBal* hamis. Pontosan ez az, ami miatt egy cselekvés többszörös hiedelmi állapotokat eredményez. Így, ha a kiinduló állapot a (*K(OttJobb)  $\wedge$  K(TisztaJobb)*), akkor a *Balra* lépés után a (*K(OttBal)  $\wedge$  K(TisztaBal)*) és a (*K(OttBal)  $\wedge$  K( $\neg$ TisztaBal)*) a két lehetséges hiedelmi állapot. Mindkét esetben ismert a *TisztaBal* értéke, így a *TisztaBal* teszt felhasználható a tervben.

Aktív érzékelés (mint az automatikus érzékelés ellentettje) esetén, az ágens csak kérésre kap új megfigyeléseket. Így a *Balra* lépés után az ágens nem tudja, hogy a bal oldali négyzet piszkos-e, ezért a (12.2) cselekvésleíró egyenletben az utolsó két feltétes következmény már nem jelenik meg. Az ágens a *TisztaságEllenőrzés* cselekvéssel derítheti ki, hogy a négyzet piszkos-e:

*Cselekvés(TisztaságEllenőrzés, KÖVETKEZMÉNY: when OttBal  $\wedge$  TisztaBal: K(TisztaBal)  $\wedge$*   
*when OttBal  $\wedge$   $\neg$ TisztaBal: K( $\neg$ TisztaBal)  $\wedge$*   
*when OttBal  $\wedge$  TisztaJobb: K(TisztaJobb)  $\wedge$*   
*when OttBal  $\wedge$   $\neg$ TisztaJobb: K( $\neg$ TisztaJobb)* (12.3)

Könnyű megmutatni, hogy aktív érzékelés esetén a *Balra* lépéssel követett *TisztaságEllenőrzés* cselekvés ugyanazt a két hiedelmi állapotot eredményezi, amit az automatikus érzékelés esetén a *Balra* adott. Aktív érzékelés esetén a fizikai cselekvések egy hiedelmi állapotot minden egyetlen követő hiedelmi állapotra képeznek le. A többszörös hiedelmi állapotok csak az érzékelő cselekvések által jöhetnek létre, melyek

specifikus tudást adnak, s ezért lehetővé teszik a feltételvizsgálatok használatát a tervekben.

Az eddigiekben az állapottér ÉS-VAGY keresésén alapuló feltételes tervkészítésre mutattunk be egy általános megközelítést. Ez a megközelítés néhány tesztproblémán elég hatékonynak bizonyult, de más feladatokra alkalmatlan. Bizonyítható, hogy a feltételes tervkészítés nagyobb algoritmikus komplexitású, mint a hagyományos. Emlékezzünk vissza, hogy az *NP* osztály definíciója alapján egy megoldásról ellenőrizhető, hogy polinomiális idejű-e. Ez a hagyományos tervekre (legalábbis a polinomiális méretűekre) igaz, így a hagyományos tervkészítés az *NP* osztályba tartozik. A feltételes tervkészítés esetén egy jelöltre ellenőrizni kell, hogy az összes lehetséges állapotra a tervben létezik-e *valamelyen*, a célt kielégítő útvonal. Az „összes/valamelyen” összeállítás nem ellenőrizhető polinomiális időben, így a feltételes tervkészítés nehezebb, mint *NP*. Ez csak úgy kerülhető el, hogy a tervkészítési fázisban figyelmen kívül hagyunk néhány lehetséges eshetőséget, és ezeket csak akkor kezeljük, ha valójában fellépnek. A következő alfejezetben ezt a megközelítést elemezzük.

## 12.5. VÉGREHAJTÁS MONITORIZÁSA ÉS ÚJRATERVEZÉSE

A végrehajtás-monitorozó (execution monitoring) ágens érzékelőivel ellenőrzi, hogy minden a terv szerint megy-e. Murphy törvénye alapján az egereknek, embereknek és a feltételes tervkészítő ágenseknek még a legjobban elkészített tervei is gyakran sikertelenek. A probléma a nem korlátos nemdeterministikusság, azaz néhány váratlan körülmeny minden felmerül, melyre az ágens cselekvés leírásai helytelenek. A valós környezetekben ezért a végrehajtás-monitorozás elengedhetetlen. Kétféle végrehajtás-monitorozást veszünk számba: az egyszerű, de gyenge cselekvésmonitorozást (action monitoring), ahol az ágens a környezet vizsgálatával ellenőrzi, hogy a következő cselekvés működni fog, és az összetettebb, de hatékonyabb tervmonitorozást (plan monitoring), melyben az ágens a terv teljes hátralevő részét ellenőrzi.

Az újratervező (replanning) ágens tudja, hogy váratlan események esetén mit kell tennie: újra meghívja a tervkészítőt, hogy a cél eléréséhez egy új tervet biztosítson. Annak elkerülésére, hogy túl sok időt töltünk tervkészítéssel, ez rendszerint a régi terv javításával történik, azaz utat keresünk a fennálló nem várt állapotból vissza, a meglévő tervhez.

Példaként térjünk vissza a 12.9. ábra dupla-Murphy porszívóvilágához. Ebben a világban egy tiszta cellára lépés néha bepiszkolja azt. De mi történik, ha az ágens ezt nem tudja vagy nem foglalkozik vele? Akkor egy nagyon egyszerű megoldással áll elő: [Balra]. Ha a terv végrehajtásánál érkezéskor nem történik piszkítás, akkor az ágens a cél elérését detektálja. Ellenkező esetben azonban, mivel az implicit Befejez lépés TisztaBal előfeltétele nem teljesül, az ágens új tervet készít: [Szív]. Ennek a tervnek a végrehajtása minden sikeres.

A végrehajtás-monitorozás és újratervezés együttesen egy általános stratégia, ami mind a teljesen, mind pedig a részben megfigyelhető környezetekre alkalmazható a tervkészítési reprezentációk széles körén, beleértve az állapottér-, a részben rendezett és a feltételes terveket. A 12.13. ábra az állapottér-tervkészítésre mutat be egy-

szerű megközelítést. A tervkészítő ágens egy céllal indít, és kiinduló tervet készít ennek eléréséhez. Ellentétben más tervkészítő ágensekkel, az újratervező ágens követi minden a hátralevő, még végrehajtatlan *terv* részletet, valamint a teljes eredeti tervet *teljes\_tern*-et. **Cselekvésmonitorozást (action monitoring)** használ: a *terv* következő cselekvésének végrehajtása előtt az ágens megvizsgálja az érzékelőit, hogy megbizonyosodjon arról, hogy a terv előfeltételei nem váltak-e váratlanul kielégítetlenné. Ha igen, akkor az ágens egy cselekvéssorozat újratervezésével megpróbál a *teljes\_tern* egy pontjára visszajutni.

```

function ÚJRATERVEZŐ-ÁGENS(érzékelés) returns egy cselekvés
  static: TB, egy tudásbázis (cselekvésleírásokat tartalmaz)
           terv, egy terv, kezdetben[]
           teljes_tern, egy terv, kezdetben[]
           cél, egy cél

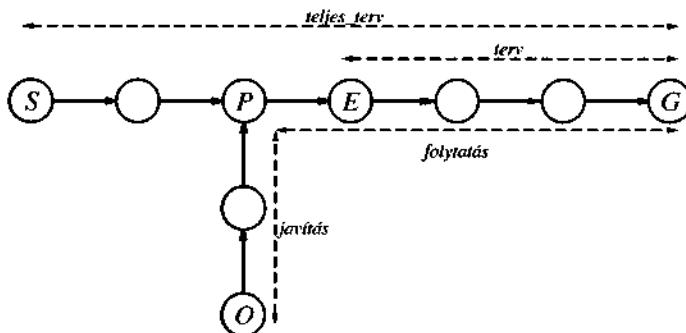
  ÉRTESEN(TB, ÉRZÉKELŐ-MONDAT-KÉSZÍTÉS(érzékelés, t))
  aktuális  $\leftarrow$  LEÍRÁS-KÉSZÍTÉS(TB, t)
  if terv = [] then
    teljes_tern  $\leftarrow$  terv  $\leftarrow$  TERVKÉSZÍTŐ(aktuális, cél, TB)
  if ELŐFELTÉTELEK(ELŐSZÖR(terv)) aktuálisan nem igaz a TB-ben then
    jelöltek  $\leftarrow$  RENDEZÉS(teljes_tern, az aktuális-tól vett távolság szerint rendezve)
    keres állapot s a jelöltek amelyekre
      kudarc  $\neq$  javítás  $\leftarrow$  TERVKÉSZÍTŐ(aktuális, s, TB)
      folytatás  $\leftarrow$  a teljes_tern s-ből induló farokrésze
      teljes_tern  $\leftarrow$  terv  $\leftarrow$  HOZZÁFŰZ (javítás, folytatás)
  return RRT(terv)

```

**12.13. ábra.** Egy cselekvésmonitorozó és -újratervező ágens. Ez szubrutrinként a TERVKÉSZÍTŐ teljes állapotot tervkészítő algoritmust használja. Ha a következő cselekvés előfeltételei nem teljesülnek, az ágens a *teljes\_tern* lehetséges *p* pontjain iterál olyat keresve, amelyhez a TERVKÉSZÍTŐ útvonalat tud tervezni. Ezt az útvonalat hívjuk *javításnak*. Ha a TERVKÉSZÍTŐ sikeres a javításban, az új terv készítéséhez összefűzi a javítást és a terv *p* utáni részét. Az ágens ezután a terv első lépését adja vissza.

A 12.14. ábra a folyamat sematikus illusztrációja. Az újratervező észreveszi, hogy a *terv* első cselekvésének előfeltételeit az aktuális állapot nem elégíti ki. Ezután meghívja a tervkészítőt, hogy készítsen egy új *javítás*-nak nevezett résztervet, amely az aktuális állapotból a *teljes\_tern* valamely *s* állapotába vezet vissza. Ebben a példában az *s* történetesen egylépésnyi visszalépést jelent az aktuális hátralevő *terv*-ből. (Ez az oka, hogy a hátralevő terv helyett a teljes tervet figyeljük.) Általánosságban *s*-et a jelen állapothoz a lehető legközelebbre választjuk. Az új terv a *javítás* és a *teljes\_tern* *s*-től hátralevő része – amit *folytatás*-nak nevezünk – összefűzéséből adódik, mellyel az ágens készen áll a végrehajtás folytatására.

Térjünk vissza az asztal és a szék azonos színűre festésének problémájához, ez alkalmmal újratervezéssel. Feltételezzük a teljesen megfigyelhető környezetet. A kiinduló állapotban a szék kék, az asztal zöld, valamint 1 doboz kék és 1 doboz zöld festék áll rendelkezésünkre. Ez a következő probléma definícióhoz vezet:



**12.14. ábra.** A végrehajtás előtt a tervkészítő egy *teljes\_terv*-nek nevezett tervet készít, mely *S*-ből *G*-be vezet. Az ágens az *E*-vel jelölt pontig végrehajtja a tervet. A *terv* hátralevő részének végrehajtása előtt a szokásos módon ellenőrzi az előfeltételeket, és azt találja, hogy valójában az *O* állapotban van, és nem az *E*-ben. Ezután meghívja a tervkészítő algoritmust, hogy készítsen egy *javítás-t*, mely egy terv az *O*-ból az eredeti *teljes\_terv* egy *P* pontjához. Az új *terv* ezután a *javítás* és a *folytatás* (az eredeti *teljes\_terv* maradék része) összefűzéséből adódik.

*Kiindulás(Színe(Szék, Kék)  $\wedge$  Színe(Asztal, Zöld)*

$\wedge$  SzíntTartalmaz(BC, Kék)  $\wedge$  FestékDoboz (BC)  
 $\wedge$  SzíntTartalmaz(RC, Piros)  $\wedge$  FestékDoboz (RC)

*Cél(Színe(Szék, x)  $\wedge$  Színe(Asztal, x))*

*Cselekvés(Fest(tárgy, szín),*

*ELŐFELTÉTEL:VanFesték(szín)*

*KÖVETKEZMÉNY:Színe(tárgy, szín))*

*Cselekvés(Nyit(doboz),*

*ELŐFELTÉTEL:FestékDoboz(doboz)  $\wedge$  Színe(doboz, szín)*

*KÖVETKEZMÉNY:Színe(szín))*

Az ágens TERVKÉSZÍTŐ-jének a következő tervvel kellene előállnia:

*[Indít; Nyit(BC); Fest(Asztal, Kék); Befejez]*

Az ágens most készen áll a terv végrehajtására. Tegyük fel, hogy minden jól megy, és az ágens kinyitja a kék festéket és az asztalra keni. Az előző alfejezet ágensei a terv lépéseinak befejezése után győzelmet kiáltanának. A végrehajtás-monitorozó ágensnek ellenben először ellenőriznie kell a *Befejez* lépés előfeltételét, ami azt mondja, hogy a két bútornak azonos színűnek kell lennie. Tegyük fel, hogy az ágens azt érzékeli, hogy a bútorok nem azonos színűek, mivel a festés hiányossága miatt az asztalon egy zöld folt maradt. Az ágensnek ezután keresnie kell egy célpontot a *teljes\_terv*-ben, amelyet megcélozhat, és egy javító cselekvéssorozatot kell kidolgozni, hogy ide jusson. Az ágens észreveszi, hogy az aktuális állapot azonos a *Fest* cselekvés előfeltételeivel, így a *javítás* egy üres sorozat, és a *terv* azonos az épp végrehajtott [*Fest, Befejez*] sorozattal. Ezzel az új tervvel a monitorozás végrehajtása folytatódik, és a *Fest* cselekvés újra végrehajtásra kerül. Ez a viselkedés addig ismétlődik, amíg az asztalt teljesen lefestettnek nem érzékeljük. Vegyük észre, hogy a ciklus a terv-végrehajtás-újratervezés folyamat során alakult ki, nem pedig egy explicit ciklus az eredeti tervben.

A cselekvésmonitorozás a végrehajtás-monitorozás nagyon egyszerű módszere, de néha kevésbé intelligens viselkedéshez vezethet. Például tegyük fel, hogy az ágens olyan tervet készít, amelyben az asztalt és a széket is pirosra festi. Ekkor kinyitja a piros festéket, és azt találja, hogy nincs elegendő festék a székhez. A cselekvésmonitorozás nem érzékelné ezt a sikertlenséget, csak *miután* a széket befestettük, amikor is a *VanFesték(Piros)* hamissá válik. Amire valójában szükségünk van, az az, hogy akkor érzékeljük a hibát, amikor olyan állapotba jutunk, ahonnan a terv hátralevő része már nem működik. A **tervmonitorozás (plan monitoring)** ezt úgy éri el, hogy a teljes hátralevő tervben szereplő előfeltételek sikerességét ellenőrzi, azaz a terv minden lépéseinél előfeltételeit, kivéve azokat, amelyeket a hátralevő terv egy másik lépéssel ér el. A tervmonitorozás a hibás terv végrehajtását a lehető leghamarabb megszakítja, azaz nem várja meg, míg a hiba valójában fellép.<sup>13</sup> Néhány esetben ez az ágenst a teljes bukástól mentheti meg, amikor a hibás terv egy olyan zsákutcába vezetne, melyből a cél már elérhetetlen.

Elég kézenfekvő a tervkészítő algoritmus módosítása, hogy az a terv minden pontján tartalmazza a hátralevő terv sikeréhez szükséges előfeltételeket. Ha a tervmonitorozást kiterjesztjük, hogy az aktuális pont helyett az összes jövőbeni pontra ellenőrizze, hogy az aktuális állapot kielégíti-e a terv előfeltételeit, akkor a tervmonitorozás kihasználhatja a **szerenesés rábukkanását (serendipity)**, azaz a véletlenszerű sikert. Ha valaki arra jártában pirosra festi az asztalt ugyanabban az időben, amikor az ágens a széket festi pirosra, akkor a végső terv előfeltételei teljesülnek (a cél teljesül), és az ágens korábban megpihenhet.

Eddig a monitorozást és újratervezést a teljesen megfigyelhető környezetek esetére mutattuk be. A részlegesen megfigyelhető környezetek esetén jóval komplikáltabb problémák léphetnek fel. Először is olyan hibák léphetnek fel, amelyet az ágens nem tud érzékelni. Másodsor „az előfeltételek ellenőrzése” érzékelő cselekvések végrehajtását teheti szükségesse, amelyet tervezni kell vagy a tervkészítési időben, ami a feltételes tervkészítéshez vezet vissza, vagy a végrehajtási időben. A legrosszabb esetben az érzékelési cselekvések végrehajtása összetett tervet igényel, melyhez monitorozás, így további érzékelési cselekvések szükségesek. És ez így megy tovább. Ha az ágens ragaszkodik minden előfeltétel ellenőrzéséhez, akkor elkövethető, hogy soha sem jut hozzá, hogy valójában *tegyen* is valamit. Az ágensnek csak a fontos változók ellenőrzését kellene szem előtt tartani, melyek jó esélytel okoznak hibát, és nem túl drága a megfigyeléstük. Ez lehetőséget ad az ágensnek, hogy megfelelőképpen reagáljon a fontos fenegetésekre, de ne veszegessen időt annak ellenőrizgetésére, hogy összeomlik-e az ég.

Miután bemutattuk a monitorozás és újratervezés módszerét, meg kell kérdeznünk: „Működik?” Ez meglepően becsapós kérdés. Ha a kérdést úgy értelmezzük, hogy „Garantálni tudjuk-e, hogy az ágens minden, még nem korlátos nemdeterminisztikusság esetén is célt ér?”, akkor a válasz nem, hiszen az ágens figyelmetlenségből zsákutcába juthat, csakúgy, mint a 4.5. alfejezetben bemutatott online keresés. Például lehet, hogy a porszívóágens nem tudja, hogy az elemei kifogyhatnak. Zárjuk ki a

<sup>13</sup> A tervmonitorozás okosabbá teszi az ágensünket egy ganajtúró bogárnál (lásd 71. oldal). Az ágensünk észrevenné, hogy a ganaj már nincs a lábainál, újratervezne, hogy szerezzen egy másikat, és azzal zárná le a lyukat.

zsákutcakat, azaz tételezzük fel, hogy az ágens olyan tervet tud létrehozni, amelyben a cél a környezet *bármely* állapotából eléri. Ha feltételezzük, hogy a környezet valóban nemdeterminisztikus, abban az értelemben, hogy egy ilyen tervnek egy adott végrehajtása esetén mindenig van *valamelykora* esélye a sikerre, akkor az ágens szükségszerűen célt ér. Az újratervező ágens képességei ezért megfelelnek a feltételes tervkészítő ágens képességeinek. Valójában módosíthatjuk a feltételes tervkészítőt, hogy csak részleges megoldást készítsen, amely „*if <teszt> then terv\_A else újratervez*” formájú lépésekkel tartalmaz. Az említett korlátozásokkal egy ilyen terv megfelelő megoldása lehet az eredeti problémának, és elkészítése szintén sokkal olcsóbb lehet, mint a teljes feltételes tervnek.

Hiba akkor lép fel, ha az ágens ismételt kísérletei is sikertelenek a cél elérésében, azaz valamilyen, számára ismeretlen előfeltétel vagy következmény blokkolja. Például ha az ágens rossz kulcskártyát kap a hotelszobájához, akkor nincs az a beillesztési és eltávolítási próbálkozás, ami kinyitná az ajtót.<sup>14</sup> Egy megoldás, hogy ugyanazon terv ismételgetése helyett véletlenszerűen választunk egy lehetséges javítóterv halmazból. Ebben az esetben egy új kulcskártya kérése a recepcióról hasznos alternatív javító terv. Mivel nem biztos, hogy az ágens képes a valóban nemdeterminisztikus esetet és a hiába-valóság esetét megkülönböztetni, egy kis változatosság a javításban általánosságban is jó ötlet.

A helytelen cselekvés problémájára egy másik megoldás a **tanulás**. Néhány próbálkozás után a tanuló ágensnek képesnek kell lennie a cselekvésleírás módosítására, hogy az igényelje, hogy a kulcs nyissa az ajtót. Ezen a ponton az újratervező automatikusan egy új tervvel áll elő, mint egy új kulcs megszerzése. A 21. fejezet ezt a fajta tanulást mutatja be.

Az újratervező ágensnek még ezekkel a lehetséges javításokkal is vannak további hátrányai. Nem hatékony a valós idejű környezetekben, és nincs korlát az újratervezési időre, ezáltal nincs korlát a cselekvésről való döntés idejére sem. Szintén képtelen saját célokat kialakítani vagy a meglévő célokhoz új célokat fogadni, azaz nem lehet hosszú életű egy összetett környezetben. Ezeket a hátrányokat kezeljük a következő fejezetben.

## 12.6. FOLYTONOS TERVKÉSZÍTÉS

Ebben az alfejezetben olyan ágenst tervezünk, amely meghatározatlanul sokáig kitart egy környezetben. Így ez nem egy „problémamegoldó”, ami egyetlen cél szeretne elérni és ezt tervezí, és addig cselekszik, amíg a cél el nem éri, hanem folyamatosan változó célok, tervkészítési és cselekvési fázisok sorozatát élí meg. Ahelyett hogy a tervkészítő és a cselekvésmonitorozó mint különálló, de egymásnak eredményeket szolgáltató folyamatokat tekintenénk, a **folytonos tervkészítő ágensben** (*continuous planning agent*) ezeket egyetlen folyamatként képzeli el.

Az ágensre minden úgy gondolunk, mint ami része egy terv végrehajtásának, az élete nagy tervének. Cselekvései tartalmazzák a végrehajtásra készen álló terv lépéseinek végrehajtását, a nyitott előfeltételek kielégítésére vagy a konfliktusok feloldására szolgáló tervfinomítást, a végrehajtás során nyert új információk fényében a terv módosítását.

<sup>14</sup> A tervjavítás hatástanon ismétlése pontosan megfelel a szöcskeölő darázs viselkedésének (lásd 72. oldal).

Nyilvánvaló, hogy amikor először új célt formál, az ágensnek nincsenek végrehajtásra kész cselekvései, ezért egy kis időt a részleges terv elkészítésével tölt. Nagyon is lehetséges azonban, hogy az ágens a terv teljes befejezése előtt elkezdi a végrehajtást, különösen ha független részcélokat kell elérni. A folytonos tervkészítő ágens folyamatosan monitorozza a világot, az új megfigyelések alapján módosítja ennek modelljét, még akkor is, ha elképzélései még mindig változnak.

Először egy példán haladunk végig, majd bemutatjuk az ágens programját, amelyet FOLYTONOS-RRT-ÁGENS-nek nevezünk, mivel a tervezett cselekvések reprezentálására részben rendezett terveket használ. Hogy a leírást egyszerűsítsük, teljesen megfigyelhető környezetet feltételezünk. Ugyanezen technikák kiterjeszthetők a részben megfigyelhető esetre is.

A példánk a kockavilág problémakörből származik (lásd 11.1. alfejezet). A kiinduló állapot a 12.15. (a) ábrán látható. A  $Mozgat(x, y)$  cselekvésre van szükségünk, ami az  $x$  kockát az  $y$  kockára mozgatja, amennyiben mindenben szabadok. A cselekvés sémája:

*Cselekvés*( $Mozgat(x, y)$ ,

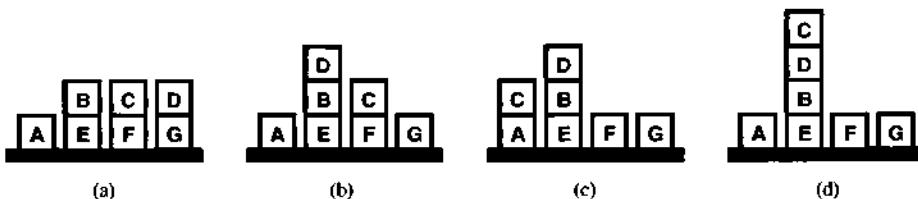
*ELŐFELTÉTEL*:  $Tiszta(x) \wedge Tiszta(y) \wedge Rajta(x, z)$

*KÖVETKEZMÉNY*:  $Rajta(x, y) \wedge Tiszta(z) \wedge \neg Rajta(x, z) \wedge \neg Tiszta(y)$

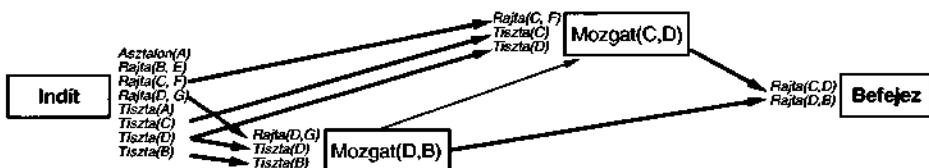
Az ágensnek először egy célt kell megfogalmaznia. Itt nem tárgyaljuk a célkialakítást, helyette feltételezzük, hogy az ágensnek valahogy megmondta (vagy saját maga elődöntötte), hogy a  $Rajta(C, D) \wedge Rajta(D, B)$  cél kell elérnie. Az ágens ehhez a célhöz kezdi a tervezést. Ellentétben minden másik ágensünkkel, amelyek lekapcsolják érzékelőket, amíg a tervkészítő egy teljes megoldást nem ad a problémára, a folytonos tervkészítő ágens inkrementálisan épít egy tervet, ahol a terv minden bővítése egy korlátos időtartamot vehet igénybe. minden bővítés után az ágens egy *NoOp* cselekvést ad vissza, és ellenőrzi az érzékeléseit. Feltételezzük, hogy az érzékelő adatok nem változnak, és az ágens gyorsan megalkotja a 12.16. ábrán látható tervet. Vegyük észre, hogy bár minden cselekvés előfeltételeit teljesíti az *Indít* állapot, van egy követési megkötés, ami a  $Mozgat(D, B)$  cselekvést a  $Mozgat(C, D)$  elő helyezi. Erre azért van szükség, hogy a  $Tiszta(D)$  igaz maradjon, amíg a  $Mozgat(D, B)$  befejeződik. A folytonos tervkészítési folyamat alatt az *Indít* minden bővítése az aktuális állapot címére. Az ágens minden cselekvés után módosítja az állapotot.

A terv készen áll a végrehajtásra, de még mielőtt az ágens cselekedhetne, a környezet közbeszól. Egy külső ágens (talán az ágens tanítója vált türelmetlenné) a (d) kockát a (b)-re helyezi, a világ most a 12.15. ábra (b) állapotának felel meg. Az ágens ezt érzékeli, észreveszi, hogy a  $Tiszta(B)$  és a  $Rajta(D, G)$  már nem igaz az aktuális állapotban, és ennek megfelelően módosítja az aktuális állapot modelljét. Az okozati kapcsolatok, amelyek a  $Mozgat(D, B)$  cselekvés  $Tiszta(B)$  és  $Rajta(D, G)$  előfeltételeit adták, már nem helyesek, el kell távolítani őket a tervből. Az új tervet a 12.17. ábra mutatja. minden pillanatban az *Indít* az aktuális állapotot reprezentálja, így ez az *Indít* különbözik az előző ábra hasonló állapotától. Vegyük észre, hogy a terv most nem teljes: a  $Mozgat(D, B)$  előfeltételei közül kettő nyitott, és a  $Rajta(D, y)$  most nem példányosított, mert már nincs okunk feltételezni, hogy a lépés a  $G$ -ről indul.

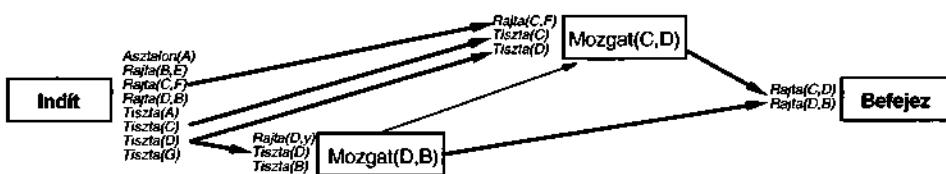
Az ágens most kihasználhatja a „segítő” közbeavatkozást, észrevéve, hogy a  $Mozgat(D, B) \xrightarrow{Rajta(D, B)} Befejez$  okozati kapcsolat helyettesíthető az *Indít*-ból a *Befejez* állapotba mutató direkt kapcsolattal. Ezt a folyamatot az okozati kapcsolat



**12.15. ábra.** Az állapotok sora, amint a folytonos tervkészítő ágens megpróbálja a (d)-nek megfelelő  $Rajta(C, D) \wedge Rajta(D, B)$  célállapotot elérni. A kiinduló állapot az (a). A (b) állapotnál egy másik ágens közbe lépett és (d)-t (b)-re helyezte. A (c) állapotban az ágens a  $Mozgat(C, D)$  cselekvést hajtotta végre, ami sikertelen volt, (c)-t az (a)-ra ejtette. Újrapróbálja a  $Mozgat(C, D)$  cselekvést, amivel eléri a (d) célállapotot.



**12.16. ábra.** A folytonos tervkészítő ágens által készített kiinduló terv. A terv egyelőre nem különbözik egy átlagos részben rendezett tervkészítő által készítettől.



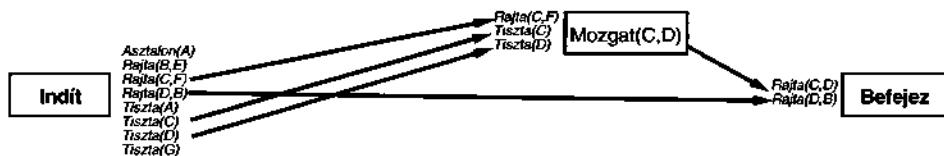
**12.17. ábra.** Miután valaki más elmozdította a D kockát a B-re, a Tisztta(B) és a Rajta(D, G) biztosítására szolgáló kapcsolatokat eldobjuk, ami ezt a tervet eredményezi

**kiterjesztésének (extension)** nevezük, és akkor alkalmazzuk, amikor egy feltétel egy későbbi helyett egy korábbi lépéssel biztosítható, új ütközés okozása nélkül.

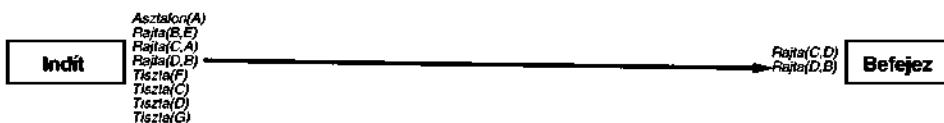
Miután a régi  $Mozgat(D, B)$  és  $Befejez$  közötti okozati kapcsolatot eltávolítottuk, a  $Mozgat(D, B)$  továbbiakban nem forrása egyetlen okozati kapcsolatnak sem. Ez most egy **redundáns lépés (redundant step)**. minden redundáns lépést és bármely ezt biztosító kapcsolatot kitörünk a tervből. Ez a 12.18. ábrán látható tervet eredményezi.

Most a  $Mozgat(C, D)$  lépés készen áll a végrehajtásra, hiszen az *Indít* lépés kielégíti minden előfeltételét, nincs szükség más megelőző lépéstre, és nem ütközik a terv egyetlen másik kapcsolatával sem. A lépést eltávolítjuk a tervből, és végrehajtjuk. Sajnos az ágens tügyetlen, és a C-t az A-ra ejti a B helyett, ami a 12.15. ábra állapotát eredményezi. Az új tervállapotot a 12.19. ábra mutatja. Vegyük észre, hogy bár most nincsenek cselekvések a tervben, a *Befejez* lépésnek még mindig van egy nyitott feltétele.

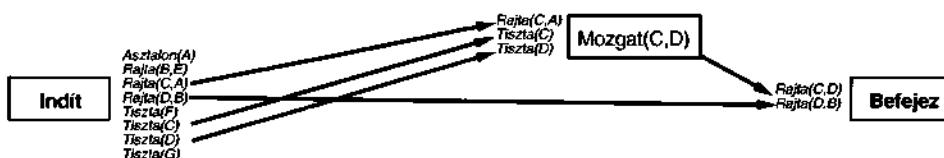
Az ágens úgy dönt, hogy a nyitott feltételhez készít tervet. Újra a  $Mozgat(C, D)$  előítéri ki a célfeltételt. Ennek előfeltételeit az *Indít* lépésből induló új okozati kapcsolatok teljesítik. Az új tervet a 12.20. ábra mutatja.



**12.18. ábra.** A *Mozgat(C, D)* által létrehozott kapcsolatot helyettesítettük az *Indít* kapcsolatával, a most már redundáns *Mozgat(D, B)* lépést pedig elhagytuk



**12.19. ábra.** A *Mozgat(C, D)* végrehajtása és a tervből való eltávolítása után az *Indít* lépés következményei azt a tényt tükrözik, hogy a C a várt D helyett az A kockára került. A *Rajta(C, D)* cél előfeltétel még minden nyitott



**12.20. ábra.** A nyitott előfeltételt a *Mozgat(C, D)* ismételt hozzáadásával oldjuk fel. Vegyük észre az előfeltételekhez szükséges új kapcsolatokat.

A *Mozgat(C, D)* újra készen áll a végrehajtásra. Ez alkalommal működik, ami a célállapotot eredményezi, melyet a 12.15. ábra (d) része mutat. Miután egy lépést elhagyunk a tervből, a *Rajta(C, D)* célfeltétel újra nyitottá válik. Mivel az *Indít* lépést frissítettük, hogy tükrözze az új állapotot, a célfeltétel azonnal kielégíthető egy kapcsolattal az *Indít* lépésből. Ez az események normális folyása, ha egy cselekvés sikeres. A végleges tervállapotot a 12.21. ábra mutatja. Mivel az *Indít* lépés az összes célfeltételt kielégíti, és nincsenek hátralevő cselekvések, az ágens szabadon eltávolíthatja a *Befejez* állapot céljait, és új célokat fogalmazhat meg.

Ebből a példából látható, hogy a folytonos tervkészítés nagyban hasonlít a részben rendezett tervkészítéshez. minden iterációban az algoritmus talál valami **tervsérülésnek (plan flaw)** nevezett javítanivalót a tervben, és megjavítja. A részben rendezett terv-



**12.21. ábra.** A *Mozgat(C, D)* végrehajtása és tervből eltávolítása után a *Rajta(C, D)* megmaradt nyitott feltételt az *Indít* lépésből induló okozati kapcsolat hozzáadásával oldjuk fel. A terv most teljes.

készítő algoritmus egy sérüléseltávolító algoritmusnak tekinthető, ahol a kétféle sérülés a nyílt előfeltétel és az okozati ütközés lehet. A folytonos tervkészítő ágens másrészről sérülések egy jóval szélesebb körét kezeli:

- **Hányzó cél:** az ágens eldöntheti, új célt vagy célokat ad a *Befejez* állapothoz. (Folytonos tervkészítésben értelmesebb lenne a *Befejez* állapot nevét *Örökkévalóság*-ra és az *Indít*-ét *Aktuális*-ra változtatni, de ragaszkodunk a hagyományokhoz.)
- **Nyílt előfeltétel:** egy okozati kapcsolatot ad a nyílt előfeltételhez, egy meglévő vagy egy új cselekvést választva (mint a részben rendezett tervkészítőben).
- **Okozati ütközések:** adott az  $A \xrightarrow{p} B$  ok-okozati kapcsolat és a  $\neg p$  következményű C cselekvés. Válasszunk az ütközés feloldásához egy rendezési megkötést vagy egy változómegkötést (mint a részben rendezett tervkészítésben).
- **Nem támogatott kapcsolat:** ha van egy  $Start \xrightarrow{p} A$  ok-okozati kapcsolat, ahol a  $p$  nem igaz az *Indít* állapotban, akkor távolítsuk el a kapcsolatot. (Ez megóv minket attól, hogy végrehajtsunk egy olyan cselekvést, melynek előfeltételei hamisak.)
- **Redundáns cselekvések:** ha egy A cselekvés nem eredményez ok-okozati kapcsolatokat, akkor távolítsuk el a kapcsolataival együtt. (Ez lehetővé teszi számunkra, hogy kihasználjuk a szerencsés helyzeteket.)
- **Végrehajtatlan cselekvés:** ha egy (a *Befejez*-től eltérő) A cselekvésnek, melynek az előfeltételeit az *Indít* kielégíti, nincs (az *Indít*-től eltérő) cselekvés elé sorolva, és nem ütközik okozati kapcsolatokkal, akkor távolítsuk el A-t és ok-okozati kapcsolatait, majd adjuk vissza mint a végrehajtandó cselekvést.
- **Szükségtelen történelmi cél:** ha nincsenek nyílt előfeltételek és nincsenek a tervben cselekvések (azaz minden ok-okozati kapcsolat közvetlen az *Indít*-ból a *Befejez* állapotba mutat), akkor elérünk az aktuális célhalmazt. Távolítsuk el a célokat és az ezekbe mutató kapcsolatokat, hogy új célokat tegyünk lehetővé.

A 12.22. ábra a FOLYTONOS-RRT-ÁGENS-t mutatja be. Ennek ciklusa az „érzékelés, eltávolítás a folyamatból, cselekvés”. Tudásbázisában egy állandó tervet tart, és minden lépésben eltávolít egy folyamatot ebből a tervből. Ezután cselekszik (bár gyakran ez a cselekvés a *NoOp*) és ismétli a ciklust. Ez az ágens az 523–524. oldalon tárgyal újratervező ágenshez felsorolt problémák nagy részét képes kezelni. Nevezetesen valós időben cselekszik, kezeli a kedvező helyzeteket, saját célokat fogalmaz meg, és kezeli a váratlan eseményeket, melyek a jövőbeli terveket érintik.

```
function FOLYTONOS-RRT-ÁGENS(érzékelés) returns egy cselekvés
    static: terv, egy terv, kezdetben csak Indít, Befejez

    cselekvés ← NoOp (alapértelmezés)
    KÖVETKEZMÉNYEK[Start] = MÓDOSÍTÁS(KÖVETKEZMÉNYEK[Start], érzékelés)
    TÖRÉS-ELTÁVOLÍTÁS(terv) // várhatóan egy cselekvés módosítása
    return cselekvés
```

**12.22. ábra.** A FOLYTONOS-RRT-ÁGENS, egy folytonos, részben rendezett tervkészítő ágens. Egy érzékelés után az ágens eltávolít egy lépőssorozatot a folytonosan javított tervből, és visszaad egy cselekvést. Gyakran számos cselekvéssor-eltávolító tervkészítési lépéstre van szükség, mialatt csak a *NoOp*-ot adja vissza, mielőtt készen állna egy valós cselekvés végrehajtására.

## 12.7. MULTIÁGENS TERVKÉSZÍTÉS

Eddig egyágenses környezetekkel (single-agent environments) foglalkoztunk, amelyben az ágensünk egyedül van. Amikor más ágensek is jelen vannak a környezetben, az ágensünk egyszerűen hozzáveheti őket a környezetről alkotott modelljéhez, az alapvető algoritmusainak módosítása nélkül. Sok esetben azonban ez gyenge teljesítményhez vezetne, mert többi ágenssel való foglalkozás nem azonos a környezet kezelésével. Nevezetesen, a természet (feltételezhetően) közömbös az ágens szándékait illetően, míg más ágensek nem.<sup>15</sup> Ez az alfejezet ezen kérdések kezelésére a multiágens tervkészítést mutatja be.

A 2. fejezetben láthattuk, hogy a multiágens környezetek lehetnek **együttműködők** (**cooperative**) vagy **versengők** (**competitive**). Egy nagyon egyszerű kooperatív példával kezdünk: egy páros teniszcsapat tervkészítésével. Olyan tervek alkothatók, amelyek a csapat mindenki játékosának a cselekvéseit meghatározzák. Az ilyen tervek hatékony elkészítéséhez használható technikákat mutatjuk be. A hatékony tervkészítés hasznos, de nem garantálja a sikert. Az ágenseknek egyet kell érteniük a felhasznált tervben! Ez valamilyen **koordinációt** (**coordination**) feltételez, amit valószínűleg **kommunikációval** (**communication**) érhetünk el.

### Kooperáció: közös célok és tervezek

A páros tenisz játszó ágenscsapatoknak közös céljuk van, a meccs megnyerése, ami különböző részcélokat eredményez. Tegyük fel, hogy a játék egy pontján a közös céljuk az átütött labda visszaadása úgy, hogy legalább egyikőjük a hálót védi. Ezt a gondolatot egy **multiágens tervkészítési** (**multiagent planning**) problémával írhatjuk le, amint azt a 12.23. ábra is mutatja.

**Ágensek(*A, B*)**

**Kiindulás(*Ott(A, [Bal, Alapvonal]), Ott(B, [Jobb, Háló])*)**  $\wedge$

**Megközelít(*Labda, [Jobb, Alapvonal]*)**  $\wedge$  **Partner(*A, B*)**  $\wedge$  **Partner(*B, A*)**

**Cél(*Visszaadott(Labda)  $\wedge$  Ott(ágens, [x, Háló])*)**

**Cselekvés(*Ütés(ágens, Labda)*,**

**ELŐFELTÉTEL:*Megközelít(Labda, [x, y])  $\wedge$  Ott(ágens, [x, y])*  $\wedge$  **Partner(ágens, partner)****

**$\wedge \neg Ott(partner, [x, y])$**

**KÖVETKEZMÉNY:*Visszaadott(Labda)***

**Cselekvés(*Menj(ágens, [x, y])*,**

**ELŐFELTÉTEL:*Ott(ágens, [a, b])***

**KÖVETKEZMÉNY:*Ott(ágens, [x, y])  $\wedge \neg Ott(ágens, [a, b])$***

**12.23. ábra.** A páros tenisz probléma. Két ágens együtt játszik, és négy pozíció egyikében lehetnek: **[Bal, Alapvonal], [Jobb, Alapvonal], [Bal, Háló] és [Jobb, Háló]**. A labda visszaadható, ha pontosan egy játékos van a megfelelő helyen.

<sup>15</sup> Nem biztos, hogy az Egyesült Királyság lakói, ahol egy pikniknek már pusztán a tervezése is esőt garantál, egyetértenek ezzel.

Ez a jelölésrendszer két újdonságot vezet be. Először az *Ágens(A, B)* deklarálja, hogy két ágensünk van, A és B, akik a tervben szerepelnek. (Ebben a problémában a szemben álló játékosokat nem ágensnek vesszük.) Másodsor minden cselekvés az ágenst explicit módon, mint egy paramétert említi, mert pontosan követnünk kell, melyik ágens mit csinál.

A multiágens tervkészítési probléma megoldása egy **együttes terv** (*joint plan*), amely minden ágenshez tartalmaz cselekvéseket. Az együttes terv egy megoldás, ha a célt elérjük azáltal, hogy mindegyik ágens végrehajtja a számára kijelölt cselekvéseket. A következő terv a tenisz probléma egy megoldása :

#### TERV 1:

*A: [Megy(A, [Jobb, Alapvonal]), Út(A, Labda)]*

*B: [NoOp(B), NoOp(B)]*

Ha minden ágens ugyanazzal a tudásbázissal rendelkezne, és ez lenne az egyetlen megoldás, akkor minden rendben lenne; minden ágens meghatározná a megoldást, és együttesen végrehajtanák. Az ágensek számára sajnálatos (és hamarosan látni fogjuk, hogy ez miért sajnálatos), hogy van egy másik terv is, ami az elsőhöz hasonlóan kielégíti a célt:

#### TERV 2:

*A: [Megy(A, [Bal, Háló]), NoOp(A)]*

*B: [Megy(B, [Jobb, Alapvonal]), Út(B, Labda)]*

Ha az A a 2. tervet választja, és B az 1-et, akkor senki sem fogja visszaadni a labdát. Fordítva, ha A az 1-et és B a 2-t választja, akkor várhatóan összeütköznek egymással, senki nem adja vissza a labdát, és a háló is fedezetlenül marad. Ezért a helyes együttes terv létezése nem jelenti, hogy a célt el is érjük. Az ágenseknek szükségük van egy **koordinációs** (*coordination*) mechanizmusra, hogy *ugyanazt* az együttes tervet érjék el, sőt mi több, az ágensek közös tudással is kell rendelkezzenek (lásd 10. fejezet) arról, hogy melyik együttes tervet hajtják végre.

## Többtestű tervezés

Ez az alfejezet a megfelelő együttes tervek megalkotására fókusztál, elődázva egyelőre a koordinációs kérdéskört. Ezt **többtestű tervkészítésnek** (*multibody planning*) nevezzük, és tulajdonképpen az a tervkészítési probléma, mellyel egy egyedülálló centralizált ágens áll szemben, mely cselekvéseket diktálhat minden fizikai entitásnak. Egy valóban multiágens esetben ez lehetővé teszi minden ágens számára, hogy felderítse melyek a lehetséges együttes tervek, melyek együttes végrehajtása sikeres lenne.

Megközelítésünkben a többtestű tervkészítést a 11.3. alfejezetben bemutatott részben rendezett tervkészítésre alapozzuk. Az egyszerűség kedvéért teljes megfigyelhetőséget feltételezzünk. Van még egy kérdés, ami nem merül fel az egyedülálló ágens esetén: a környezet a továbbiakban nem igazán **statikus** (*static*), mert a többi ágens cselekedhet, amíg egy bizonyos ágens gondolkozik. Ezért foglalkoznunk kell a **szinkronizációval** (*synchronization*). Az egyszerűség kedvéért feltételezzük, hogy minden cselekvés végrehajtási ideje azonos, illetve hogy az összetett terv minden pontján a cselekvések végrehajtása szimultán történik.

Bármely időpontban minden ágens pontosan egy cselekvést hajt végre (beleértve a *NoOp*(Üres) operátort is). Ezen konkurens cselekvések halmazát **együttes cselekvésnek (joint action)** nevezzük. A tenisz problémában (530. oldal) például a *(NoOp(A), Ut(B, Labda))* az A és B ágenseknek egy együttes cselekvése. Egy együttes terv együttes cselekvések részben rendezett gráfja. Például a tenisz probléma második tervé az együttes cselekvések következő sorozatával írható le:

*(Megt(A, [Bal, Háló]), Megt(B, [Jobb, Alapvonal]))*  
*(NoOp(A), Ut(B, Labda))*

Végrehajthatnánk a tervkészítést a hagyományos részben rendezett tervkészítő algoritmussal, azt az összes lehetséges együttes cselekvés halmazára futtatva. Az egyetlen probléma a halmaz mérete: 10 cselekvéssel és 5 ágenssel  $10^5$  együttes cselekvést kapunk. Elég unalmas lenne az összes cselekvés előfeltételét és következményét helyesen definiálni, és nem lenne hatékony egy ilyen nagy halmazzal tervezni.

Egy másik lehetőség az együttes cselekvések implicit definiálása úgy, hogy minden egyes cselekvéshez leírjuk, hogy milyen kölcsönhatásban áll a többi lehetséges cselekvéssel. Ez egyszerűbb lenne, hiszen a legtöbb cselekvés független a legtöbb másiktól, ezért csak azon néhány cselekvést kellene listáznunk, amelyek valójában kölcsönhatásban vannak. Ezt a hagyományos STRIPS vagy ADL cselekvésleírások egy új tulajdonsággal való kiterjesztésével tehetjük meg: ez a **konkurens cselekvések listája (concurrent action list)**. Ez nagyon hasonlít a cselekvés leírásában szereplő előfeltételhez, kivéve, hogy az állapotváltozók leírása helyett ez cselekvéset ír le, amelyeket egyszerre kell, vagy éppen nem szabad egyszerre végrehajtani. Például a *Találat* cselekvés a következőképpen írható le:

*Cselekvés(Ut(A, Labda),*  
*KONKURENS: $\neg$ Ut(B, Labda)*  
*ELŐFELTÉTEL: Megközelít(Labda, [x, y])  $\wedge$  Ott(A, [x, y])*  
*KÖVETKEZMÉNY: Visszaadott(Labda))*

Itt egy tiltó egyidejűségi megkötésünk van, azaz az *Ut* cselekvés végrehajtása alatt egy másik ágens nem hajthat végre egy másik *Ut* cselekvést. Hasonlóképpen *megkövetelhetünk* egyidejű cselekvést. Például amikor két ágensre van szükség egy üditőkkel teli hűtőtáska teniszpályához cipeléséhez. Ezen cselekvés leírása azt fejezi ki, hogy az A ágens nem hajthatja végre a *Cipel* cselekvést, ha csak nincs egy másik B ágens, aki egyidejűleg egy *Cipel* cselekvést hajt végre ugyanazon hűtőtáskával:

*Cselekvés(Cipel(A, hűtőtáska, itt, ott),*  
*KONKURENS: Cipel(B, hűtőtáska, itt, ott)*  
*ELŐFELTÉTEL: Ott(A, itt)  $\wedge$  Ott(Hűtőtáska, itt)  $\wedge$  Hűtőtáska(hűtőtáska)*  
*KÖVETKEZMÉNY: Ott(A, ott)  $\wedge$  Ott(hűtőtáska, ott)*  
 *$\wedge$   $\neg$ Ott(A, itt)  $\wedge$   $\neg$ Ott(hűtőtáska, itt))*

Ezen reprezentáció segítségével a részben rendezett tervkészítőhöz nagyon közelíti a tervkészítő hozható létre. Mindössze három különbség van:

1. Az  $A \prec B$  temporális rendezési relációt kiegészítendő megengedjük az  $A = B$ -t és az  $A \preceq B$ -t, ami „egyidejűséget”, illetve „megelőzőséget vagy egyidejűséget” jelent.

2. Amikor egy új cselekvés egyidejű cselekvésekkel igényel, példányosítani kell ezen cselekvésekkel a terv új vagy létező cselekvéseinek felhasználásával.
3. A tiltottan egyidejű cselekvések a megkötések újabb forrásai. minden megkötést fel kell oldani az ütköző cselekvések egymás előre vagy mögött rendelésével.

Ez a reprezentáció a többtörzsű problémakörökhoz alkalmazott, részben rendezett tervkészítővel ekvivalens. Kiterjeszhetnénk ezt a megközelítést az utolsó két fejezet finomításával – HFH-ek részleges megfigyelhetőség, feltételek, végrehajtás-monitorozás és újratervezés –, de ez túlmutat a könyv célkitűzésein.

## Koordináló mechanizmusok

A legegyszerűbb módszer, amivel ágensek egy csoportja biztosíthatja az egyetértést egy együttes tervben, hogy egy **konvenciót (convention)** fogadnak el az együttes cselekvés megkezdése előtt. A konvenció az együttes tervek közötti választásra vonatkozó bármely olyan megkötés, amely túlmutat az alapmegkötésen, amely szerint egy együttes tervnek működni kell, ha minden ágens alkalmazza. Például a „maradj a saját térfeléden” konvenció a páros partnereket a második terv kiválasztására sarkallja, míg az „egy játékos mindig a hálónál marad” konvenció az egyes tervhez vezetné őket. Néhány konvenció, mint az út megfelelő oldalán való vezetés, olyan széles körben elterjedt, hogy ezeket már **társadalmi törvényeszerűségekként (social laws)** kezeljük. Az emberi nyelvek szintén konvencióknak tekinthetők.

Az előző alfejezet konvenciói alkalmazási terület specifikusak és a cselekvésleírások megkötéseivel implementálhatók, amelyekkel kizártuk a konvenciók megszegéseit. Egy általánosabb megközelítés, hogy problémakör-független konvenciókat használunk. Például ha mindegyik ágens ugyanazon többtestű tervkészítő algoritmust futtatja ugyanazon bemenetekkel, követi azt a konvenciót, miszerint az elsőként megtalált, használható összetett tervet hajtsa végre, miközben biztos benne, hogy a többi ágens ugyanezzel a választással él. Egy sokkal robusztusabb, de jóval költségesebb stratégia lenne az összes összetett terv elkészítése, majd az ebből való választás (mondjuk annak kiválasztása, amelynek leírt reprezentációja ábécé-sorrendben az első).

Konvenciók szintén adódhannak a folyamatok evolúciós fejlődésével. Például a közösségen belüli rovarok nagyon kidolgozott összetett terveket hajtanak végre, melyeket a kolónia egyedeinek közös genetikus összeállítása biztosít. Az azonosságot szintén nyomatékosítja a tény, hogy a konvencióktól való eltérés csökkenti az evolúciós alkalmaságot, így bármely alkalmazható összetett terv stabil egyensúlyi helyzetet eredményezhet. Hasonló megfontolások alkalmazhatók az emberi nyelv fejlődésére, ahol nem az a fontos, hogy egyes személyek milyen nyelvet beszélnek, hanem az a tény, hogy minden személy ugyanazt a nyelvet beszéli.

Egy utolsó példa a madarak gyülekezési viselkedése. Megfelelő szimulációt nyerhetünk, ha minden egyes madár ágens (melyet néha madaroidnak vagy roidnak nevezünk) a következő három szabályt követi valamelyen kombináció alapján:

1. Különválás: távolodjunk el a szomszédoktól, amikor túl közel kerülünk;
2. Kohézió: kerüljünk átlagos távolságra a szomszédainktól;
3. Irányultság: kerüljünk a szomszédokhoz képest átlagos orientációba (haladási irányba).

Ha minden madár azonos szabályrendszert hajt végre, a csapat repülése egy kialakuló viselkedést (**emergent behavior**) mutat, hasonlóan egy nagyjából azonos sűrűségű pszeudodomerev test viselkedéséhez, amely nem bomlik fel az idő során. Mint a bogaraknál, itt sincs szükség arra, hogy minden ágens ismerje a teljes összetett tervet, ami a többi ágens cselekvését modellezí.

A konvenciók ahelyett, hogy minden egyes új problémához kifejlesztenénk őket, tipikusan egyedi multiágens tervkészítő problémák egy egész csoportjához készülnek. Ez rugalmatlansághoz és hibához vezethet, amit néha láthatunk, ha a páros tenisz játszmában a labda nagyjából azonos távolságra van a két partner között. Egy alkalmazható konvenció hiányában az ágensek kommunikálhatnak (**communication**), hogy egy közös tudást érjenek el az alkalmazható összetett tervből. Például a teniszpáros egy játékosa felkiált hat „Enyém!” vagy „Tiéd!”, hogy jelezze a preferált tervet. A kommunikációs mechanizmusokat nagyobb mélységben a 22. fejezetben tárgyaljuk, ahol megfigyelhetjük, hogy a kommunikáció nem szükségszerűen szóbeli információcserét jelent. Például egy játékos úgy is közölheti az általa előnyben részesített összetett tervét a másikkal, hogy végrehajtja annak kezdő részletét. A teniszjátszma problémánkban, ha az A ágens elindul a hálóhoz, akkor a B ágens rákényszerűl, hogy visszamenjen az alapvonalhoz megütni a labdát, mivel a második terv az egyetlen összetett terv, ami A hálóhoz mozgásával kezdődik. Ez az irányítási megközelítés, amelyet gyakran a **terv felismerésének** (**plan recognition**) nevezünk, akkor működik, ha egyetlen cselekvés (vagy cselekvések rövid sorozata) elegendő, hogy egyértelműen azonositsuk az összetett tervet.

Azt a terhet, amely biztosítja, hogy az ágensek egy sikeres összetett tervhez jussanak, vagy az ágensek készítőire, vagy magukra az ágensekre helyezhetjük. Az első esetben mielőtt az ágensek tervezni kezdenének, az ágens készítőjének bizonyítania kell, hogy az ágensek szabályai és stratégiái sikeresek lesznek. Az ágensek önmagukban lehetnek reaktívak, amennyiben ez működik az őket körülvevő környezetben, valamint ha nem kell, hogy pontos modelljük legyen a többi ágensról. Az utóbbi esetben az ágensek megfontoltak: bizonyítaniuk vagy demonstrálniuk kell, hogy a tervük hatékony lesz, figyelembe véve a többi ágens megfontolásait. Például egy A és B logikai ágenst tartalmazó környezetben mindenketten rendelkezhetnek a következő definícióval:

$$\forall p, s \quad \text{Megfelelő}(p, s) \Leftrightarrow \text{KözösTudás}(\{A, B\}, \text{Eléri}(p, s, \text{Cél}))$$

Ez azt mondja, hogy bármely  $s$  szituációban, a  $p$  terv egy alkalmazható összetett terv, ha az ágensek közösen tudják, hogy a  $p$  eléri a célt. További axiómákra van szükségünk, hogy biztosítsuk egy adott összetett terv végrehajtására vonatkozó közös szándék (**joint intention**) együttes ismeretét. Az ágensek csak ebben az esetben kezdhették a cselekedni.

## Versengés

A többágenses környezeteknek nem mindegyike áll együttműködő ágensekből. Az egy-másnak ellentmondó működésű ágensek **versenyben** (**competition**) vannak egymással. Erre egy példát jelentenek a két játékos igénylő, de zérus összegű játékok, mint amilyen a sakk. A 6. fejezetben láthattuk, hogy a sakközösségek figyelembe kell vennie az ellenfél következő néhány lehetséges lépését is. Azaz egy versengő környezetben levő ágens-

nek (a) fel kell ismernie, hogy vannak más ágensek, (b) meg kell határoznia a többi ágens lehetséges terveit, (c) meg kell határoznia, hogy a többi ágens hogyan tervez a saját terveivel való kölcsönhatást, és (d) el kell döntenie, hogy ezek fényében mi a legjobb cselekvés. A versengés, hasonlóan az együttműködéshez, a többi ágens tervének modelljét igényli. Másrészről, egy versengő környezetben nincs elhatározás egy összetett terv mellett.

A 12.4. alfejezet felvázolta a játékok és a feltételes tervkészítési problémák közötti analógiát. A 12.10. ábrán szereplő feltételes tervkészítő algoritmus olyan tervet készít, amely a környezetről a legrosszabb esetet feltételezve is működik, így olyan versenyhelyzetekben is alkalmazható, ahol az ágenst csak a siker vagy a bukás érdekli. Amikor az ágens és ellenfelei a terv *költségeivel* is foglalkoznak, akkor a minimax megoldás a helyénvaló. Eddig nagyon kevés munkát fordítottak a minimax módszer olyan módszerekkel való kombinálására, mint a részben rendezett tervkészítés vagy a HFH-tervkészítés, ami túlmutat a 6. fejezetben használt állapottér-keresési modellen. A versengés kérdésére a játékelméettel foglalkozó 17.6. alfejezetben még visszatérünk.

## 12.8. ÖSSZEFoglalás

Ez a fejezet a valódi világban alkalmazott tervkészítés és cselekvés nehézségeivel foglalkozott. Összefoglaljuk a fő megállapításokat.

- A legtöbb cselekvés erőforrásokat (**resources**) használ, például pénzt, benzint vagy nyersanyagokat. Ezen erőforrásokat készletekre vonatkozó mérőszámokként cél-szerű kezelni, mintsem hogy megpróbálunk következetet mondjuk a világ minden egyes pénzérémjéről és papírpénzéről. A cselekvések generálhatnak és elfogyaszthatnak erőforrásokat, és rendszerint olcsó és hatékony, ha a további finomítások végrehajtása előtt a részleges terveket ellenőrzük, hogy az erőforrásokra vonatkozó korlátozásokat kielégítik-e.
- Az idő az egyik legfontosabb erőforrás. Ez speciális ütemező algoritmusokkal kezelhető, vagy az ütemezést integrálhatjuk a tervkészítésbe.
- A hierarchikus feladat háló (**hierarchical task network**) (HFH) tervkészítés lehetővé teszi az ágens számára, hogy dekompozíciós szabályok formájában tanácsokat fogadjon el a feladatkör tervezőjétől. Ez lehetővé teszi, hogy nagyon nagy terveket készítsünk, amelyekre számos valódi világ-beli alkalmazáshoz szükség van.
- A hagyományos tervkészítő algoritmusok teljes és helyes információt és determinisztikus, teljesen megfigyelhető környezetet feltételeznek. A legtöbb problémakörre nem igaz ez a feltételezés.
- A hiányos információ kezelhető azáltal, hogy érzékelő cselekvéseket tervezünk, amelyekkel megszerezzük a szükséges információt. A **feltételes tervek** (**conditional plans**) lehetővé teszik az ágens számára, hogy a végrehajtás során érzékelje a környezetét, hogy eldönthesse, mely ágát követi a tervnek. Néhány esetben **érzékelőmentes** (**sensorless**) vagy **alkalmazkodó tervkészítés** (**conformant planning**) használható az érzékelést nem igénylő terv elkészítéséhez. Mind az érzékelőmentes, mind pedig a feltételes tervek előállíthatók a **hiedelmi állapotok** (**belief states**) terében történő kereséssel.

- A helytelen információk kielégítetlen előfeltételeket eredményeznek a cselekvések és a tervezés számára. A végrehajtás-monitorozás (*execution monitoring*) a terv sikeres befejezése érdekében érzékeli az előfeltételek megszegéseit.
- Az újratervező ágens (*replanning agent*) végrehajtás-monitorozást használ és szükség esetén javításokat szűr be.
- A folytonos tervkészítő (*continuous planning*) ágens előrehaladása során új célokat alkot és valós időben reagál.
- Multiágens (*multiagent*) tervezésre van szükség, amikor a környezetben más ágensek is találhatók, melyekkel együttműködni vagy versengeni kell, vagy amelyeket koordinálni kell. A többtestű (*multibody*) tervkészítés az együttes cselekvésleírások hatékony dekompozícióját felhasználva együttes terveket alkot, de ki kell egészíteni valamelyen koordinációval, amennyiben két együttműködő ágensnek egyet kell érteni abban, hogy melyik együttes tervet hajtsák végre.

## Irodalmi és történeti megjegyzések

A folytonos idővel való tervkészítést elsőként DEVISER vetette fel (Vere, 1983). Az időnek a tervezében való szisztematikus reprezentációjával Dean és társai dolgoztak a FORBIN rendszerben (Dean és társai, 1990). A NONLIN+ (Tate és Whiter, 1984) és a SIPE (Wilkins, 1988; 1990) kezelni tudta a korlátos erőforrások különböző tervlépésekhez való hozzárendelését. Az O-PLAN (Bell és Tate, 1985) HFH-tervkészítő az idő- és erőforrás-korlátozások egységes és általános reprezentációját tartalmazta. A szövegben említett Hitachi-alkalmazás kiegészítéseképpen, az O-PLAN módszert a Price Waterhouse-ban a szoftverbeszerzési tervkészítéshez alkalmazták, míg a Jaguar Autógyárban a hátsó tengely összeszerelési tervének készítésénél. Számos hibrid tervkészítő és ütemező rendszert alkalmaztak: az Isis (Fox és társai, 1982; Fox, 1990) egy ütemezési feladatban alkalmazták a Westinghouse-nál, a GARI (Descotte és Latombe, 1985) mechanikus alkatrészek gépesítését és készítését tervezte, a FORBIN-t gyári vezérlésre használták, végezetül a NONLIN+ hajózási logisztikai tervkészítéshez alkalmazták.

Az 1980-as években megindult elméleti munka tetőzése után, napjainkban ismét a temporális tervkészítés kerül előtérbe, amikor az új algoritmusok és a megnövekedett feldolgozási teljesítmény lehetővé tette a gyakorlati felhasználást. A két tervkészítő a SAPA (Do és Kambhampati, 2001) és a T4 (Haslum és Geffner, 2001) előrefelé történő állapottér keresést alkalmaztak, amit az időtartammal és erőforrásokkal rendelkező cselekvések kezeléséhez egy kifinomult heurisztikával egészítettek ki. Egy másik lehetőség a nagyon költséges cselekvésleíró nyelvek használata, melyeket kézzel írt és alkalmazási terület specifikus heurisztikákkal irányíthatunk, amint az az ASPEN (Fukunaga és társai, 1997), a HSTS (Jonsson és társai, 2000) és az IxTeT (Ghallab és Laruelle, 1994) esetén is történik.

Az ütemezés az űrkutatásban nagyon hosszú múltra tekint vissza. A T-SCHED (Drabble, 1990) rendszert az UoSAT-II műhold küldetésvezérlő parancssorozatainak ütemezésére használták. Az O-PLAN-en alapuló OPTIMUM-AJV (Aarup és társai, 1994) és a PLAN-ERS1 (Fuchs és társai, 1990) az űrrépülőgép-összeszerelésben a megfigyelési tervkészítésben használták fel az Európai Űrkutatási Ügynökségnél (ESA).

A SPIKE rendszert (Johnston és Adorf, 1992) a NASA-nál a Hubble-űrteleszkóp megfigyelés-tervezéséhez használták, míg az ūrsíkló földi feldolgozásütemező rendszer (Deale és társai, 1994) emberi erőforrás ütemezést végez egészen 16 000 munkás-műszak értéig. A Remote Ágens (Muscettola és társai, 1998) lett az első önálló tervkészítő és ütemező, mely ūrsíklót vezérelt, mialatt a Deep Space One ūrszonda fedélzetén repült 1999-ben. Az emberi erőforrás ütemezés irodalmát Vaessens és társai (Vaessens és társai, 1996) foglalták össze, az elméleti eredményeket Martin és Shmoys (Martin és Shmoys, 1996) jelentette meg.

A STRIPS programban található a **macrops** – elemi lépések sorozatából álló „makró-operátorok” (**macro operators**) – tanulásra szolgáló eszköztár, az első hierarchikus mechanizmusnak tekinthető (Fikes és társai, 1972). A hierarchiát szintén alkalmazták a LAWALY rendszerben (Siklossy és Dreussi, 1973). Az ABSTRIPS rendszer (Sacerdoti, 1974) az **absztrakciós hierarchia** (**abstraction hierarchy**) ötletét vezette be, ahol a magasabb szintű tervezés megengedte a cselekvések alacsonyabb szintű előfeltételeinek figyelmen kívül hagyását, hogy egy működő terv általános struktúráját származtat-hassa. Austin Tate PhD-tézise (Tate, 1975b) és Earl Sacerdoti (Sacerdoti, 1977) munkája a HTM tervkészítés modern formájának alapötleteit vezette be. Számos gyakorlati tervkészítő, beleértve az O-PLAN és a SIPE tervkészítőket, HFH-tervkészítő. Yang (Yang, 1990) a HFH-tervkészítést hatékonnyá tevő cselekvések jellemzőit tárgyalja. Erol, Hendler és Nau (Erol és társai, 1994, 1996) egy teljes hierarchikus dekompozíciót alkalmazó tervkészítőt mutat be, valamint számos komplexitásbeli eredményt a tiszta HFH-tervkészítőkhöz. Más szerzők (Ambros-Ingerson és Steel, 1988); Young és társai (Young és társai, 1994); Barrett és Weld (Barrett és Weld, 1994); Kambhampati és társai (Kambhampati és társai, 1998) az ebben a fejezetben bemutatott hibrid megközelítést javasolták, melyben a dekompozíciók mindenkor a részben rendezett tervkészítéshez használható finomítások egy másik formáját jelentik.

A STRIPS-ben szereplő makróoperátorokkal kezdődően a hierarchikus tervkészítés egyik célja a korábbi, általánosított tervek formájában rendelkezésre álló tervkészítési tapasztalat újrahasznosítása. A magyarázatalapú tanulás (**explanation-based learning**) technikáját, amit részleteiben a 19. fejezet mutat be, számos rendszerben, mint a SOAR (Laird és társai (Laird és társai, 1986) és PRODIGY (Carbonell és társai, 1989) alkalmazták a korábban elkészített tervek általánosítására. Egy másik megközelítés, hogy a korábban kiszámított terveket eredeti formájukban tároljuk el, majd az új hasonló feladatokban az eredeti feladat analógiájára újrahasznosítjuk őket. Ezt a megközelítést esetalapú **tervkészítésnek** (**case-based planning**) (Carbonell, 1983; Alterman, 1988; Hammond, 1989) nevezünk. Kambhampati (Kambhampati, 1994) érvélete szerint az esetalapú tervkészítést a finomító tervkészítéshez hasonlóan kellene vizsgálni. Emellett lefekteti az esetalapú részben rendezett tervkészítés formális alapjait.

A valós környezetek kiszámíthatatlanságát és részleges megfigyelhetőségét elsőként tervkészítési technikákat használó robotikai projektekben – beleértve a Shakey (Fikes és társai, 1972) és a FREDDY (Michie, 1974) projekteket – ismerték fel. A problémakör több figyelmet kapott McDermott nagy hatású *Tervezés és cselekvés* (*Planning and Acting*) c. cikkének megjelenését követően (McDermott, 1978a).

A korai, feltételeket és ciklusokat nem tartalmazó tervkészítők nem tükröztek pontossan a feltételes tervkészítés megközelítésmódját, de ennek ellenére néha kényszerítően reagáltak a környezet bizonytalanságára. Sacerdoti NOAH rendszere kényszerítést hasz-

nált a „kulcsok és dobozok” (keys and boxes) probléma megoldásában, egy olyan tervkészítési kihívásban, ahol a tervkészítő keveset tud a kiinduló állapotról. Mason (Mason, 1993) kijelentette, hogy a robotikai tervkészítés nélkülözni tudja és nélkülöznie is kell az érzékelést, és bemutatott egy érzékelőmentes tervet, ami a kiindulási pozíciótól függetlenül képes egy eszköz csuklómozdulatok sorozatával az asztal egy megadott pozíciójába mozgatni. Ezt az ötletet a robotika kontextusában tárgyaljuk (lásd 25.17. ábra).

Goldman és Boddy (Goldman és Boddy, 1996) bevezették az alkalmazkodó tervkészítés (**conformant planning**) fogalmát, az érzékelőmentes tervkészítőkre, melyek a bizonytalanságot a környezet ismert állapotokba való kényszerítésével kezelik, meggyezve, hogy az érzékelőmentes tervek gyakran hatékonyak még akkor is, ha az ágens rendelkezik érzékelőkkel. Az első megfelelően hatékony alkalmazkodó tervkészítő Smith és Weld (Smith és Weld, 1998) Conformant Graphplan (CGP) rendszere volt. Ferraris és Giunchiglia (Ferraris és Giunchiglia, 2000), valamint Rintanen (Rintanen, 1999) függetlenül fejlesztettek ki SATPLAN-alapú alkalmazkodó tervkészítőket. Bonet és Geffner (Bonet és Geffner, 2000) a hiedelmi állapotok terén végzett heurisztikus keresésalapú alkalmazkodó tervkészítőt mutat be, olyan ötletekre alapozva, melyeket először 1960-ban fejlesztettek ki a részlegesen megfigyelhető Markov döntési folyamatokhoz (Partially Observable Markov Decision Process – POMDP) (lásd 17. fejezet). Jelenleg a leggyorsabb hiedelmi állapot alkalmazkodó tervkészítők, mint például a HSCP (Bertoli és társai, 2001a) bináris döntési diagramokat (BDD) (Bryant, 1992) használnak a hiedelmi állapotok leírására, valamint öt nagyságrenddel gyorsabbak, mint a CGP.

A WARPLAN-C (Warren, 1976), ami a WARPLAN egy változata, volt az első feltételes cselekvéseket használó tervkészítők egyike. Olawski és Gini (Olawski és Gini, 1990) a feltételes tervkészítés fő problémáit vetette fel.

A fejezetben bemutatott feltételes tervkészítési megközelítés a Jimenez és Torras (Jimenez és Torras, 2000), valamint Hansen és Zilberstein (Hansen és Zilberstein, 2001) által kifejlesztett ciklikus ÉS-VAGY gráfok hatékony keresési algoritmusán alapul. Bertoli és társai (Bertoli és társai, 2001b) egy BDD-alapú megközelítést mutatnak be, ami ciklikus feltételes terveket készít. A C-BURIDAN rendszer (Draper és társai, 1994) egy folytonos tervkészítést végez valószínűségi kimenetű cselekvésekre, melynek problémája a POMDP vezérlése során is felmerült (lásd 17. fejezet). Szoros kapcsolat van a feltételes tervkészítés és az automatizált programkészítés között. A 9. fejezet számos ilyen hivatkozást tartalmaz. A két területet egymástól függetlenül tárták fel, aminek oka az az óriási költsékgülönbég, ami egy gépi utasítás végrehajtása, valamint robot- vagy manipulátorcselekvések végrehajtása között fennáll. Linden (Linden, 1991) a két terület közötti tudásmegosztást szorgalmazza.

Történelmi távlatból már látható, hogy hogyan vezetett a két fő klasszikus tervkészítő algoritmus kibővített verziókhoz a bizonytalan környezetek kezelése céljából. A keresésalapú technikák a hiedelmi állappotter keresésre vezettek (Bonet és Geffner, 2000). A SATPLAN algoritmus a sztochasztikus SATPLAN (Majercik és Littman, 1999) algoritmusra, valamint a kvantorokat alkalmazó logikai tervezésre vezetett (Rintanen, 1999). A részben rendezett tervkészítésből az UWL (Etzioni és társai, 1992), a CNLP (Peot és Smith, 1992) és a CASSANDRA (Pryor és Collins, 1996) származik. A GRAPHPLAN, az érzékeléses GRAPHPLAN (sensory GRAPHPLAN-SGP) (Weld és társai, 1998) algoritmusra vezetett, de a teljesen valószínűségi GRAPHPLAN algoritmus még kifejlesztésre vár.

A végrehajtás monitorozás első fő megjelenése a PLANEX (Fikes és társai, 1972), ami egy STRIPS tervkészítővel működött a Shakey robot irányításán. A PLANEX háromszög táblákat használt – ami egy szükségszerűen hatékony tárolási mechanizmus a terv előfeltételeinek tárolására a terv minden pontján –, hogy teljes újratervezés nélkül helyreállhasson egy részleges végrehajtási hiba után. A végrehajtás Shakey-modelljét a 25. fejezetben tárgyaljuk tovább. A NASL tervkészítő (McDermott, 1978a) a tervkészítési problémát egyszerűen, mint egy komplex cselekvés végrehajtásának specifikációját tekintette, így a tervezés és végrehajtás teljesen egységesé vált. Az összetett cselekvések levezetésére tételebizonyítást használ.

A SIPE (System for Interactive Planning and Execution Monitoring) (Wilkins, 1988, 1990) volt az első tervkészítő, ami az újratervezés problémáját szisztematikusan kezelte. Különböző problémakörök demonstrációs feladataiban is felhasználták, például az ürsiklóhordozó fedélzeti terv készítésére vagy emberi erőforrás ütemezésére egy ausztrál sörgyárban. Egy másik vizsgálat a SIPE rendszert többemeletes épületek tervezésében alkalmazta, ami az egyik legbonyolultabb problémakör, amit tervkészítő valaha kezelt.

Az IPEM (Intergrated Planning, Execution, and Monitoring) (Ambros-Ingerson és Steel, 1988) volt az első rendszer, ami integrálta a részben rendezett tervkészítést és a végrehajtást, hogy folytonos tervkészítő ágensre jusson. Az általunk tárgyalott folytonos, részben rendezett tervkészítő ágens (CONTINUOUS-POP-AGENT) az IPEM a PUCCINI (Golden, 1998) és a CYPRESS (Wilkins és társai, 1995) tervkészítők ötleteit kombinálja.

Az 1980-as évek közepén néhányan úgy gondolták, hogy a részben rendezett tervkészítés és a kapcsolódó technikák soha nem futhatnak eléggy gyorsan, hogy egy valódi-világ-beli ágens számára használható viselkedést biztosítsanak (Agre és Chapman, 1987). Ezek helyett a **reaktív tervkészítő** (*reactive planning*) rendszereket vezették be. Ezek alapesetben reflexszerű ágensek, esetenként belső állapottal, melyek számos különböző feltétel-cselekvés szabályábrázolás segítségével implementálhatók. Brooks (Brooks, 1986) bennfoglalási architektúrája (lásd. 7. és 25. fejezet) rétegzett véges állapotú állapotpéket használt, lábbal és kerékkel ellátott robotok mozgásának és akadályok elkerülésének a szabályozására. Pensi (Agre és Chapman, 1987) egy (teljesen megfigyelhető) videojátéket tudott játszani, az aktuális célok „látott” reprezentációja, az ágens belső állapota, valamint kétértékű áramkörök kombinálásával.

Az „univerzális tervek” (Schoppers, 1987) a reaktív tervkészítés keresőtábla módszerént került kifejlesztésre, de mint kiderült, ez a Markov döntési folyamatokban használt eljárásmód (policy) ötletének újrafelfedezése volt. Egy univerzális terv (vagy egy eljárásmód) bármely állapotban az adott állapotban végrehajtandó cselekvésre mutató leképezési tartalmazza. Ginsberg (Ginsberg, 1989) lelkes támadást indított az univerzális tervek ellen, ami kezelhetetlenségi eredményeket is tartalmazott a reaktív tervkészítési probléma néhány alakjára. Erre Schoppers 1989-ben egy hasonlóan lelkes választ adott (Schoppers, 1989).

Mint a legtöbb esetben a hibrid megközelítés feloldja az ellentmondást. Jól tervezett hierarchiák használatával a HFH tervkészítők, mint a PRS (Georgeff és Lansky, 1987) és a RAP (Firby, 1996), valamint a folytonos tervkészítő ágensek használható válaszidőket és összetett hosszú távú terveket érhetnek el számos problémakörben.

A multiágens tervkészítés napjainkban vált népszerűvé, bár hosszú történelme van. Konolige (Konolige, 1982) a multiágens tervkészítés elsőrendű logikai leírását adta meg, miközött Pednault (Pednault, 1986) egy STRIPS-alakú leírást készített. A kölcsönös szándék

ötlete, ami elengedhetetlen, ha az ágensek együttes tervet hajtanak végre, a kommunikációs viselkedés kutatásából ered (Cohen és Levesque, 1990; Cohen és társai, 1990). Az általunk bemutatott többtestű részben rendezett tervkészítés Boutilier és Brafman (Boutilier és Brafman, 2001) munkáján alapul.

Az itt szereplő, a multiágens tervkészítésben az együttműködést célzó munka bemutatása éppcsak a jéghegy csúcsa. Durfee és Lesser (Durfee és Lesser, 1989) azt tárgyalja, hogy a feladatok hogyan oszthatók meg az ágensek között egyezkedés útján. Kraus és társai (Kraus és társai, 1991) egy Diplomacyt – egy egyezkedést, együttműködést, felbomlást és ügyeskedést igénylő táblajátékot – játszó rendszert mutatnak be. Stone (Stone, 2000) bemutatja, hogy az ágensek hogyan tudnak csapattagként együttműködni a robotfoci versengő, dinamikus, részben megfigyelhető környezetében. A (Weiss, 1999) a multiágens rendszerek egy könyvhosszúságú áttekintése. Az 533. oldal mód modellje Reynoldsnak köszönhető (Reynolds, 1987), aki ennek a *Batman visszatér* (*Batman returns*) c. film denevérseregeire és pingvinszapatára való alkalmazásával Oscar-díjat is nyert.

## Feladatok

- 12.1.** Vizsgáljuk meg figyelmesen a 12.1. alfejezetben az idő és az erőforrások reprezentációját.
- Miért jobb, ha egy cselekvés következménye az *Időtartam(d)*, mint ha a cselekvésnek egy *IDŐTARTAM:d* alakú külön mezője lenne? (Segítség: vegyük figyelembe a feltételes következményeket és a diszjunktív következményeket.)
  - Az **ERŐFORRÁS: m** miért egy külön mező a cselekvésben, nem pedig egy következmény?
- 12.2.** Egy fogyóeszköz (*consumable resource*) egy olyan erőforrás, ami (részlegesen) felhasználásra kerül a cselekvésben. Például a motor beszerelése az autóba csavarokat igényel. Miután a csavarokat felhasználtuk, már nem használhatók további rögzítésekhez.
- Magyarázza meg, hogyan módosítsuk a 12.3. ábra reprezentációját úgy, hogy 100 kiinduló csavarunk legyen az  $E_1$  motor 40 csavart, az  $E_2$  pedig 50 csavart igényel. A + és - szimbólumok felhasználhatók az erőforrásokat leíró következményleírásokban.
  - Mutassa meg, hogy a részben rendezett tervkészítésben alkalmazott okozati kapcsolatok és cselekvések közötti **ütközés (conflict)** definícióját hogyan kell módosítani, hogy a fogyóeszközöket kezelje.
  - Néhány cselekvés, például a gyár ellátása csavarokkal vagy egy autó feltankolása, *növelheti* az erőforrások készletét. Egy erőforrás monoton nem növekvő, ha egyetlen cselekvés sem növeli. Magyarázza meg, hogyan használható ez a tulajdonság a keresési tér megmetszéséhez.
- 12.3.** Adjon dekompozíciót a 12.7. ábra *KivitelezőtVeszFel* és *EngedélytSzerez* lépései-re, és mutassa meg, hogy a dekomponált részterek hogyan állnak össze egy teljes tervvé.

- 12.4. Adjon példát a házépítési problémakörben két olyan absztrakt résztervre, ami nem egyesíthető egy teljes tervvég lépések megosztása nélkül. (Segítség: azok a pontok, ahol egy ház fizikai részei csatlakoznak, egyúttal a résztervek várható csatlakozási kölcsönhatási pontjai is.)
- 12.5. Sokan mondják, hogy a HFH-tervkészítés előnye, hogy „tegyünk egy körutat Los Angelesból New Yorkba, és vissza” típusú problémákat is meg tud oldani, amelyeket nehéz nem HFH-jelölésekkel kifejezni, hiszen a kiinduló és a cél állapotok azonosak lennének (*Off(LA)*). Tud mondani megfelelő reprezentációt, illetve meg tudja-e oldani ezt a problémát HFH-k nélkül?
- 12.6. Mutassa meg, hogy az eredeti STRIPS cselekvésleírás hogyan írható át HFH-dekompozícióvá. Használja az *Eléri(p)* cselekvést a *p* feltétel eléréséhez szükséges cselekvés jelölésére.
- 12.7. A hagyományos programozási nyelvekben számos művelet modellezhető olyan cselekvéssel, amely megváltoztatja a világ állapotát. Például egy értékadási operátor egy memóriaterület tartalmát másolja, míg egy nyomatás operátor megváltoztatja a kimenet állapotát. Egy, ezekből a műveletekből álló programot tervnek tekinthetünk, melynek célját a program specifikációja definiálja. A felkészítő algoritmusok ezért felhasználhatók egy adott specifikációjú program elkészítéséhez.  
(a) Írjon egy operátorsémát az értékadási operátorhoz (egy változó értékének átadása egy másik változónak). Vegye figyelembe, hogy az eredeti érték fejlőliródik!  
(b) Mutassa meg, hogy az objektumok létrehozása hogyan használható a felkészítő által két változó értékének egy átmeneti változó felhasználásával történő felcserélésére.
- 12.8. Tekintsük a következő indoklást: egy bizonytalan kiinduló állapotot megengedő keretrendszerbe a **diszjunktív következmények** (*disjunctive effects*) csak jelölési könnyebbéget jelentenek, és nem a kifejezőképesség növelésére szolgálnak. Egy  $P \vee Q$  diszjunktív következményt tartalmazó *a* cselekvéssémát minden helyettesíthetünk a **when**:  $R: P \wedge$  **when**  $\neg R: Q$  feltételes következményekkel, ami cserében két hagyományos cselekvésre redukálható. Az *R* kifejezés egy a kiinduló állapotban ismeretlen véletlen állítást takar, melyhez nem tartozik érzékelő cselekvés. Helyes ez az indoklás? Vegyük külön két esetet, egyet, melyben csak egy *a* cselekvésséma szerepel a tervben, és egy másikat, melyben több mint egy példány van.
- 12.9. Miért nem tudja a feltételes tervkészítés kezelni a nem korlátos nemdeterminisztikusságot?
- 12.10. A kockavilágban két STRIPS cselekvést kellett bevezetnünk, hogy az *Üres* predikátumot helyesen megtartsuk, a *Mozgat* és az *AsztalraTesz* cselekvéseket. Mutassuk meg, hogy a feltételes következmények hogyan használhatók minden eset egyetlen cselekvéssel történő leírására.

- 12.11.** A feltételes következményeket a porszívóvilág Szív cselekvésére mutattuk be, ahol az, hogy melyik kocka lesz tiszta, attól függ, hogy mely kockán áll a robot. Tud mondani olyan ítéletlogikai változókat, melyek a porszívóvilág állapotait definiálják úgy, hogy a Szív cselekvésnek *feltétel nélküli* leírása legyen? Adja meg a Szív, a *Balra* és a *Jobbra* cselekvések leírását az állításaik felhasználásával, és mutassa meg, hogy ezek kielégítők a világ összes lehetséges állapotának leírásához.
- 12.12.** Adja meg a Szív cselekvés teljes leírását a dupla-Murphy porszívó problémára, ami néha piszkot hagy maga után, amikor egy tiszta célterületre mozog, és néha piszkot rak le, ha a Szív cselekvést egy tiszta négyzeten hajtjuk végre.
- 12.13.** Keressen egy megfelelően piszkos szőnyeget, amely akadálymentes és porszívózza ki. Rajzolja le a porszívó által választott útvonalat, amilyen pontosan csak tudja. Indokolja a választ hivatkozva a fejezetben tárgyalt tervkészítési módszerekre.
- 12.14.** A következő idézetek samponos flakonok hátuljáról származnak. Azonosítsa mindegyiket, mint egy feltétel nélküli, feltételes vagy végrehajtás monitorozó tervet. (a) „Mosás. Öblítés. Ismételje.” (b) „Tegye a sámpont a fejére, és hagyja ott néhány percig. Öblítsen és ismételje, ha szükséges.” (c) „Amennyiben az irritáció nem szűnik, forduljon orvoshoz.”
- 12.15.** A 12.10. ábrán szereplő És-VAGY-GRÁF-KERESÉS algoritmus csak a gyökér és az aktuális állapot közötti útvonalat ellenőri le ismétlődő állapotokra. Mindehhez tegyük fel, hogy az algoritmus eltárolt *minden* meglátogatott állapotot, és összeheti ezzel a listával (lásd például 3.19. ábra GRÁF-KERESÉS). Határozza meg, hogy milyen információt kellene tárolni, és azt, hogy az algoritmus hogyan használja ezt fel, amikor ismétlődő állapotot talál. (Segítség: szüksége lesz arra, hogy különbözetet tegyen azon állapotok, melyekhez egy sikeres részterv készült korábban, és azon állapotok között, melyekre nem talált résztervet.) Magyarázza meg, hogyan használhatunk **címkeket (labels)**, hogy elkerüljük a résztervek duplikálását.
- 12.16.** Mutassa meg precízen, hogy hogyan módosítjuk az És-VAGY-GRÁF-KERESÉS algoritmust, hogy ciklikus tervet generáljon, amennyiben ciklusmentes terv nem létezik. Ehhez három problémával kell megküzdenie: a terv lépéseinak címkezése, hogy egy ciklikus terv vissza tudjon mutatni a terv egy korábbi pontjára, a VAGY-KERESÉS módosításával, hogy egy ciklikus terv megtalálása után ciklusmentes tervet keressen, és a terv leírásának olyan bővítésével, mely jelzi, hogy a terv ciklikus. Mutassa meg, hogyan működik az algoritmus (a) a tripla-Murphy porszívóvilágban, (b) a váltakozó dupla-Murphy porszívóvilágban. Használhat számítógépes implementációt az eredmények ellenőrzéséhez. A (b) eset tervé felírható egy hagyományos ciklusszintaxis használatával?
- 12.17.** Specifikálja teljesen a hiedelem állapot módosító folyamatot részben megfigyelhető környezetekre. Azaz a módszert, ami az új hiedelmi állapot leírást számítja

(mint ismeret állítások listáját) az aktuális hiedelmi állapot leírásból és a feltételek következményeket tartalmazó cselekvésleírásokból.

- 12.18.** Írjon a (12.2.) egyenlettel analóg cselekvésleírásokat a *Jobbra* és a *Szív* cselekvésekre. Adja meg a leírását a *PozícióEllenőrzés* cselekvésnek, a (12.3.) egyenletek megfelelően. Ismételje ezt meg a 12.11. feladat alternatív állítás halma-zának felhasználásával.
- 12.19.** Tekintse meg az 525. oldal listáját, ami felsorolja, hogy mit nem tud az újratervező ágens végrehajtani. Vázoljon fel egy algoritmust néhány probléma kezelésére.
- 12.20.** Vegyük a következő feladatot: egy páciens olyan tünetekkel érkezik az orvoshoz, amelyet kiszáradás vagy  $D$  fertőzés okozhat (de nem mindenkor). Két lehetséges cselekvés van: *Izsik*, ami feltétel nélkül kúrálja a kiszáradást, és *Gyógyszert szed*, amely a  $D$  fertőzést kúrálja ki, de káros mellékhatása van, ha a páciens kiszáradt. Írja fel a probléma leírását PDDL-ben, és az összes releváns világ számbavétele mellett ábrázolja az érzékelőmentes tervet, ami megoldja a problémát.
- 12.21.** Az előző feladat gyógyszerszedési problémájához adja hozzá a *Vizsgál* cselekvést, melynek feltételes következménye a *TenyészetNövekedés*, amennyiben a *Fertőzés* igaz, és minden esetben van egy érzékelési következménye: *Ismert(TenyészetNövekedés)*. Ábrázolja a feltételes tervet, ami megoldja a problémát és minimalizálja a *GyógyszertSzed* cselekvés felhasználását.

V. RÉSZ

**BIZONYTALAN**

**TUDÁS ÉS**

**KÖVETKEZTETÉS**

---

# 13. BIZONYTALANSÁG

*Ebben a fejezetben meglátjuk, mit kell egy ágensnek tennie, ha nem minden kristálytiszta.*

## 13.1. CSELEKVÉS BIZONYTALAN TUDÁS ESETÉN

A III. és IV. részben leírt logikai ágensek ismeretelméleti kijelentéseket tesznek az állítások igaz, hamis vagy ismeretlen voltáról. Ha egy ágens elegendő mennyiségi tényt ismer a környezetére vonatkozóan, akkor a logikai megközelítés révén olyan terveket tud származtatni, amelyek biztosítják a sikert. Ami nagyon jó. Sajnos azonban *az ágensek szinte soha nem térnek hozzá a környezetiük érintő teljes igazsághoz*. Vagyis az ágenseknek **bizonytalanság (uncertainty)** közepepte kell működniük. A 7. fejezet wumpus világának ágense például csak olyan szenzorokkal rendelkezik, amelyek lokális információt szolgáltatnak; a világ nagy része nem figyelhető meg közvetlenül. A wumpus ágens gyakran képtelen felfedezni, hogy két négyzet közül melyik tartalmaz csapdát. Ha a két négyzet az aranyhoz vezető úton van, akkor az ágensnek kockáztnia kell, és rá kell lépnie az egyikre.

A világ a valóságban nagyságrendekkel bonyolultabb a wumpus világánál. Egy logikai ágens nem biztos, hogy képes kimerítő és helyes leírást adni arról, hogy a cselekedetei mennyire lesznek eredményesek. Tegyük fel például, hogy az ágens ki szeretne vinni valakit a repülőtérré egy bizonyos járathoz. Ehhez egy  $A_{90}$  nevű tervet tekint, amely szerint a járat indulása előtt 90 perccel kell elindulnia otthonról és normális sebességgel kell vezetnie. Mindemellett, még ha a repülőtér csak 15 kilométerre van is, az ágens nem juthat a következő határozott kijelentésre: az  $A_{90}$  terv alapján időben kiérünk a repülőtérré, hanem csak egy gyengébb következtetésre: az  $A_{90}$  terv alapján időben kiérünk a repülőtérré, ha az autóm nem romlik el vagy nem fogy ki belőle az üzemanyag, továbbá, ha nem történik velünk baleset, valamint a hídon sem lesz baleset, és a gép nem száll fel korábban, továbbá ha nem lesz földrengés... E feltételek egyike sem vezethető le, így a terv sikere sem következtethető ki biztosan. Mindez a 10. fejezetben említett minősítési problémára (**qualification problem**) mutat példát.

Ha egy logikai ágens nem tud arra a következtetésre jutni, hogy valamely cselekvés-sorozat eléri a célját, akkor képtelen lesz cselekedni. A feltételes tervezés bizonyos mértékben úrrá lehet a bizonytalanságon, de csak akkor, ha az ágens érzékelési tevékenységei képesek megszerezni a szükséges információt, továbbá ha nincs túl sok különböző eshetőség. Egy másik lehetőség az lehetne, ha az ágenst egy egyszerű, de hibás elméettel ruháznánk fel, amely *biztosan* lehetővé tenné valamely terv származtatását; egy ilyen terv az esetek többségében valóban működik is, ugyanakkor problémák lépnek fel, ha az események az ágens elméletének ellentmondanak. Sőt az ágens elméletének pon-



tossága és használhatósága közötti egyensúly kezelése maga is megköveteli a bizonytalanság végiggondolását. Összegezve, egyetlen tisztán logikai ágens sem lesz képes arra a következetésre jutni, hogy az  $A_{90}$  tervet kell követnie.

Mindenkorral tegyük fel, hogy az  $A_{90}$  ténylegesen a helyes és követendő terv. De mit is értünk ezen a kijelentésen? Ahogy már a 2. fejezetben kifejtettük, ez a kijelentés azt takarja, hogy a végrehajtható összes lehetséges terv közül az  $A_{90}$  az, amelyik a környezetre vonatkozó adott információ mellett várhatóan az ágens teljesítményének legnagyobb mértékét biztosítja. A teljesítmény foka magában foglalja a járathoz időben való kiérést a repülőtérre, továbbá a repülőtéren való hosszú, értelmetlen várakozás és a gyorshajtás miatti megbírságolás elkerülését. Az ágens rendelkezésére álló információ ezen kimenetelek közül egyiket sem szavatolja az  $A_{90}$  esetén, de bizonyos mértékű hihetőséget nyújtja annak, hogy azok be fognak következni. Más tervek, mint például az  $A_{120}$ , megnövelné az ágens hitét abban, hogy időben kiérnek a repülőtérre, de egyben megnövelné a hosszú várakozás valószínűségét is. A helyes cselekedet – az ésszerű döntés (*rational decision*) – ezért figg mind a különböző célok viszonylagos fontosságától, mind pedig a megvalósulásuk valószínűségétől és mértékétől. Az alfejezet hátravezérlésében ezeket a gondolatokat szeretnénk hangsúlyozni, előkészítvén az ebben és a következő fejezetekben bemutatott bizonytalan következetés és az ésszerű döntés általános elméleteinek a kifejlesztését.



## Bizonytalan tudás kezelése

Ebben az alfejezetben a bizonytalan tudás természetét vizsgáljuk meg közelebbről. Egy egyszerű diagnosztikai példán keresztül mutatjuk be az érintett fogalmakat. A diagnosztika – legyen az orvosi, gépjármű-javítási vagy bármi egyéb – olyan feladat, amely szinte minden tartalmaz bizonytalanságot. Hogy lássuk a logikai megközelítés sikertelenségét, próbáljuk meg egy fogorvosi diagnosztikai rendszer szabályait meghatározni elsőrendű logika segítségével. Tekintsük a következő szabályokat:

$$\forall p \ Tünet(p, \text{Fogfájás}) \Rightarrow \text{Betegség}(p, \text{Lyuk})$$

A gond az, hogy ez a szabály rossz. Nincs minden fogfájós betegnek lyukas foga; néhányuknak lehet ínyisorvadása vagy tályoga, vagy valamilyen más problémája.

$$\begin{aligned} \forall p \ Tünet(p, \text{Fogfájás}) \Rightarrow \\ \text{Betegség}(p, \text{Lyuk}) \vee \text{Betegség}(p, \text{Ínyisorvadás}) \vee \text{Betegség}(p, \text{Tályog}) \dots \end{aligned}$$

Sajnos, ahhoz, hogy igazzá tegyük a szabályt, közel végiglen sok okot kellene hozzátennünk. Megpróbálhatjuk a szabályt ok-okozati szabályá formálni:

$$\forall p \ \text{Betegség}(p, \text{Lyuk}) \Rightarrow \text{Tünet}(p, \text{Fogfájás})$$

De még ez a szabály sem lesz helyes, hiszen nem minden lyukas fog okoz fájdalmat. Az egyetlen módja, hogy kijavítsuk a szabályt, az az, hogy logikailag teljessé tesszük: a bal oldalt ki kell bővíteni minden olyan lehetséges okkal, amely lyukas fogak esetén fogfájást okoz. De még ebben az esetben is figyelembe kell venni a diagnózis megállapításánál annak lehetőségét, hogy a betegnek egymástól teljesen függetlenül is lehet fogfájása és lyukas foga.

Próbálkozásaink az elsőrendű logika alkalmazására olyan területeken, mint az orvosi diagnosztika, három fő okból is kudarcot vallanak:

- **Lustaság (laziness):** túl nagy munkát jelent az ok és okozatok teljes eseményhalmazának felsorolása, amely elengedhetetlenül szükséges annak biztosítására, hogy egy szabály minden érvényes legyen. Ezenkívül túl körülmenyes az eredményül kapott hatalmas méretű szabályok használata.
- **Az elméleti tudatlanság (theoretical ignorance):** az orvosi tudományterület elmélete nem teljes.
- **A gyakorlati tudatlanság (practical ignorance):** még ha ismerjük is a teljes szabályrendszeret, egy bizonyos beteggel kapcsolatban akkor is lehetünk bizonytalanok, mert előfordulhat, hogy még nem fejeződött be, vagy egyáltalán nem végezhető el az összes szükséges vizsgálat.

Arról van szó, hogy a fogfájás és a lyuk közötti kapcsolat egyik irányban sem feltétlen logikai következmény. Ez jellemző az orvosi területekre, valamint más szakértői területekre (jog, üzlet, tervezés, autójavítás, kertészkedés, időpontok összeegyeztetése stb.). Az ágens tudása legjobb esetben is csak egy **bizonyos mértékű meggyőződést** vagy **hiedelmet (degree of belief)** nyújthat az adott kijelentésekkel kapcsolatban. A meggyőződési értékek kezelésére az elsőleges eszközünk a **valószínűség-számítás (probability theory)** lesz, amely egy 0 és 1 közötti számszerű meggyőződési mértéket rendel az egyes mondatokhoz. (A bizonytalan következetésre alkalmas néhány más lehetséges módszerrel a 14.7. alfejezetben olvashatunk.)

A **valószínűség lehetőséget nyújt a lustáságunkból vagy tudáshártyunkból fakadó bizonyatlanság kifejezésére (összegzésére)**. Nem lehetünk biztosak abban, hogy mi kínoz egy bizonyos beteget, ugyanakkor hiheitjük, hogy mondjuk 80% az esély arra – azaz 0,8 a valószínűsége annak –, hogy a betegnek van lyukas foga, ha fogfájásra panaszkodik. Azaz, várákozásaink szerint az összes, az ágens tudása alapján a jelenlegi szituációtól megtérülőzhettekben eset 80%-ban a páciensnek van lyukas foga. Ezt a valószínűséget statisztikai adatokból vonhatjuk le – az eddig vizsgált fogfájós betegek 80%-ának lyukas volt a foga –, vagy valamilyen általános szabályból, illetve tények kombinációjából származtathatjuk. A 80% azokat az eseteket foglalja magában, amikor minden olyan tényező jelen van, amely hatására a lyuk fogfájással jár, valamint amikor a betegeknek teljesen függetlenül van fogfájása és lyukas foga is. A hiányzó 20%-ban pedig a fogfájás összes többi lehetséges oka található, amelyek megerősítéséhez vagy tagadásához túl lusták vagy tudatlanok vagyunk.

Egy adott kijelentéshez rendelt 0 valószínűség annak az egyértelmű meggyőződésnek felel meg, hogy a mondat állítása hamis, míg az 1 valószínűség egyenértékű azzal a határozott meggyőződéssel, hogy a mondat állítása igaz. A 0 és 1 közötti valószínűségek a mondat igazságtartalmában való hit közbenső mértékeinek felelnek meg. Az állítás **valójában** persze vagy igaz, vagy hamis. Fontos tehát megjegyeznünk, hogy a meggyőződés mértéke és az igazságtartalom mértéke különböző fogalmak. A 0,8 valószínűség nem jelent „80%-ban igaz”-at, hanem egy 80%-os mértékű meggyőződést – vagyis egy igen erős elvárást (reményt) az állítás igazságával szemben. Következetesen, hogy a világban a tények vagy érvényesek, vagy nem. Az igazság mértéke, mint a meggyőződés mértékének az ellentéte, a **fuzzy logika (fuzzy logic)** tárgya, amelyet a 14.7. alfejezetben tárgyalunk.

A logikában egy olyan állítás, mint „a páciensnek van lyukas foga” az interpretációjától és a világtól függően vagy igaz, vagy hamis; csak akkor igaz, ha az a tény, amelyre hivatkozik, megfelel a tényállásnak. A valószínűség-számításban egy olyan mondat, mint „0,8 annak a valószínűsége, hogy a betegnek lyukas a fogai” az ágens megyőződését fejezi ki, és nem közvetlenül a valóságra vonatkozik. Ez a megyőződés az ágens addigi észleléseiitől függ. Az észlelések alkotják azt a tényt vagy tényállást (**evidence**), amelyen a valószínűségi kijelentések alapulnak. Tegyük fel például, hogy az ágens kihúz egy lapot egy megkevert kártyapakliból. Mielőtt ránéz a lapra, 1/52 valószínűséget kell rendelnie a mellé az állítás mellé, hogy a kihúzott lap a pikk ász. Megnézés után ugyanennek a valószínűsége vagy 0, vagy 1. Vagyis, egy kijelentéshez rendelt valószínűség inkább annak felel meg, hogy egy logikai állítás (vagy annak tagadása) következik-e a tudásbázisból, mint annak, hogy igaz-e vagy sem. Ahogy a tudásbázishoz hozzáadott állítások megváltoztatják az abból levonható következtetéseket, ugyanúgy a valószínűség is megváltozik, amikor több tény birtokába jutunk.<sup>1</sup>

Ebből következően minden valószínűségi kijelentésnek hivatkoznia kell azokra a tényekre, amelyek alapján az adott valószínűség az állításhoz lett rendelve. Amint egy ágens új észlelések birtokába jut, ezek figyelembevételével módosítja a valószínűségek becslését. Mielőtt tények birtokába jutunk, előzetes, illetve a *priori* (*prior*) vagy *feltétel nélküli (unconditional)* valószínűségről beszélünk, a tények birtokában pedig *utólagos*, illetve a *posteriori* (*posterior*) vagy *feltételes (conditional)* valószínűségről. Az ágens a legtöbb esetben rendelkezni fog bizonyos tényekkel az érzékelései hatására, és érdekelt lesz az általa felügyelt kimenetelek a posteriori valószínűségeinek kiszámításában.

## Bizonytalanság és racionális döntések

A bizonytalanság megjelenése gyökeresen megváltoztatja azt a módot, ahogyan az ágens a döntéseit meghozza. Egy logikai ágensnek általában van valamilyen célja, és bármely tervet végrehajt, amely biztosítja a cél elérését. Azon az alapon választ ki vagy utasít vissza egy cselekvést, hogy az eléri a célt vagy sem, függetlenül attól, hogy más cselekvések mire vezetnek. Más lesz a helyzet azonban, amint a bizonytalanság belép a képhez. Gondoljuk végig újra az  $A_{90}$  tervet a repülőtérré jutás szempontjából. Tegyük fel, hogy ez 95%-os valószínűséggel biztosítja a sikert. Ésszerű döntést jelent ez? Nem feltétlenül: lehetnek más tervezetek, például az  $A_{120}$ , amelynek nagyobb esélye van a sikerre. Ha életbevágó az, hogy ne késsük le a járatot, akkor célszerűbb megkockáztatni egy hosszabb várakozást a repülőtéren. Mi a helyzet például az  $A_{1440}$  tervvel, amely a felszállás előtti 24 órával való indulást javasolja. A legtöbb esetben ez nem jó választás, mert bár majdnem biztosra vehetjük az időben való érkezést, sajnos a tűrhetetlenül hosszú várakozást is.

Ilyen típusú döntések meghozatala előtt az ágensnek fel kell állítania egy **preferencia-sorrendet (preferences)** a különböző tervezetek lehetséges **kimenetelei (outcome)** között.

<sup>1</sup> Ez teljesen eltér attól, hogy a világ megváltozása következtében egy állítás igazzá vagy hamissá válik. Ha valószínűségeket akarunk használni egy változó világ kezelésére, akkor ugyanolyan mechanizmusokra – átlapot, távolság és események – lesz szükségünk, mint amelyet a 10. fejezetben használtunk a logikai reprezentációban. Ezeket a mechanizmusokat a 15. fejezetben tárgyaljuk.

Egy adott kimenetel teljesen határozott állapotot jelent, magában foglalva, hogy az ágens időben érkezik a repülőtére vagy sem, és hogy milyen hosszú lesz a repülőtéri várakozás. A preferenciák figyelembevételével történő lefrásra és következtetésre a **hasznosságelméletet (utility theory)** fogjuk használni. (A **hasznosság – utility** – fogalmán a továbbiakban az értendő, hogy „milyen a haszon minősége”, és nem valamifajta közhasznúság, mint az elektromos művek vagy vízművek esetében.) A hasznosságelmélet szerint minden állapotnak van egy hasznavehetőségi mértéke vagy haszná az ágens számára, és az a számára több hasznos hozó állapotokat fogja előnyben részesíteni.

Egy állapot hasznossága viszonylagos az ágens számára, akinek preferencia-sorrend-jét a hasznosságfüggvény hivatott képviseli. A 6. fejezet játszmáiban szereplő jutalmak például hasznosságfüggvények. Sakkozás közben annak az állapotnak a hasznossága, amelyben Fehér játszmát nyer, nyilvánvalóan igen magas a fehér bábukkal játszó ágens számára, ugyanakkor igen alacsony a feketével játszó számára. Vagy bizonyos játékosok (beleértve a szerzőket is) igen elégedettek lehetnek egy világbajnok elleni döntetlennel, ugyanakkor más játékosok (beleértve a korábbi világbajnokot) nem. Az ízlést vagy a személyes választást nem lehet megindokolni: azt gondolhatjuk, hogy egy ágens, aki a műfagyt előnyben részesíti a csokireszelékkel szemben különös, sőt félre van vezetve, de azt nem mondhatjuk, hogy irracionális. A hasznosságelmélet még az önzetlenséget is megengedi, egyszerűen azáltal, hogy az ágens a saját hasznosságát növelő tényezők közé beleveszi mások jólétét.

A preferenciák sorrendje, amit a hasznossággal fejezhetünk ki, az ésszerű döntések általános elméletében valószínűségekkel van kombinálva. Ezt hívjuk **döntéselméletnek (decision theory)**:

$$\text{döntéselmélet} = \text{valószínűség-elmélet} + \text{hasznosságelmélet}$$

A döntéselmélet alapgondolata szerint *egy ágens akkor és csak akkor racionális, ha olyan cselekvést választ, amely az adott cselekvés összes lehetséges kimenetelére átlagosan legmagasabb várható hasznat hozza*. Ezt hívják a **Maximális Várható Haszon (MVH, Maximum Expected Utility, MEU)** elvének. Az elv működését a 6. fejezetben láthattuk, amikor az ostáblajáték optimális lépését vizsgáltuk röviden. A későbbiekben látni fogjuk, hogy ez az elv teljesen általános érvényű.



## Egy döntéselméleti ágens tervezése

A 13.1. ábra egy, a cselekvései kiválasztásához döntéselméleti módszereket alkalmazó ágens felépítését mutatja. Egy absztrakt szinten az ágens megegyezik a 7. fejezetben leírt logikai ágens felépítésével. Az elsődleges különbség közöttük az, hogy a döntéselméleti ágens pillanatnyi állapotra vonatkozó tudása bizonytalan; az ágens **meggyőződési** vagy **hiedelemállapota (belief state)** a világ összes lehetséges aktuális állapotainak a valószínűségeit megjeleníti. Az idő előrehaladtával az ágens egyre több tényt gyűjt össze, és meggyőződési állapota is változik. Adott meggyőződési állapotra alapozva az ágens valószínűségi becsléseket tud adni az egyes cselekedetek kimeneteire vonatkozóan, következésképpen ki tudja választani a legnagyobb várható hasznossággal bíró lépést. Ez és a következő fejezet a valószínűségi információ általános megjelenítésére és az azon alapuló számításokra összpontosít. A 15. fejezet a meggyőződési állapot

```

function DE-ÁGENS(észlelés) returns egy cselekvés
  static: meggyőződési állapot, a világ állapotára vonatkozó valószínűségi meggyőződések halmaza
    cselekvés, az ágens cselekvése

  a meggyőződési állapot aktualizálása cselekvés és érzékelés alapján
  cselekvések kimeneteleihez tartozó valószínűségek kiszámítása,
    adott műveletek leírása és pillanatnyi meggyőződési állapot
  a legmagasabb várható haszonnal járó cselekvés kiválasztása
    adott kimenetekhez tartozó valószínűségek és a hasznosságra vonatkozó információ
  return cselekvés

```

**13.1. ábra.** Döntéselméleti ágens, amely racionális cselekvéseket választ ki. A lépéseket a következő fejezetben részletezzük.

reprezentálásának és aktualizálásának, valamint a környezet becslésének speciális módszereivel foglalkozik. A 16. fejezet a hasznosságelméletet taglalja mélyiségeiben, míg a 17. fejezetben a komplex döntések módszereit építjük fel.

## 13.2. VALÓSZÍNŰSÉGI ALAPFOGALMAK

Miután felállítottuk a racionális ágens általános keretét, szükségünk lesz egy formális nyelvre a bizonytalan tudás leírásához és a következtetéshez. minden olyan jelölésnek, amely a meggyőződésünk fokának leírására szolgál, képesnek kell lennie két fő dolog kezelésére: az egyik a kijelentések jellege, amelyekhez meggyőződési mértéket akarunk rendelni, a másik pedig a meggyőződés mértékének az ágens tapasztalatától való függése. A valószínűség-elmélet itt bemutatott változata az ítéletlogika egy kiterjesztését használja állításaihoz. A tapasztalattól való függés az a priori valószínűségi kijelentések és a feltételes valószínűségi állítások szintaktikai megkülönöztetésében tükröződik. Az a priori valószínűségi kijelentéseket akkor alkalmazzuk, mielőtt még tények birtokába jutnánk, míg a feltételes valószínűségi kijelentések explicit módon tartalmazzák a megszerzett tényeket.

### Állítások

A hiedelmi mértékeket mindig **állításokhoz** (**propositions**) rendeljük – amelyek ez és ez a helyzet típusú kijelentések. Az állítások leírására eddig két formális nyelvet – az ítéletlogikát és az elsőrendű logikát – használtuk. Ez az alfejezet egy olyan nyelvet ír le, amelyet a valószínűség-elmélet jellegzetesen használ, és amely valamelyest kifejezőbb az ítéletlogikánál. (A 14.6. alfejezet azokat a módszereket taglalja, amelyek megadják, hogy milyen hiedelmi mértékek tulajdoníthatók az elsőrendű logika egyes kijelentéseinek.)

A nyelv alapeleme a **valószínűségi** vagy **véletlen változó** (**random variable**), ami úgy tekinthető, mint ami egy kezdetben ismeretlen „állapotú” világ egy „részére” vonatkozik. Például a *Lyuk* a bal alsó bölcsességfogam esetleges lyukasságát mutat-

ja. A véletlen változók a kényszerkielégítési problémáknál megismert CSP-változókhöz és az ítéletlogikánál használt ítéletszimbólumhoz hasonló szerepet játszanak. A véletlen változókat mindenkor nagybetűvel kezdjük. (Ugyanakkor az ismeretlen véletlen változókat változatlanul kis- és egybetűs nevekkel fogjuk jelölni, például:  $P(a) = 1 - P(\neg a)$ .)

Minden valószínűségi változóhoz tartozik egy **értéktartomány (domain)**, amelyből az értékeket veheti. Például a *Lyuk* tartománya az *(igaz, hamis)* lehetne.<sup>2</sup> (Az értékeket kisbetűs nevekkel fogjuk jelölni.) Az állítások legegyszerűbb fajtája azt jelenti ki, hogy a valószínűségi változó valamilyen konkrét értéket vesz fel a tartományon belül. Például a *Lyuk = igaz* azt reprezentálja, hogy nem valóban lyukas a bal alsó bölcsességg fogam.

A véletlen változók – a CSP-változókhoz hasonlóan – tipikusan három csoportba sorolhatók a tartomány fajtájától függően:

- (**Boole-típusú logikai véletlen változók (Boolean random variables)**), mint a *Lyuk*, amelyeknek az *(igaz, hamis)* a tartománya. Az olyan állításokat, mint a *Lyuk = igaz* gyakran rövidítve, csak a kis kezdőbetűs nevével – *lyuk* – fogjuk jelölni, míg a *Lyuk = hamis* állítás rövidített jelölése: *\neg lyuk*.
- **Diszkrét véletlen változók (discrete random variables)**, amelyek speciális esetben logikai változók is lehetnek, egy megszámlálható tartományból vesznek fel értéket. Például az *Időjárás* tartománya a *(napos, esős, felhős, havazik)* lehet. A tartomány értékeinek egymást kizároknak és összességében kimerítőknek (teljeseknek) kell lenniük. Ha ez nem okozhat félreértest, akkor a *havazik* rövidítés fogja jelölni például az *Időjárás = havazik* állítást.
- **Folytonos véletlen változók (continuous random variables)**, amelyek valós értéket vehetnek fel. A tartomány lehet akár a teljes valós tengely, akár annak egy részhalmaza, mint a  $[0, 1]$  intervallum. Például az az állítás, hogy  $X = 4,02$  azt jelenti ki, hogy az  $X$  véletlen változó értéke pontosan 4,02. A véletlen változóra vonatkozó állítások egyenlőtlenségek is lehetnek, mint például  $X \leq 4,02$ .

Néhány kivételtől eltekintve, mi a diszkrét esetre fogunk koncentrálni.

Az összetett állítások létrehozásához az olyan elemi állítások, mint a *Lyuk = igaz* vagy a *Fogfájás = hamis*, bármely szokásos logikai kapcsolat felhasználásával kombinálhatók. Például a *Lyuk = igaz  $\wedge$  Fogfájás = hamis* egy olyan állítás, amelyhez valamilyen hihetőségi (hihetetlenségi) mértéket rendelhetünk. Ahogy az előző bevezetésben leírtuk, a fenti állítást úgy is jelölhetjük, hogy *fogszuvásodás  $\wedge$  \neg fogfájás*.

## Elemi események

Az **elemi esemény (atomic event)** jelölés hasznos a valószínűség-elmélet alapjainak megértésében. Egy elemi esemény a világ – amely tekintetében az ágens bizonytalan – állapotának egy *teljes* leírását jelenti. Úgy is tekinthetjük, mint a világot alkotó összes változóhoz való konkrét érték hozzárendelését. Például, ha a világomat csak a *Lyuk* és a

<sup>2</sup> Egyesek elvárása szerint a tartományt halmazként kellene megadni: *{igaz, hamis}*. Mi állítások  $n$ -eseként írjuk le, mivel a későbbiekben ez megkönnyíti egy rendezés hozzárendelését.

*Fogfájás* logikai változók alkotják, akkor pontosan négy különböző elemi esemény létezik; amelyek közül a  $Lyuk = \text{hamis} \wedge Fogfájás = \text{igaz}$  egy esemény.<sup>3</sup>

Az elemi eseményeknek van néhány fontos tulajdonsága:

- Az elemi események *egymást kölcsönösen kizáró* események – legfeljebb egyikük lehet igaz. Például, nem lehet egyszerre igaz a  $lyuk \wedge fogfájás$ , valamint a  $lyuk \wedge \neg fogfájás$ .
- Az összes elemi esemény halmaza kimerítő – legalább az egyiknek igaznak kell lennie. Azaz az összes elemi esemény egyesítése logikailag egyenértékű az *igaz* állítással.
- minden egyes elemi esemény maga után vonja következményként az összes állítás igazságát vagy hamisságát függetlenül attól, hogy azok egyszerűek vagy összetettek. Ez a logikai kapcsolatok szokásos szemantikájának alkalmazása révén mutatható meg (lásd 7. fejezet). Például a  $lyuk \wedge \neg fogfájás$  elemi esemény következménye a *fogszuvásodás* igaz volta és a  $lyuk \Rightarrow fogfájás$  hamissága.
- Bárminely állítás logikailag egyenértékű azon elemi események diszjunkciójával, amelyekből az állítás következik. Például a *lyuk* állítás ekvivalens a  $lyuk \wedge fogfájás$  és a  $lyuk \wedge \neg fogfájás$  elemi események egyesítésével.

A 13.4. feladat a fenti tulajdonságok bizonyítását célozza.

## A priori valószínűség

Az a állításhoz tartozó **feltétel nélküli (unconditional)** vagy **a priori valószínűség (prior probability)** azt a megyőződési mértéket jelenti, amely *bármely más információ hiányában* az állításhoz kapcsolható; jelölése  $P(a)$ . Például ha 0,1 annak az a priori valószínűsége, hogy van lyukas fogam, akkor

$$P(Lyuk = \text{igaz}) = 0,1 \text{ vagy } P(Lyuk) = 0,1\text{-et írhatunk.}$$

Fontos megjegyeznünk, hogy  $P(a)$  csak akkor használható, ha nincs semmilyen más információ a birtokunkban. Amint ismertté válik valamilyen új információ, a továbbiakban már  $a$  adott új információ mellett **feltételes valószínűségével** kell következtetnünk. A feltételes valószínűségekkel a következő alfejezet foglalkozik.

Bizonyos esetekben előfordulhat, hogy beszélni szeretnénk egy véletlen változó összes lehetséges értékének valószínűségéről. Ilyen esetekben a **P(Időjárás)** kifejezés használható, amely az időjárás minden egyes állapotához rendelt valószínűségi értékekből képzett vektort jelöli. Következésképpen, ahelyett hogy az alábbi négy egyenletet írnánk le

$$P(Időjárás = \text{napos}) = 0,7$$

$$P(Időjárás = \text{esős}) = 0,2$$

$$P(Időjárás = \text{félhős}) = 0,08$$

<sup>3</sup> A valószínűség-elmélet számos formája az elemi eseményt, más néven **mintát (sample point)** egy primítyvek tekinti. A véletlen változót pedig, mint egy függvényt definiálja, amelynek bemenete egy elemi esemény, kimeneteként pedig a megfelelő tartomány egy értékét adjja. Ez a megközelítés talán általánosabb, de ugyanakkor kevésbé intuitív.

$$P(\text{Időjárás} = \text{havazik}) = 0.02$$

elegendő egyszerűen azt írnunk, hogy

$$\mathbf{P}(\text{Időjárás}) = (0.7, 0.2, 0.08, 0.02)$$

Az ilyen kijelentés az *Időjárás* véletlen változó előzetes valószínűség-eloszlását (**probability distribution**) definiálja.

Olyan kifejezéseket is használni fogunk, mint a  $\mathbf{P}(\text{Időjárás}, \text{Lyuk})$ , hogy egy véletlen változóhalmaz összes lehetséges kombinációjának valószínűségeit jelölni tudjuk.<sup>4</sup> Ekkor a  $\mathbf{P}(\text{Időjárás}, \text{Lyuk})$  egy  $4 \times 2$ -es valószínűségi táblázatot jelent. Ez az *Időjárás* és *Lyuk* **együttes valószínűség-eloszlása** (**joint probability distribution**).

Hasznos lehet az is, ha világot leíró véletlen változók teljes halmazáról gondolkozunk. Az olyan együttes valószínűség-eloszlást, amely lefedi a teljes halmazt **teljes együttes valószínűség-eloszlásnak** (**full joint probability distribution**) nevezünk. Például, ha a világ csak a *Lyuk*, a *Fogfájás* és az *Időjárás* változókból áll, akkor a teljes együttes valószínűség-eloszlást a

$$\mathbf{P}(\text{Lyuk}, \text{Fogfájás}, \text{Időjárás})$$

adja meg. Ez az együttes valószínűség-eloszlás egy 16 elemű,  $2 \times 2 \times 4$ -es táblázattal reprezentálható. A teljes együttes valószínűség-eloszlás minden **egyes elemi esemény valószínűségét**, és így a kérdéses világgal kapcsolatos összes bizonytalanságot **meghatározza**. A 13.4. alfejezetben látni fogjuk, hogy a teljes együttes valószínűség-eloszlás alapján bármely valószínűségi kérdés megválaszolható.

Folytonos változók esetén az eloszlás nem foglalható össze táblázatos formában, mivel a lehetséges értékek száma végtelen. Ehelyett annak valószínűsége, hogy egy valószínűségi változó egy adott  $x$  értéket vesz fel, általában  $x$  egy paraméterezett függvényeként definiálható. Például az  $X$  véletlen változó jelölje a holnapi hőmérséklet maximumát Berkeleyben. Ezzel a

$$P(X = x) = U[18, 26](x)$$

kijelentés azt a hiedelmet fejezi ki, hogy  $X$  egyenletes eloszlást mutat 18 és 26 °C között. (Néhány hasznos folytonos valószínűségi változó definícióját az A) függeléken találjuk meg.) A folytonos valószínűségi változókra vonatkozó valószínűségi eloszlást **valószínűség-sűrűségfüggvénynek** (**probability density function**) nevezünk. A sűrűségfüggvények jelentése különbözik a diszkrét eloszlásokétól. Például a korábbiakban megadott hőmérsékleteloszlásból kiindulva  $P(X = 20,5) = U[18, 26](20,5) = 0,125/\text{°C}$  adódik. Ez *nem* azt jelenti, hogy annak az esélye, hogy a holnapi maximális hőmérséklet pontosan 20,5 °C lesz 12,5%; ennek a valószínűsége természetesen 0. Technikailag ez azt jelenti, hogy annak a valószínűsége, hogy a kérdéses hőmérséklet a 20,5 °C egy kicsiny környezetébe fog esni, határértékét tekintve egyenlő azzal, hogy a 0,125-öt elosztjuk a szakasz °C-ban megadott szélességevel:

$$\lim_{dx \rightarrow 0} P(20,5 \leq dx \leq X \leq 20,5 + dx)/dx = 0,125/\text{°C}$$

<sup>4</sup> Az általános jelölési szabály szerint az eloszlás a nagybetűs változók összes értékét tartalmazza. Azaz. a  $\mathbf{P}(\text{Időjárás}, \text{lyuk})$  a valószínűségek egy olyan négyelemű vektorát jelenti, amelyben minden egyes időjárás-állapot valószínűsége szerepel *Lyuk = igaz* mellett.

Néhány szerző a diszkrét eloszlások és a sűrűségfüggvények jelölésére más szimbólumot használ; mi  $P$ -vel fogjuk jelölni mindenketőt, mivel ritkán lehet ezeket összekeverni, és az egyenletek általában azonos formájúak. Jegyezzük meg ugyanakkor, hogy még a valószínűségek mértékegység nélküli számok, a sűrűségfüggvényeknek van mértékegysége, a fenti példában  $1/^\circ\text{C}$ .

## Feltételes valószínűség

Amint az ágens bizonyos tények birtokába jut a korábban ismeretlen, a tartományra jellemző véletlen változóra vonatkozóan, az a priori valószínűségek többé nem használhatók. Ehelyett a **feltételes (conditional)** vagy a **posteriori (posterior)** valószínűségeket használhatjuk. Jelölése  $P(a|b)$ , ahol  $a$  és  $b$  tetszőleges állítás lehet.<sup>5</sup> Értelmezése:  $a$  valószínűsége, ha  $b$ -t és csak  $b$ -t tudjuk. Például,

$$P(\text{Lyuk}|Fogfájás) = 0,8$$

azt jelenti, hogy ha egy betegnél megfigyeltük, hogy fogfájása van, és semmilyen más információnk nincs vele kapcsolatban, akkor annak a valószínűsége, hogy szuvas a foga 0,8. Egy  $P(\text{lyuk})$  típusú a priori valószínűség tekinthető a  $P(\text{lyuk} |)$  feltételes valószínűség speciális esetének, ahol a feltételezett „semmi bizonyíték” jelenti.

A feltételes valószínűségek megadhatók feltétel nélküliek segítségével. A definíció

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \quad (13.1)$$

minden  $P(b) > 0$  esetén igaz. Az egyenletet írhatjuk

$$P(a \wedge b) = P(a|b)P(b)$$

alakban is, amelyet **szorzatszabálynak (product rule)** hívunk. A szorzatszabályt talán könnyebb megjegyezni: ez abból következik, hogy  $a$  és  $b$  együttes teljesüléséhez, szükséges, hogy  $b$  igaz legyen, valamint hogy  $a$  is igaz legyen  $b$  feltétele mellett. A szabályt megadhatjuk fordítva is:

$$P(a \wedge b) = P(b|a)P(a)$$

Bizonyos esetekben könnyebb a konjunkciók feltétel nélküli (a priori) valószínűségeit használni, azonban mi az esetek többségében feltételes valószínűségeket fogunk alkalmazni valószínűségi következtetéseink eszközöként.

A  $P$  jelölést használhatjuk feltételes eloszlásokra is.  $P(X|Y)$  a  $P(X = x_i|Y = y_j)$  értékeit adja meg minden lehetséges  $i$ -re,  $j$ -re. Annak példájaként, hogy ez mennyivel tömörebbé teszi a jelölést, képzeljük el a szorzatszabály alkalmazását minden olyan esetre, ahol az  $a$  és  $b$  állítások  $X$  és  $Y$  bizonyos értékeit veszik fel. A következő egyenleteket fogjuk kapni:

$$P(X = x_1 \wedge Y = y_1) = P(X = x_1|Y = y_1) = P(Y = y_1)$$

$$P(X = x_1 \wedge Y = y_2) = P(X = x_1|Y = y_2) = P(Y = y_2)$$

$$\vdots$$

<sup>5</sup> A „|” operátor a lehető legkevesebb előzményt jelenti, azaz a  $P(a \wedge b|c \vee d) P((a \wedge b)|(c \vee d))$ -vel egyenértékű.

Mindezt összefoglalhatjuk egyetlen egyenletben is:

$$\mathbf{P}(X, Y) = \mathbf{P}(X|Y)\mathbf{P}(Y)$$

Ne felejtsük, hogy ez egy olyan egyenlethalmaz jelölésére szolgál, amely kapcsolatba hozza a táblázatok megfelelő elemeit, és *nem* a táblázatok mátrix szorzásáról van szó. Csábító, de helytelen a feltételes valószínűségeket bizonytalansággal kiegészített logikai implikációkkal tekinteni. Például a  $P(a|b) = 0,8$  állítást *nem lehet* úgy értelmezni, hogy amikor csak igaz  $b$ ,  $P(a)$  0,8-del egyenlő. Ez két okból is téves: először is  $P(a)$  minden  $a$  előzetes valószínűségét, és nem a valamelyen tény megléte esetén alkalmazandó utólagos valószínűségét jelöli; másrészt pedig a  $P(a|b) = 0,8$  csak akkor alkalmazható, ha  $b$  az *egyetlen* tény, amelynek birtokában vagyunk. Ha ismerünk egy további  $c$  információt is, akkor a hihetőségi mértékét  $P(a|b \wedge c)$  fogja adni, amely akár független is lehet  $P(a|b)$ -től. Például  $c$  megadhatja közvetlenül azt is, hogy  $a$  igaz vagy hamis. Ha megvizsgálunk egy beteget, aki fogfájásra panaszcodik, és találunk egy lyukas fogat, akkor a további, *lyuk* információnak jutottunk birtokába, és így természetesen, arra a következetésre fogunk jutni, hogy  $P(\text{lyuk}|\text{fogfájás} \wedge \text{lyuk}) = 1,0$ .

### 13.3. VALÓSZÍNŰSÉGI AXIÓMÁK

Az eddigiekben az állításokra, az a priori, valamint a feltételes valószínűségi kijelentésekre vonatkozó szintaxist definiáltunk. A továbbiakban meg kell fogalmaznunk a valószínűségi kijelentések valamelyen szemantikáját is. Ezt azokkal az alapvető axiómákkal kezdjük, amelyek a valószínűségi skálát és annak végpontjait határozzák meg:

1. minden valószínűség 0 és 1 közé esik. Bármely  $a$  kijelentésre

$$0 \leq P(a) \leq 1$$

2. A biztosan igaz (azaz érvényes) állítások valószínűsége 1, a biztosan hamis (azaz kielégíthetetlen) állításoké pedig 0.

$$P(\text{igaz}) = 1 \quad P(\text{hamis}) = 0$$

Ezután egy olyan axiómára van szükségünk, amely a logikailag összefüggő állításokat kapcsolja össze. A legegyszerűbb, ha a diszjunkcióra vonatkozó valószínűséget az alábbiak szerint definiáljuk:

3. A diszjunkció valószínűsége

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Ez utóbbi szabályt könnyű megjegyezni arról, hogy azok az esetek, ahol  $a$  igaz és azok az esetek, ahol  $b$  igaz, együtteremtésben lefedik az eseteket, ahol  $a \vee b$  igaz; ugyanakkor a két halmazba eső eseteket összeadva kétszer számoljuk azokat, amelyek a halmazok közös részébe esnek, így ki kell vonnunk  $P(a \wedge b)$ -t.

Ezt a három axiómát gyakran nevezik Kolmogorov-axiómánaknak (**Kolmogorov's axioms**) az orosz matematikus Andrei Kolmogorov emlékére, aki nevéhez fűződik az ezen alapaxiómákból kiinduló valószínűség-elmélet felépítése. Vegyük észre, hogy az axiómák csak az a priori valószínűségekkel, és nem a feltételes valószínűségekkel foglalkoznak; ennek oka, hogy az utóbbiakat már a korábbiakban definiáltuk a (13.1) egyenlet szerint.

## Honnan származnak a valószínűségek?

Vég nélküli viták folytak a valószínűségi számok forrása és értelme körül. A frekvencionista (**frequentist**) álláspont szerint ezek az értékek csak kísérletekből származhatnak: ha 100 ember vizsgálata során az derül ki, hogy 10-nek szuvas a foga, akkor azt mondhatjuk, hogy a szuvas fog valószínűsége körülbelül 0,1. E szemlélet szerint az az állítás, miszerint „a lyuk valószínűsége 0,1” azt jelenti, hogy ha végtelen sok mintát megvizsgálnánk, határértékét tekintve 0,1 lenne annak a résznek a hánnya, ahol ezt tapasztalnánk. Tetszőleges, de véges számú minta esetén pedig megbecsülhetjük a valós arányt, valamint kiszámíthatjuk azt is, hogy várhatóan makkora lesz a becslésünk pontossága.

Az objektivista (**objectivist**) megközelítés szerint a valószínűségek inkább az univerzum valóságos jellemzőinek tekinthetők – ahol egy jellemző a dolgok hajlama bizonyos típusú viselkedésre –, mint egy megfigyelő meggyőződési foka leírásának. Például az, hogy egy nem hamis pénzdarab feldobásánál 0,5-es valószínűséggel fej jön ki, magának a pénzdarabnak a hajlandósága. E szemlélet szerint a frekvencionista mérések ezen hajlandóságok megfigyelésére tett kísérletek. A legtöbb fizikus egyetért abban, hogy a kvantumjelenségek valójában valószínűségek, de makroszkopikus szinten – például a pénzfeldobásnál – a bizonytalanság a kezdeti feltételek elhanyagolása miatt lép fel, ami látszólag nem konzisztens a hajlandóságalapú szemlélettel.

A szubjektivista (**subjectivist**) szemlélet szerint a valószínűségek inkább az ágens meggyőződésének jellemzésére szolgálnak, és nincs külső fizikai jelentőségük. E szerint az orvos vagy az elemző aszerint találhatja ki a számokat, hogy mondhatta véleményem szerint a lyuk valószínűsége 0,1 körül várható. Kifejlesztettek olyan további, még megbízhatóbb módszereket is – mint például az 557–558. oldalon részletezett fogadási rendszereket –, amelyekkel az emberek valószínűséghozzárendelését szeretik volna kiismerni. A vonatkoztatási osztály (reference class) probléma jól illusztrálja a szubjektivitás jelenlétéét. Képzeljünk el egy frekvencionista orvost, aki meg akarja állapítani annak az esélyeit, hogy egy páciens egy bizonyos betegségen szenved. Ehhez az orvos más, de fontos szempontokból (kor, tünet, esetleg nem tekintetében) azonos betegeket akar figyelembe venni, majd megnézni közöttük az ilyen betegségeken szenvedők arányát. Azonban, ha az orvos minden figyelembe vesz, amit ismer a beteggel kapcsolatban – a grammra pontos testsúlyát, a haja színét, az anyja leánykori nevét stb. – arra az eredményre fog jutni, hogy nincs még egy olyan ember, aki pontosan megegyezne a beteggel, és így nincs olyan vonatkoztatási osztály sem, amelyről kísérleti eredményeket lehetne gyűjteni. Mindez nyugtalantító kérdése a tudományfilozófiának.

Laplace közömbösségi elve (**principle of indifference**) (Laplace, 1816) szerint a tény tekintetében szintaktikailag „szimmetrikus” állításokhoz azonos valószínűséget kell rendelni. Erre vonatkozóan történtek különböző finomítási javaslatok, amelyek Carnap és más filozófusok azon törekvéseiben csúcsosodtak ki, hogy egy olyan precíz induktív logikát (**inductive logic**) fejlesszenek ki, amely képes bármely állítás helyes valószínűségét tetszőleges megfigyelések alapján meghatározni. Jelenlegi felfogásunk szerint nem létezik különálló induktív logika; inkább úgy gondoljuk, hogy minden ilyen logika egy szubjektív a priori valószínűségi eloszláson alapszik, amelynek hatása a megfigyelések számának növekedésével csökken.

## A valószínűségi axiómák használata

A valószínűségi axiómákból nagyon sok hasznos tény vezethető le. Például a tagadásra vonatkozó jól ismert szabály következik, ha  $\neg a$ -val helyettesítjük  $b$ -t a harmadik axiómában:

$$\begin{aligned} P(a \vee \neg a) &= P(a) + P(\neg a) - P(a \wedge \neg a) && (\text{a 3. axiómából, } b = \neg a \text{ helyettesítéssel}) \\ P(\text{igaz}) &= P(a) + P(\neg a) - P(\text{hamis}) && (\text{logikai ekvivalencia alapján}) \\ 1 &= P(a) + P(\neg a) && (\text{a 2. axiómából}) \\ P(\neg a) &= 1 - P(a) && (\text{algebrai műveletekkel}) \end{aligned}$$

A következtetés harmadik sora önmagában is hasznos tény és kiterjeszhető a Boole-típusú logikai esetből az általános diszkrét esetre. Legyen  $\langle d_1, \dots, d_n \rangle$  a  $D$  diszkrét változó értelmezési tartománya. Könnyű megmutatni (13.2. feladat), hogy

$$\sum_{i=1}^n P(D = d_i) = 1$$

Azaz, egyetlen változó tetszőleges valószínűségi eloszlása összegzésének 1-et kell adnia.<sup>6</sup> Igaz az is, hogy változók tetszőleges halmazának bármely *együttes* valószínűségi eloszlását összegezve, eredményül szintén 1-et kapunk: ennek belátására elegendő egyetlen többdimenziós, az eredeti változók értelmezési tartományai Descartes-szorzataként létrehozott tartomány felett értelmezett változót létrehoznunk.

Emlékezzünk, hogy bármely  $a$  állítás egyenértékű minden olyan elemi esemény diszjunkciójával, ahol  $a$  igaz; nevezük az események ezen halmazát  $e(a)$ -nak. Emlékezzünk arra is, hogy az elemi események egymást kölcsönösen kizáják, így a 2. axióma szerint az ilyen események konjunkciója nulla. Következésképpen a 3. axiómából a következő egyszerű összefüggésre juthatunk: *Egy állítás valószínűsége megegyezik azon elemi események valószínűségeinek összegével, ahol az állítás igaz, azaz*

$$P(a) = \sum_{e_i \in e(a)} P(e_i) \quad (13.2)$$

Ez az egyenlet egyszerű módszert kínál bármilyen állítás valószínűségének meghatározására, ha adott az összes elemi esemény valószínűségét meghatározó teljes együttes valószínűség-eloszlás (lásd 13.4. alfejezetet). A következő alfejezetekben a valószínűségek kezelésének további szabályait fogjuk származtatni. Mindenekelőtt azonban az axiómák megalapozottságát vizsgáljuk meg.

## Ami a valószínűségi axiómákat indokolja

A valószínűségi axiómák tekinthetők az ágens által elhihető valószínűségi hiedelmek korlátainak. Ez valamennyire hasonlít a logikai esethez, amikor például egy logikai ágens nem hihet egyszerre  $A$ -ban,  $B$ -ben és  $\neg(A \wedge B)$ -ben. De van itt egy további bonyodalom is. A logikai esetben az együttes bekövetkezés definíciójának jelentése szerint a fenti állítások közül legalább egynek *hamisnak kell lennie a valóságban*, vagyis egy ágens számára ésszerűtlen mindenből hinni. A valószínűségeket tekintve azonban a kijelentések nem közvetlenül a valóságra vonatkoznak, hanem az ágens saját ismereteire. Akkor miért nem hihet egy ágens a meggyőződések következő – a harmadik axiómát egyértelműen megsértő valószínűség-hozzárendelések – halmazában?

$$\begin{aligned} P(a) &= 0,4 & P(a \wedge b) &= 0,0 \\ P(b) &= 0,3 & P(a \wedge b) &= 0,8 \end{aligned} \quad (13.3)$$

Az effajta kérdés évtizedekig vita tárgya volt azok között, akik a valószínűségek használatát tartották egyedül jogosnak a meggyőződési mértékek kifejezésére, és azok köztől, akik más megközelítési módokat javasoltak. Itt most egy olyan érvelést mutatunk

<sup>6</sup> Folytonos változónál az összegzést integrálal kell helyettesíteni:  $\int_{-\infty}^{\infty} P(X = x) dx = 1$

be a valószínűségi axiómákkal kapcsolatban, amelyet először Bruno de Finetti fejtett ki 1931-ben.

De Finetti érvelésének kulcsa a meggyőződési fok és a cselekvések közötti összefüggés. A gondolat lényege, hogy ha egy ágensnek van valamilyen szintű meggyőződése egy  $a$  állítással kapcsolatban, akkor képesnek kell lennie olyan téteket megadni, amelyek mellett mindegy, hogy az  $a$  állításra, vagy ellene fogad. Képzeljük el úgy, mint egy játszmát két ágens között: az 1. ágens szerint az én hiedelmem az  $a$  eseményben 0,4. A 2. ágens ezután szabadon dönthet, hogy  $a$  mellett vagy ellen fogad akkora téttel, amely a nyilatkozott hiedelemfokkal konzisztens. Vagyis a 2. ágens választhatja azt, hogy arra fogad, hogy  $a$  esemény bekövetkezik, 4 dollárt téve az 1. ágens 6 dollárjával szemben, vagy tehet 6 dollárt az 1. ágens 4 dollárja ellenében arra fogadva, hogy  $a$  nem fog bekövetkezni.<sup>7</sup> Ha egy ágens meggyőződésében nem tükröződik pontosan a világ ismerete, akkor számíthatunk arra, hogy hosszú távon veszíteni fog a szemben álló ágenssel szemben, akinek a hiedelmei pontosabb képet mutatnak a világ állapotáról.

 De Finetti ennél sokkal erősebb állítást is bizonyított: *ha az 1. ágens olyan meggyőződési mértékekben hisz, amelyek megsértik a valószínűség-számítás axiómáit, akkor létezik olyan stratégia a 2. ágens számára, amely biztosítja, hogy az 1. ágens minden alkalommal veszíteni fog.* Tehát ha elfogadjuk, hogy egy ágensnek a valószínűségeknek megfelelően kell felennie a pénzét, akkor azt is el kell fogadnunk, hogy ésszerűtlen olyan meggyőződésekkel vallani, amelyek megsértik a valószínűségi axiómákat.

Azt gondolhatjuk, hogy ez a fogadósi nagyon körmönfont. Mi van például, ha valaki megtagadja, hogy fogadjon? Elrontja-e ez az egész érvelést? Minderre az a válasz, hogy a fogadásos játék elvont modellje annak a döntési helyzetnek, amelyben minden ágens minden pillanatban *elkerülhetetlenül* részt vesz. minden cselekvés (beleértve a tétlenséget is) egyfajta fogadás, és a következmény tekinthető a fogadás nyereményének. Épp úgy nem utasíthatjuk vissza a fogadást, mint ahogy nem állíthatjuk meg az idő műlását.

Nem mutatjuk be de Finetti elméletének bizonyítását, de mutatunk rá egy példát. *Tegyük fel, hogy az 1. ágens meggyőződése megfelel a (13.3) egyenletnek.* A 13.2. ábra azt mutatja, hogy ha a 2. ágens azt választja, hogy 4 dollárt tesz  $a$ -ra, 3 dollárt  $b$ -re és 2 dollárt  $\neg(a \vee b)$ -re, akkor az 1. ágens minden veszíteni fog, függetlenül attól, hogy  $a$  vagy  $b$  fog bekövetkezni.

Állítás	1. ágens Meggyőződés	2. ágens		Kimenetel az 1. ágens számára			
		Fogadás	Tét	$a \wedge b$	$a \wedge \neg b$	$\neg a \wedge b$	$\neg a \wedge \neg b$
$a$	0,4	$a$	4–6 között	-6	-6	4	4
$b$	0,3	$b$	3–7 között	-7	3	-7	3
$a \vee b$	0,8	$\neg(a \vee b)$	2–8 között	2	2	2	-8
				-11	-1	-1	-1

**13.2. ábra.** Mivel az 1. ágens meggyőződése nem következetes, ezért a 2. ágens tud úgy téteket tenni, amely  $a$  és  $b$  kimenetelétől függetlenül biztosítja, hogy az 1. ágens veszítsen

<sup>7</sup> Érvelhetünk azzal, hogy az ágens döntését a bankviszonyok fogják meghatározni, azaz 1 dollár elvesztése nincs kiegyenlítőzve 1 dollár azonos valószínűség mellett való megnyerésével. Kellően kicsire választha azonban a téteket, ez a probléma elkerülhető. Savage (Savage, 1954) analízise az egész problémát elkerüli.

Más, erősen filozofikus érveléseket is előterjesztettek a valószínűségek használata mellett, amelyek közül a legnevezetesebb Cox és Carnap nevéhez fűződik (Cox, 1946; Carnap, 1950). Lévén a világ olyan, amilyen, a gyakorlati példák sokszor minden bizonyítéknál erősebbek. Így a valószínűség-számításon alapuló következető rendszerek sikere minden érvelésnél hatékonyabb volt a hitetlenek megtérítésében. A következőkben azt vizsgáljuk meg, hogy az axiómák hogyan használhatók a következetésben.

## 13.4. TELJES EGYÜTTES VALÓSZÍNŰSÉG-ELOSZLÁSON ALAPULÓ KÖVETKEZTETÉS

Ebben az alfejezetben a **valószínűségi következetés** (probabilistic inference) egy egyszerű módszerét fogjuk leírni – hogy hogyan határozhatók meg az állítások a posteriori valószínűségekre vonatkozó megfigyelt bizonyítékok alapján. „Tudásbázisként”, a teljes együttes valószínűség-eloszlást fogjuk használni, amelyből az összes kérédésre adandó válasz levezethető. Menet közben számos hasznos, a valószínűségeket tartalmazó egyenletek kezelésére alkalmas módszert is bevezetünk.

Kezdjük egy egészen egyszerű példával, egy olyan tartománnyal, amely minden összes három logikai változóból áll. Ezek: *Fogfájás*, *Lyuk*, *Beakadás* (a fogorvos kellemetlen acélszondája beleakad a fogamba). A teljes együttes eloszlás a 13.3. ábra szerinti  $2 \times 2 \times 2$ -es táblázatból fog állni.

	<i>fogfájás</i>		$\neg$ <i>fogfájás</i>	
	<i>beakadás</i>	$\neg$ <i>beakadás</i>	<i>beakadás</i>	$\neg$ <i>beakadás</i>
<i>lyuk</i>	0,108	0,012	0,072	0,008
$\neg$ <i>lyuk</i>	0,016	0,064	0,144	0,576

13.3. ábra. A *Fogfájás*, *Lyuk*, *Beakadás* világ egy teljes együttes valószínűség-eloszlása

Vegyük észre, hogy az együttes valószínűség-eloszlásban – a valószínűségi axiómáknak megfelelően – a valószínűségek összege 1. Hasonlóképpen, a (13.2) egyenlet közvetlen módöt nyújt bármely, egyszerű vagy összetett állítás valószínűségének a meghatározására: egyszerűen azokat az elemi eseményeket kell meghatároznunk, amelyekben az állítás igaz, majd összegeznünk kell a hozzájuk rendelt valószínűségeket. Például, hat olyan elemi esemény van, amelyben a *lyuk*  $\vee$  *fogfájás* igaz:

$$\begin{aligned} P(\text{lyuk} \vee \text{fogfájás}) &= 0,108 + 0,012 + 0,072 + 0,008 \\ &\quad + 0,016 + 0,064 = 0,28 \end{aligned}$$

Az egyik általános feladat a változók egy részhalmaza vagy egyetlen változó fölötti valószínűségi eloszlás kifejezése. Például az első sor bejegyzéseinek összege a *lyuk* feltétel nélküli vagy peremeloszlását<sup>8</sup> (**marginal probability**) adja:

$$P(\text{lyuk}) = 0,108 + 0,012 + 0,072 + 0,008 = 0,2$$

<sup>8</sup> Azért hívják így, mert a biztosítási matematikusok/statisztikusok általános gyakorlata szerint a megfigyelt gyakoriságokat a biztosítási táblázatok szélére (margójára) szokták írni.

Ezt a folyamatot **marginalizálásnak** (**marginalization**) vagy **kiátlagolásnak** (**summing out**) hívjuk, mivel a *Lyuk*-on kívüli változókat „kiátlagoljuk”. A következő általános behatárolási szabályt fogalmazhatjuk meg a változók tetszőleges **Y** és **Z** halmaza esetén:

$$P(Y) = \sum_z P(Y|z)P(z) \quad (13.4)$$

Azaz, egy **Y** feletti eloszlás megkapható, ha az összes többi változót kiátlagoljuk az **Y**-t tartalmazó együttes eloszlásokból a szorzat szabályban

$$P(Y) = \sum_z P(Y|z)P(z) \quad (13.5)$$

Ezt a szabályt **feltételefeloldásnak** (**conditioning**) hívjuk. A marginalizálás és a feltételelfeloldás szabályai előnyöseknek fognak bizonyulni mindenfajta valószínűségi kifejezéseket tartalmazó következetésben.

A legtöbb esetben valamely változó, bizonyos másokra vonatkozó tények esetén fennálló, **feltételes** valószínűségének kiszámítása fog bennünket érdekelni. A feltételes valószínűségek meghatározásához, először is alkalmazzuk a (13.1) egyenletet, hogy egy feltétel nélküli valószínűségen alapuló kifejezésre jussunk, majd ezen kifejezés értékét meghatározzuk a teljes együttes eloszlásból. Például a *lyuk* valószínűségét a fogfájás tény fennállása esetére az alábbiak szerint határozhatjuk meg:

$$P(\text{lyuk}|fogfájás) = \frac{P(\text{lyuk} \wedge \text{fogfájás})}{P(\text{fogfájás})} = \frac{0,108 + 0,012}{0,108 + 0,012 + 0,016 + 0,064} = 0,6$$

Ellenőrzésképpen kiszámíthatjuk annak valószínűségét, hogy nincs *lyuk*, de tudjuk, hogy fogfájás igen:

$$P(\neg\text{lyuk}|fogfájás) = \frac{P(\neg\text{lyuk} \wedge \text{fogfájás})}{P(\text{fogfájás})} = \frac{0,016 + 0,064}{0,108 + 0,012 + 0,016 + 0,064} = 0,4$$

Vegyük észre, hogy a fenti két számításban az  $1/P(\text{fogfájás})$  konstans értékű marad függetlenül attól, hogy a *Lyuk* mely értékét számítjuk. Valójában ez egy, a  $P(\text{Lyuk}|fogfájás)$ -t **normalizálás** (**normalization**) konstansnak tekinthető, amely biztosítja, hogy a feltételes valószínűségek összege 1 lesz. A valószínűségekkel foglalkozó fejezetekben az ilyen konstansokat  $\alpha$ -val fogjuk jelölni. E jelölés segítségével a két előző kifejezés egybefoglalható:

$$\begin{aligned} P(\text{Lyuk}|fogfájás) &= \alpha P(\text{Lyuk}, \text{fogfájás}) \\ &= \alpha [P(\text{Lyuk}, \text{fogfájás}, \text{beakadás}) + P(\text{Lyuk}, \text{fogfájás}, \neg\text{beakadás})] \\ &= \alpha (\langle 0,108, 0,016 \rangle + \langle 0,012, 0,064 \rangle) = \alpha \langle 0,12, 0,08 \rangle = \langle 0,6, 0,4 \rangle \end{aligned}$$

A normalizálás sok valószínűségi számításnál hasznos rövidítésnek bizonyul.

A példából levezethető egy általános következetési eljárás. Ragaszkodjunk ahhoz az esethez, ahol a keresés egy változót érint. A következő jelölésekre lesz még szükségünk: jelölje **X** a keresés változóját (a példában ez a *Lyuk*), **E** jelentse a tény változók halmazát (a példában a *Fogfájás* az egyetlen ilyen), **e** ezek megfigyelt értékét mutatja és a többi, meg nem figyelt változót **Y** foglalja magában (a példa esetében csak a *Beakadás* tartozik ide).  $P(X|e)$ -t keressük, és a következőképpen számíthatjuk ki:

$$\mathbf{P}(X|e) = \alpha \mathbf{P}(X, e) = \alpha \sum_y \mathbf{P}(X, e, y) \quad (13.6)$$

ahol az összegzést az összes lehetséges  $y$  fölött végezzük (azaz az  $Y$  meg nem figyelt változók értékeinek összes lehetséges kombinációja esetén). Vegyük észre, hogy az  $X$ ,  $E$  és  $Y$  változó együttesen a változók teljes halmazát alkotja az adott tárgytartományban, így  $\mathbf{P}(X, e, y)$  egyszerűen a teljes valószínűségi eloszlás egy valószínűségi részhalmaza. Az algoritmust a 13.4. ábra mutatja.

```

function FELSOROL-EGYÜTTES-KÉRDEZÉS( $X, e, P$ ) returns egy  $X$  feletti valószínűségi eloszlás
  inputs:  $X$ , a lekérdezés változója
     $e$ , az  $E$ -ben levő változók megfigyelt értékei
     $P$ , az  $\{X\} \cup E \cup Y$  változók együttes eloszlása /*  $Y = \text{rejtett változók}$  */
   $Q(X) \leftarrow X$  fölötti valószínűség-eloszlás, kezdetben üres
  for each  $X x_i$  értékére do
     $Q(x_i) \leftarrow \text{FELSOROL-EGYÜTTES}(x_i, e, Y, [], P)$ 
  return NORMALIZÁL( $Q(X)$ )

function FELSOROL-EGYÜTTES( $x, e, \text{változók}, \text{értékek}, P$ ) returns egy valós szám
  if ÜREŠ?(változók) then return  $P(x, e, \text{értékek})$ 
   $Y \leftarrow \text{ELSÖ}(változók)$ 
  return  $\sum_y \text{FELSOROL-EGYÜTTES}(x, e, \text{MARADÉK}(változók), [y|\text{értékek}], P)$ 

```

**13.4. ábra.** A valószínűségi következetés egy olyan algoritmusá, amely valamely teljes együttes valószínűségi eloszlás bejegyzéseit veszi számba

Az eljárás ciklusa minden  $X$ , minden  $Y$  összes értéke fölött fut, számba véve minden olyan elemi eseményt, amely  $e$  rögzített értéke mellett lehetséges, az együttes valószínűségi tábla alapján összegzi ezek valószínűségeit, és normalizálja az eredményeket.

Ha adott a teljes együttes valószínűségi eloszlás, akkor a FELSOROL-EGYÜTTES-KÉRDEZÉS egy teljes algoritmus, amely alkalmas minden diszkrét változóra vonatkozó valószínűségi lekérdezés megválaszolására. Azonban nem minősíthető jónak: egy  $n$  db Boole-típusú logikai változó által leírt tartományban  $O(2^n)$  méretű bemeneti táblára van szüksége, és a tábla feldolgozása is  $O(2^n)$  időt igényel. Valós problémák esetén nem csupán három, hanem több száz vagy több ezer véletlen változó figyelembevételére van szükség. Nagyon hamar teljesen kivitelezhetetlenné válik a szükséges hatalmas számú valószínűség definiálása – nem létezik ugyanis olyan tapasztalat, amellyel a táblázat egyes bejegyzései külön-külön becslőhetők lennének.

Ezért a táblázatos formában megadott teljes együttes valószínűségi eloszlás a gyakorlatban nem használható eszköz következetető rendszerek felépítésére (mindemellett a fejezet végén található történeti megjegyzésekben találkozhatunk egy ilyen módszeren alapuló valós alkalmazással). E technikát úgy kell tekinteniünk, mint egy elméleti alapot, amelyre a hatékonyabb módszerek épülhetnek. A fejezet további részeiben bemutatjuk azokat az alapötleteket, amelyekre szükségünk lesz a 14. fejezet valós rendszereinek kifejlesztéséhez.

### 13.5. FÜGGETLENSÉG

Bővítsük ki a 13.3. ábrán látható teljes együttes valószínűségi eloszlást egy negyedik, *Időjárás* változóval. Ezzel a teljes együttes valószínűségi eloszlás  $P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}, \text{Időjárás})$  lesz 32 bejegyzéssel (mivel az *Időjárás* változónak négy értéke lehet). A teljes eloszlás a 13.3. ábrán mutatott táblázat négy „példányát” – minden egyes időjárastípusra egyet – fogja magában foglalni. Természetes kérdésnek tűnik, hogy vajon ezeknek a táblázatpéldányoknak mi az egymáshoz és az eredeti háromváltozós táblához való viszonyuk? Hogyan függ össze például a  $P(\text{fogfájás}, \text{beakadás}, \text{lyuk}, \text{Időjárás} = \text{felhős})$  a  $P(\text{fogfájás}, \text{beakadás}, \text{lyuk})$ -kal?

A megválaszolás egyik lehetséges módja a szorzatszabály alkalmazása:

$$\begin{aligned} P(\text{fogfájás}, \text{beakadás}, \text{lyuk}, \text{Időjárás} = \text{felhős}) \\ = P(\text{Időjárás} = \text{felhős} | \text{fogfájás}, \text{beakadás}, \text{lyuk})P(\text{fogfájás}, \text{beakadás}, \text{lyuk}) \end{aligned}$$

Amennyiben eltekintünk az isteni hatalommal bíró páciensektől, nem gondolhatjuk komolyan, hogy valakinek a fogászati problémái befolyásolják az időjárást. Ezért észszerűnek tűnik a következő kijelentés:

$$P(\text{Időjárás} = \text{felhős} | \text{fogfájás}, \text{beakadás}, \text{lyuk}) = P(\text{Időjárás} = \text{felhős}) \quad (13.7)$$

Ebből a következőre következtethetünk:

$$P(\text{fogfájás}, \text{beakadás}, \text{lyuk}, \text{Időjárás} = \text{felhős}) = P(\text{Időjárás} = \text{felhős})P(\text{fogfájás}, \text{beakadás}, \text{lyuk})$$

Hasonló egyenlet írható fel a  $P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}, \text{Időjárás})$  eloszlás minden egyes bejegyzéséhez. Az általános egyenletet pedig a következőképpen írhatjuk fel:

$$\begin{aligned} P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}, \text{Időjárás}) \\ = P(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk})P(\text{Időjárás}) \end{aligned}$$

Ebből következően a négyváltozós, 32 elemes táblázat létrehozható egy 8 és egy 4 elemes táblázatból. A dekompozíció sematikus vázlatát a 13.5. (a) ábra mutatja.

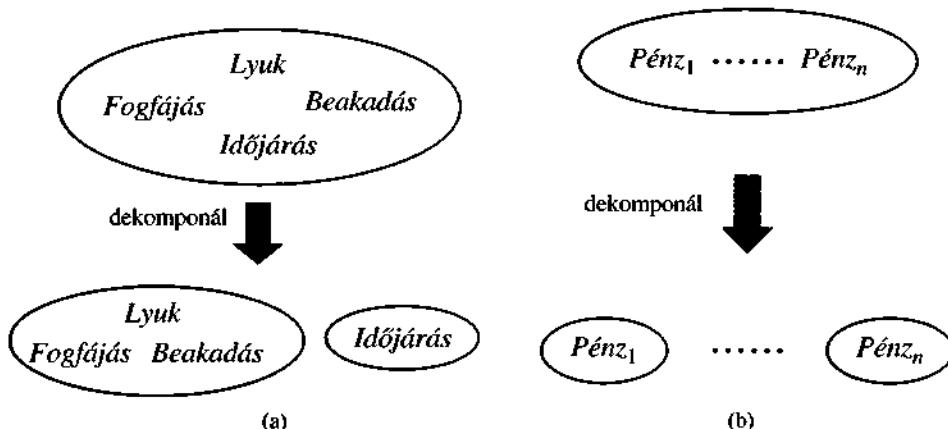
Azt a tulajdonságot, ami alapján a (13.7) egyenletet felírtuk függetlenségnek (más szóval **marginális függetlenségnek** vagy **abszolút függetlenségnek**) (**independence**, **marginal independence**, **absolute independence**) nevezzük. Esetünkben az időjárás független bárkinek a fogászati problémáitól. Az  $a$  és  $b$  állítások függetlensége feliratot:

$$P(a|b) = P(a) \text{ vagy } P(b|a) = P(b) \text{ vagy } P(a \wedge b) = P(a)P(b) \quad (13.8)$$

A fenti képletek egymással ekvivalensek (13.7. feladat). Az  $X$  és  $Y$  változók közötti függetlenség a következőképpen fejezhető ki (még egyszer felhívjuk a figyelmet, hogy ezek ekvivalensek):

$$P(X|Y) = P(X) \text{ vagy } P(Y|X) = P(Y) \text{ vagy } P(X, Y) = P(X)P(Y)$$

A függetlenségi állítások általában a tartománnal kapcsolatos ismereteken alapulnak. Mint korábban láthattuk, ezek jelentősen lecsökkenthetik a teljes együttes eloszlás meghatározásához szükséges információ mennyiségét. Ha a változók teljes halmaza szétbontható független részhalmazokra, akkor a teljes együttes eloszlás felbontható



13.5. ábra. Két példa nagyméretű együttes eloszlások kisebb eloszlásokra való felbontására az abszolút függetlenség alapján. (a) Az időjárás és a fogászati problémák függetlenek. (b) A pénzfeldobások függetlenek.

ezben részhalmazok felett értelmezett, egymástól független együttes eloszlásokra. Például  $n$  független pénzfeldobás eredményének együttes eloszlása  $P(C_1, \dots, C_n)$  felírható  $n$  egy változós  $P(C_i)$  eloszlás szorzataként. Még kézzelfoghatóbban: a fogászat és a meteorológia függetlensége jó dolog, különben a fogászati gyakorlat mély meteorológiai ismereteket igényelne, és fordítva.

Amennyiben rendelkezésre állnak, akkor a függetlenségi állítások segíthetnek lecsökkenteni a tartományleírások méretét, valamint a következtetési feladat bonyolultságát. Sajnos azonban a változók teljes halmazait csak nagyon ritkán lehet függetlenség alapján egyértelműen szétválasztani. A függetlenség nem lesz igaz az olyan esetekben, amikor bármilyen, akár indirekt kapcsolat is, de fennáll két változó között. Ezen kívül, a független részhalmazok mérete is lehet egészen nagy – például a fogászat problémákkor egymással összefüggő betegségek tücatjait és tünetek százait takarhatja. Az ilyen típusú problémák kezelésére a függetlenség közvetlen kimondásánál kifinomultabb módszerekre lesz szükségünk.

## 13.6. A BAYES-TÉTEL ÉS HASZNÁLATA

Az 554. oldalon definiáltuk a szorzatszabályt (**product rule**), és rámutattunk, hogy ez a konjunkció kommutativitása miatt két alakban írható:

$$P(a \wedge b) = P(a|b)P(b)$$

$$P(a \wedge b) = P(b|a)P(a)$$

A jobb oldalak egyenlőségéből  $P(a)$ -val való osztás után következik, hogy

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \quad (13.9)$$

Ezt az egyenletet **Bayes-szabályként (Bayes' rule)** ismerjük (nevezik Bayes-törvénynek vagy Bayes-tételnek is).<sup>9</sup> Ez az egyszerű egyenlet az alapja az összes korszerű valószínűségi következtetést alkalmazó MI-rendszernek. A többérettékű változókat tartalmazó általánosabb eset a P jelölésekkel a következő átírással adható meg:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

amelyet ismételten úgy kell értelmezni, hogy egy egyenlethalmazt képvisel, amely egyenletek mindegyike a változók meghatározott értékeire vonatkozik. Lesznek olyan esetek, amikor egy még általánosabb felirást használhatunk, valamilyen e háttérben feltételével:

$$P(Y|X, e) = \frac{P(X|Y, e)P(Y|e)}{P(X|e)} \quad (13.10)$$

## Bayes tételenek alkalmazása: egyszerű eset

A Bayes-tétel első pillantásra nem tűnik túl használhatónak. Egyetlen feltételes valószínűség kiszámításához három kifejezés – egy feltételes és két feltétel nélküli valószínűség – megadása szükséges.

A gyakorlatban a Bayes-tétel jól használható, mivel gyakran rendelkezünk a fenti három kifejezésre vonatkozó jó valószínűségi becsléssel, miközben a negyediket kell kiszámítanunk. Olyan feladatoknál, mint az orvosi diagnosztika, gyakran ismerjük az ok-okozati kapcsolatok feltételes valószínűségeit, miközben egy diagnózist szeretnénk felállítani. Az orvos tudja azt, hogy az agyhártyagyulladás az esetek mondjuk 50%-ában nyakmerevedést okoz a betegeknél. Az orvos ezenfelül ismer néhány feltétel nélküli tényt is: annak előzetes valószínűsége, hogy egy beteg agyhártyagyulladást kap, 1/50 000, míg annak előzetes valószínűsége, hogy egy betegnek merev a nyaka 1/20. Jelölje  $s$  azt az állítást, hogy a betegnek megmerevedett a nyaka, valamint  $m$  azt az állítást, hogy a betegnek agyhártyagyulladása van. Ekkor

$$P(s|m) = 0,5$$

$$P(m) = 1/50000$$

$$P(s) = 1/20$$

$$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{0,5 \times 1/50000}{1/20} = 0,0002$$

Vagyis a nyakmerevedésről panaszkodó 5000 beteg közül várhatóan csak egynek lesz agyhártyagyulladása. Vegyük észre, hogy annak ellenére, hogy az agyhártyagyulladásnak igen gyakori tünete (0,5 valószínűséggel) a nyakmerevedés, annak valószínűsége, hogy egy nyakmerevedéses betegnek ténylegesen agyhártyagyulladása van, mégis

<sup>9</sup> Az angol irodalomban Bayes' rule az elnevezése. Strunk és White *The Elements of Style* c. könyvének 1. oldalán található 1. szabály értelmében a Bayes' helyett inkább a Bayes's lenne a helyes jelölés, azonban az előbbi elterjedtebben használják. (Szerencsére ilyen probléma a magyarban nincs. A szerk.)

csekély. Ez abból következik, hogy a nyakmerevedés a priori valószínűsége sokkal nagyobb, mint az agyhártyagyulladásé.

A 13.4. alfejezetben bemutattunk egy eljárást, amely segítségével elkerülhető a tény valószínűségének (példánkban  $P(s)$ ) megbecsülése, úgy, hogy a lekérdezett változó minden egyes értékéhez (itt  $m$  és  $\neg m$ ) egy utólagos valószínűséget számítunk ki, majd az eredményeket normalizáljuk. Hasonló eljárás alkalmazható, ha a Bayes-tételt használjuk. Ismert, hogy

$$\mathbf{P}(M|s) = \alpha(P(s|m)P(m), P(s|\neg m)P(\neg m))$$

Következésképpen ahhoz, hogy ezt a megközelítést használni tudjuk,  $P(s)$  helyett  $P(s|\neg m)$ -et kell tudnunk becsülni. Nincs ingyenebb – van, hogy ez könnyebb és van, hogy nehezebb. A normalizált Bayes-tétel általános alakja

$$\mathbf{P}(Y|X) = \alpha \mathbf{P}(X|Y) \mathbf{P}(Y) \quad (13.11)$$

ahol  $\alpha$  a normalizáló konstanst jelöli, amely segítségével a  $\mathbf{P}(Y|X)$  elemeinek összegét 1-gyé tudjuk tenni.

A Bayes-téttel kapcsolatban magától értetődő kérdés, hogy miért ismerhetjük a feltételes valószínűséget az egyik irányból, a másik irányból pedig nem. Az agyhártyagyulladás tartományban a doktor tudhatja, hogy a nyakmerevedésből csak minden 5000-dik esetben következik agyhártyagyulladás, vagyis a doktor a tünetektől a kivállató ok felé, azaz **diagnosztikai** (**diagnostic**) irányban rendelkezik mennyiségi információval. Egy ilyen orvosnak nincs szüksége a Bayes-tétel alkalmazására. Sajnálatos módon azonban a *diagnosztikai tudás gyakran sokkal törékenyebbnél bizonval az okokozati összefüggések*knél. Ha például hirtelen agyhártyagyulladás-járvány tör ki, akkor az agyhártyagyulladás előzetes valószínűsége,  $P(m)$  megnő. Az az orvos, aki a járványt megelőző statisztikai adatok alapján számította ki  $P(m|s)$ -t, nem fogja tudni, hogyan változtassa meg az agyhártyagyulladásra vonatkozó értéket, míg az a doktor, aki a másik három érték segítségével számítja  $P(m|s)$ -t, látni fogja, hogy ennek értéke  $P(m)$ -mel arányosan meg fog nőni. Még fontosabb azonban, hogy a  $P(s|m)$  okozati információ értékét a járvány *nem befolyásolja*, hiszen ez kizárolag az agyhártyagyulladás lefolyásától függ. Ez a fajta közvetlen ok-okozati és modellalapú tudás alkalmazása teszi a valószínűségi rendszereket döntően robusztussá, amely a valódi világban való felhasználhatósághoz szükséges.



## A Bayes-tétel alkalmazása: több együttes tény figyelembevétele

Láttuk, hogy a Bayes-tétel hasznos lehet az egyetlen tény – például nyakmerevedés – feltételezése mellett valószínűségi kérdések megválaszolásánál. Nevezetesen, megmutattuk, hogy a valószínűségi információ gyakran  $P(okozat|ok)$  formában áll rendelkezésre. Mi történik azonban akkor, ha kettő vagy több tény van a birtokunkban? Például milyen következtetésre juthat a fogorvos, ha az az undok acélszondája lyukra akad a beteg fájó fogában? Ha ismerjük a teljes együttes eloszlást (lásd 13.3. ábra), a válasz kiolvasható:

$$\mathbf{P}(Lyuk|fogfájás \wedge breakadás) = \alpha(0,108, 0,016) \approx (0,871, 0,129)$$

Azt is tudjuk ugyanakkor, hogy ez a megközelítés nagyszámú változó esetén nem használható.

Megpróbálkozhatunk a Bayes-tétel alkalmazásával is, átfogalmazva a kérdést:

$$P(Lyuk|fogfájás \wedge beakadás) = \alpha P(fogfájás \wedge beakadás|Lyuk)P(Lyuk) \quad (13.12)$$

Ahhoz, hogy ez az átfogalmazás működjön, a *Lyuk* minden értékére ismernünk kell a *fogfájás*  $\wedge$  *beakadás* együttes bekövetkezésének feltételes valószínűségét. Noha ez két tényváltozó esetén használható lehet, nagyszámú változó esetén már nem alkalmazható. Ha  $n$  figyelembe veendő tényváltozónk van (röntgen, diéta, szájhigiénia stb.), akkor a megfigyelt értékek lehetséges kombinációinak száma  $2^n$ , amely esetek mindegyikénél ismernünk kell a feltételes valószínűséget. Ennyi erővel akár vissza is térhetünk a teljes együttes valószínűség-eloszlás használatához. Ez vezetett arra, hogy a kutatók a valószínűségszámítás helyett közelítő módszereket kezdték el használni több tény együttes figyelembetételénél, mert bár az így kapott válaszok pontatlanok, de kevesebb számolást igényelnek.

A fenti út követése helyett inkább további állításokat kell keresnünk a tárgytartományról, amelyek lehetővé teszik a kifejezések egyszerűsítését. A 13.5. alfejezetben bevezetett **függetlenség (independence)** fogalma kínálja a megoldás kulcsát, azonban finomítást igényel. Kellemes lenne, ha a *Fogfájás* és a *Beakadás* függetlenek lennének, de nem azok: ha a szonda megakad a fogban, akkor az valószínűleg lyukas, ami várhatóan fájdalmat okoz. Mindemellett, ezek a változók *függetlenek*, amennyiben a *lyuk megléte vagy hiánya adott tény*. Mindkettő a lyuk közvetlen következménye, azonban egyiknek sincs közvetlen hatása a másikra: a fogfájás a fogidegek állapotától függ, míg a szonda pontosságát a fogorvos szakértelme határozza meg, amelyben nincs szerepe a fogfájásnak.<sup>10</sup> Ezt a tulajdonságot matematikailag a következőképpen írhatjuk le

$$P(fogfájás \wedge beakadás|Lyuk) = P(fogfájás|Lyuk)P(beakadás|Lyuk) \quad (13.13)$$

Ez az egyenlet a *Lyuk* ténye esetén a *fogfájás* és *beakadás* között fennálló **feltételes függetlenséget (conditional independence)** fejezi ki. (13.13)-at (13.12)-be behelyettesítve megkapjuk a lyuk valószínűségét:

$$P(Lyuk|fogfájás \wedge beakadás) = \alpha P(fogfájás|Lyuk)P(beakadás|Lyuk)P(Lyuk)$$

Ezzel ugyanarra az információigényre jutottunk, mint a minden egyes tényt külön használó következetesnél: a keresés változójának  $P(Lyuk)$  a priori valószínűségét, valamint minden egyes okozat saját okának fennállása esetén igaz feltételes valószínűségét kell ismernünk.

Két változó,  $X$  és  $Y$  egy adott  $Z$  harmadik melletti feltételes függetlenségének általános definíciójáta következő egyenlet adja:

$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$

A fogorvostartományban, például, logikusnak tűnik a *Fogfájás* és *Beakadás* között feltételes függetlenséget feltételezni a *Lyuk* ténye esetén:

$$P(Fogfájás, Beakadás|Lyuk) = P(Fogfájás|Lyuk)P(Beakadás|Lyuk) \quad (13.14)$$

Vegyük észre, hogy ez a kijelentés valamelyest erősebb, mint amit a (13.13) egyenlet fejez ki. Ez utóbbi csak a *Fogfájás* és *Beakadás* bizonyos értékeihez rendel független-

<sup>10</sup> Feltételezve, hogy a beteg és a fogorvos két különböző személy.

séget. Éppúgy, mint a (13.8) egyenletben az abszolút függetlenségnél, a következő ekvivalens kifejezések is használhatók:

$$\mathbf{P}(X|Y,Z)=\mathbf{P}(X|Z) \text{ és } \mathbf{P}(Y|X,Z)=\mathbf{P}(Y|Z)$$

A 13.5. alfejezetben megmutattuk, hogy a teljes függetlenség bizonyíthatósága esetén a teljes együttes valószínűségi eloszlás szétbontható sokkal kisebb részekre. Ugyanez lesz igaz feltételes függetlenség esetén is. Példának okáért, a (13.14) szerinti kijelentés fennállása esetén, a felbontás a következőt jelentheti:

$$\begin{aligned} & \mathbf{P}(\text{Fogfájás}, \text{Beakadás}, \text{Lyuk}) \\ &= \mathbf{P}(\text{Fogfájás}, \text{Beakadás} | \text{Lyuk}) \mathbf{P}(\text{Lyuk}) \quad (\text{szorzatszabály}) \\ &= \mathbf{P}(\text{Fogfájás} | \text{Lyuk}) \mathbf{P}(\text{Beakadás} | \text{Lyuk}) \mathbf{P}(\text{Lyuk}) \quad (13.14)\text{-et felhasználva} \end{aligned}$$

Ezzel a módszerrel az eredeti nagy táblázat felbontható három kisebbre. Az eredeti táblázat hét független számot tartalmaz ( $2^3 - 1$ , mivel a számok összegének 1-et kell adnia). A kisebb táblázatokban öt független szám található (mindegyik feltételes valószínűségi eloszlásra  $2 \times (2^1 - 1)$ , valamint  $2^1 - 1$  a Lyuk előzetes valószínűségi eloszlására). Ez nem tűnik túl nagy győzelemnek, azonban, ha a Lyuk ténye mellett  $n$  tünet bizonyul függetlennek, akkor a reprezentáció  $O(n)$  nagyságrendben növekszik  $O(2^n)$  helyett. Következésképpen a feltételes függetlenségi kijelentések lehetővé teszik nagy valószínűségi rendszerek kezelhetőségét, sőt, a feltételes függetlenségi kijelentések gyakrabban rendelkezésre is állnak, mint az abszolút függetlenségre vonatkozók. A Lyuk fogalmilag **szétválasztja (separate)** a Fogfájs-t és a Beakadás-t, mivel minden kettőnek közvetlen következménye. Nagy valószínűségi tartományok feltételes függetlenségen keresztül lazán kapcsolódó részhalmazokra történő szétbontása a modern MI történetének legnagyobb eredményei közé tartozik.

A fogorvosi eset jó példa az olyan, rendszeresen bekövetkező mintára, ahol egyetlen ok közvetlenül befolyásol számos olyan okozatot, amelyek az ok fennállása esetén feltételesen függetlenek. A teljes valószínűségi eloszlás az alábbiak szerint írható fel:

$$\mathbf{P}(\text{Ok}, \text{Okozat}_1, \dots, \text{Okozat}_n) = \mathbf{P}(\text{Ok}) \prod_i \mathbf{P}(\text{Okozat}_i | \text{Ok})$$

Az ilyen valószínűségi eloszlásokat **naiv Bayes-modellnek** hívjuk – „naiv”, mert gyakran használják (egyszerűsítésként) olyan esetekben is, ahol az „okozati” változók valójában *nem* függetlenek az „ok” fennállása esetén. (A naiv Bayes-modellt néha **Bayes-osztályozónak (Bayes classifier)** is hívják némi leg meggondolatlanul, ami arra sarkallta az igazi Bayes-következtetést alkalmazókat, hogy a naiv Bayes-modellt **együgyű Bayes- (idiot Bayes)** modellnek hívják.) A naiv Bayes-modellek a gyakorlatban akkor is meglepően jól működnek, ha a valószínűségi feltételezés nem igaz. A 20. fejezetben olyan módszereket írnunk le, amelyekkel a naiv Bayes-eloszlások megtanulhatók a megfigyelésekkel.

## 13.7. A WUMPUS VILÁG ÚJRALÁTOGATÁSA

E fejezet gondolatai közül sok eredményesen kombinálható a wumpus világ valószínűségi következtetési problémáinak megoldására. (A wumpus világ teljes leírása a 7.

fejezetben található.) A wumpus világban fellépő bizonytalanságot az ágens érzékelésének a világra vonatkozóan a csak helyi információt szolgáltató részlegessége okozza. Például a 13.6. ábra egy olyan szituációt ábrázol, amelyben mindenhol elérhető négyzet – [1, 3], [2, 2] és [3, 1] – tartalmazhat csapdát. Egyszerű logikai következtetéssel nem dönthető el, hogy melyik a legvalószínűbben biztonságos négyzet, így a logikai ágens kénytelen véletlenszerűen választani. Látni fogjuk ugyanakkor, hogy egy valószínűségi ágens sokkal sikeresebb, mint egy logikai ágens.

A célunk az lesz, hogy minden négyzetre kiszámítsuk a csapda valószínűségét. (A példa erejéig elfelejtjük a wumpust és az aranyat.) A wumpus világ idevágó tulajdonságai a következők: (1) egy csapda szellőt okoz a szomszédos négyzetekben, és (2) az [1,1]-en kívül minden négyzet 0,2 valószínűsggel tartalmaz csapdát. Első lépésként a számításhoz szükséges véletlen változókat kell meghatározunk:

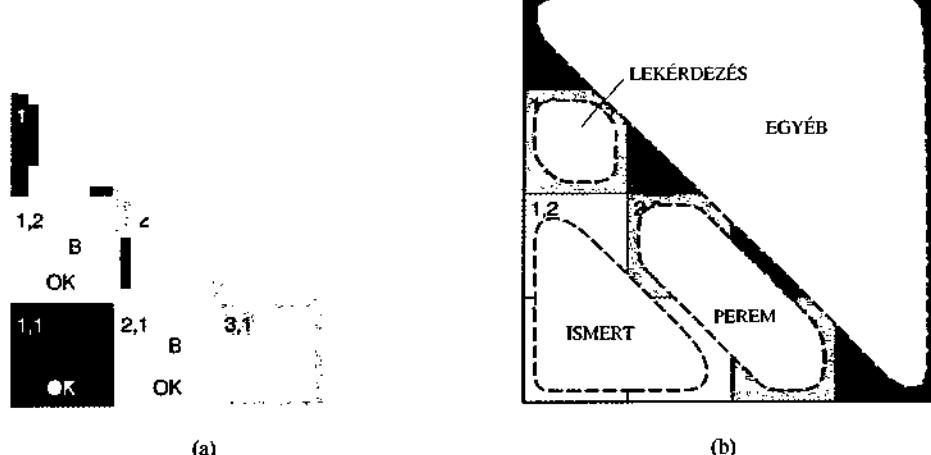
- Az ítéletlogikához hasonlóan, itt is minden négyzethez egy  $(C_{i,j})$  logikai változóra van szükségünk, amelynek értéke akkor és csak akkor igaz, ha az  $[i,j]$  négyzet tartalmaz csapdát.
- Vannak továbbá  $S_{i,j}$  logikai változóink is, amelyek igaz értéke egyértelműen az  $[i,j]$  négyzetben tapasztalható szellőre utal; példánkban csak a megfigyelt – esetünkben az [1, 3], [2, 2] és [3, 1] négyzetekre – vonatkozó változókat használjuk.

A következő lépés a  $\mathbf{P}(C_{1,1}, \dots, C_{4,4}, S_{1,1}, S_{1,2}, S_{2,1})$  teljes valószínűségi eloszlás meghatározása. A szorzatszabály alkalmazásával

$$\mathbf{P}(C_{1,1}, \dots, C_{4,4}, S_{1,1}, S_{1,2}, S_{2,1}) = \mathbf{P}(S_{1,1}, S_{1,2}, S_{2,1} | C_{1,1}, \dots, C_{4,4}) \mathbf{P}(C_{1,1}, \dots, C_{4,4})$$

adódik.

A dekompozíció nagyon könnyűvé teszi az együttes valószínűségi értékek meghatározását. Az első rész a szellő feltételes valószínűség konfigurációját adja adott csapda konfigurációinál; ennek értéke 1, ahol a szellő szomszédos egy csapdával, minden



13.6. ábra. (a) Az ágens beragad, miután [1, 2]-ben és [2, 1]-ben is szellő észlel – nincs feltátható biztonságos hely. (b) A négyzetek szétosztása *Ismert*, *Perem* és *Egyéb* csoportokba az [1, 3]-ra vonatkozó lekérdezéshez.

más esetben 0. A második tag a csapdakonfiguráció előzetes valószínűsége. A négyzetek a többi négyzettől függetlenül 0,2 valószínűséggel tartalmaznak csapdát; következésképpen

$$P(C_{1,1}, \dots, C_{4,4}) = \prod_{i,j=1,1}^{4,4} P(C_{i,j}) \quad (13.15)$$

Egy  $n$  csapdát tartalmazó konfigurációjánál ennek értéke  $0,2^n \times 0,8^{16-n}$ .

A 13.6. (a) ábra szerinti helyzetnél a bizonyíték magában foglalja minden meglátogatott négyzetre az ott megfigyelt szellőt (vagy annak hiányát) azzal a tényel együtt, hogy a meglátogatott négyzetek nem tartalmaznak csapdát. Ezeket az  $s = \neg s_{1,1} \wedge s_{1,2} \wedge s_{2,1}$  és  $ismeret = \neg c_{1,1} \wedge \neg c_{1,2} \wedge \neg c_{2,1}$  rövidítésekkel fogjuk jelölni. A mi számunkra a  $P(C_{1,3}|ismeret, s)$  típusú lekérdezések megválasztása érdekes: adott megfigyelések mellett mennyire valószínű, hogy az [1, 3] csapdát tartalmaz?

A válaszhoz követhetjük a (13.6) egyenlet szerinti szabványos megközelítést, amelyet a FELSOROL-EGYÜTTES-KÉRDEZÉS eljárás valósít meg, azaz a teljes együttes valószínűségi eloszlás bejegyzései szerinti összegzést. Jelölje *Ismeretlen* azt az összetett változót, amelyet a nem *Ismert* csoportba tartozó négyzetek és a lekérdezett [1, 3] négyzet  $C_{i,j}$  változói alkotnak. Ezzel a (13.6) egyenletből

$$P(C_{1,3}|ismeret, s) = \alpha \sum_{ismeretlen} P(C_{1,3}, ismeretlen, ismeret, s)$$

A teljes együttes valószínűségeket korábban már meghatároztuk, így készen is vagyunk – kivéve, ha a számítási bonyolultság kérdésével is törődünk. 12 ismeretlen négyzetünk van; így az összegzés  $2^{12} = 4096$  tagból fog állni. Általánosságban, az összegzés exponenciálisan nő a négyzetek számával.

Az intuíció azt súgja, hogy itt valamit még figyelmen kívül hagyunk. És tényleg, bárki megkérdezheti, hogy a többi négyzet nem lényegtelen-e? A [4, 4] tartalma nincs hatással arra, hogy az [1, 3] tartalmaz-e csapdát! Ez az intuíció valóban helyes. Jelölje *Perem* azokat a – lekérdezés változójától különböző – változókat, amelyek a meglátogatott négyzetekkel szomszédosak, esetünkben a [2, 2] és a [3, 1]. Továbbá *Egyéb* legyen a többi ismeretlen négyzet változója; példánkban – ahogy a 13.6. (b) ábra is mutatja – 10 ilyen négyzet van. A kulcs meglátás az, hogy megfigyelt szellők feltételesen függetlenek a többi változótól, ha adottak az *ismeret*, a *perem* és a lekérdezés változók. A többi, ahogy mondják, már csak egy kis algebra.

A fenti meglátás használatához a lekérdezést olyan alakra hozzuk, amelyben a szellő az összes többi változó feltétele mellett van megadva, majd az egészet egyszerűsítjük a feltételes függetlenség alapján:

$$\begin{aligned} P(C_{1,3}|ismeret, s) &= \alpha \sum_{ismeretlen} P(s|C_{1,3}, ismeret, ismeretlen) P(C_{1,3}, ismeret, ismeretlen) \\ &\quad (\text{szorzószabály}) \\ &= \alpha \sum_{perem \text{ egyéb}} \sum_{egyéb} P(s|ismeret, C_{1,3}, perem, egyéb) P(C_{1,3}, ismeret, perem, egyéb) \\ &= \alpha \sum_{perem \text{ egyéb}} \sum_{egyéb} P(s|ismeret, C_{1,3}, perem) P(C_{1,3}, ismeret, perem, egyéb) \end{aligned}$$

ahol az utolsó lépés használja ki a feltételes függetlenséget. E kifejezés első tagja nem függ a többi változótól, így az összegzés belülre vihető:

$$\mathbf{P}(C_{1,3}|ismert, s) = \alpha \sum_{perem} \mathbf{P}(s|ismert, C_{1,3}, perem) \sum_{egyéb} \mathbf{P}(C_{1,3}, ismert, perem, egyéb)$$

Függetlenség esetén a (13.15) egyenlethez hasonlóan az előző kifejezés tényezőkre bontható, majd a tényezők sorrendje átrendezhető:

$$\begin{aligned} \mathbf{P}(C_{1,3}|ismert, s) &= \alpha \sum_{perem} \mathbf{P}(s|ismert, C_{1,3}, perem) \sum_{egyéb} \mathbf{P}(C_{1,3}) C(ismert) C(perem) C(egyéb) \\ &= \alpha C(ismert) \mathbf{P}(C_{1,3}) \sum_{perem} \mathbf{P}(s|ismert, C_{1,3}, perem) C(perem) \sum_{egyéb} C(egyéb) = \\ &= \alpha' \mathbf{P}(C_{1,3}) \sum_{perem} \mathbf{P}(s|ismert, C_{1,3}, perem) C(perem) \end{aligned}$$

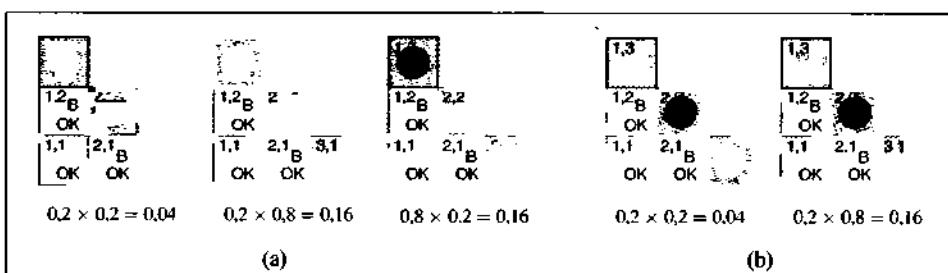
ahol az utolsó lépében  $C(ismert)$ -et belefoglaltuk a normalizáló konstansba, és kihaszítottuk, hogy  $\sum_{egyéb} C(egyéb)$  értéke 1.

*egyéb*

Most már csak négy tag van a  $C_{2,2}$  és  $C_{3,1}$  peremváltozók feletti összegzésben. A függetlenség és feltételes függetlenség miatt egyáltalán nem szükséges figyelembe vennünk a többi négyzetet. Vegyük észre, hogy a  $\mathbf{P}(s|ismert, C_{1,3}, perem)$  kifejezés értéke 1, ha a határ megegyezik a szellő érzékelésekkel és minden más esetben 0. Azaz,  $C_{1,3}$  minden értékére azon határváltozók logikai modelljei fölött végezzük az összegést, amelyek az megfelelnek az ismert tényeknek. (Érdemes ezt összehasonlítani a 7.5. ábra modelljei szerinti számítással.) A modelleket és a hozzájuk kapcsolódó  $C(perem)$  a priori valószínűségeket a 13.7. ábra mutatja. Ezzel

$$\mathbf{P}(C_{1,3}|ismert, s) = \alpha' \langle 0,2(0,04 + 0,16), 0,8(0,04 + 0,16) \rangle \approx \langle 0,31, 0,69 \rangle$$

Azaz, az [1, 3] (és a szimmetria miatt a [3, 1]) nagyjából 31% valószínűséggel tartalmaz csapdát. Hasonló számításokból (amelyet az olvasó könnyen elvégezhet) a [2, 2]-re



13.7. ábra.  $C_{2,2}$  és  $C_{3,1}$  peremváltozók, az egyes modellek  $C(perem)$  értékét mutató konzisztens modelljei: (a) három, két vagy három csapdát jelző modell  $C_{1,3} = igaz$  mellett, és (b) két, egy vagy két csapdát jelző modell  $C_{1,3} = hamis$  mellett.

86%-os valószínűséggel adódik csapda. A wumpus ágensnek határozottan el kell kerülnie a [2, 2]-t!

Ebben az alfejezetben azt mutattuk meg, hogy még a látszólag bonyolult problémák is pontosan megfogalmazhatók valószínűség-elméleti alapokon, és meg is oldhatók egyszerű algoritmusok segítségével. Hogy *hatékony* megoldásokat kapjunk, a szükséges összegzéseket a függetlenségekre és feltételes függetlenségekre alapozva egyszerűsítjük. Ezek az összefüggések gyakran egybeesnek a probléma részproblémáakra való felbontására vonatkozó természetes felfogásunkkal. A következő fejezetben ilyen összefüggések formális megjelenítésére alkalmazunk módszereket, valamint olyan algoritmusokat fejlesztünk ki, amelyek hatékonyan használhatók e reprezentációk alapján elvégzett valószínűségi következtetések végrehajtására.

## 13.8. ÖSSZEFOGLALÁS

Ez a fejezet arra mutat rá, hogy a valószínűség megfelelő módja a bizonytalanságra vonatkozó következtetéseknek.

- Bizonytalanság következik mind a lustaságból, mind pedig a tudatlanságból. Összetett, dinamikus vagy hozzáérhetetlen világok esetén minden elkerülhetetlen.
- A bizonytalanság azt jelenti, hogy a továbbiakban sok olyan egyszerűsítés, amely deduktív következtetés során megengedhető, hamis eredményre vezet.
- A bizonytalanság az ágens képtelenségét fejezi ki arra, hogy egyértelmű döntést hozzon egy állítás igazságát illetően. A valószínűségek az ágens megyőződésének fokát fejezik ki.
- Az alapvető valószínűségi állítások magukban foglalják az egyszerű és az összetett állításokra vonatkozó **a priori** és **feltételes valószínűségeket (prior and conditional probabilities)**.
- A **teljes együttes valószínűség-eloszlás (full joint probability distribution)** valószínűséget rendel a véletlen változó minden teljes körű érték-hozzárendeléséhez. A teljes együttes valószínűség-eloszlás általában sokkal nagyobb méretű annál, semhogyan létrehozzuk vagy használjuk.
- A valószínűségi axiómák kényeszereket jelentenek az állításokhoz ésszerűen hozzárendelt valószínűségek értékére vonatkozóan. Az axiómákat megsértő ágens bizonyos körülmények között ésszerűtlenül fog viselkedni.
- Ha rendelkezésre áll a teljes valószínűségi eloszlás, akkor a lekérdezések megválasztását egyszerűen a lekérdezések állításához tartozó elemi események bejegyzéseinek összeadásával elvégezhetjük.
- A véletlen változók részhalmazai között fennálló **abszolút függetlenség (absolute independence)** maga után vonja a teljes együttes valószínűségi eloszlás kisebb együttes valószínűségi eloszlásokból álló szorzattényezőkre bontásának lehetőségét. Ez nagymértékben csökkentheti a bonyolultságot, azonban a gyakorlatban ritkán fordul elő.
- A **Bayes-tétel (Bayes' rule)** lehetővé teszi, hogy ismert feltételes valószínűségekből számítsunk ki ismeretleneket, általában az okozat irányába következtetve. Nagy mennyiségi tény rendelkezésre állása esetén a Bayes-tétel alkalmazása ugyanolyan számításigény-robbanáshoz vezet, mint a teljes együttes valószínűségi eloszlás.

- A tartományon belüli közvetlen oksági kapcsolatok okozta **feltételes függetlenség (conditional independence)** lehetővé teszi, hogy a teljes valószínűségi eloszlást kisebb, feltételes valószínűségi eloszlások szorzataként írhassuk fel. A **naiv Bayes**-(**naive Bayes**) modell adott okváltozó mellett feltételezi az okozati változók feltételes függetlenségét, a modell mérete pedig az okozati változók számával lineárisan nő.
- A wumpus világ ágense ki tudja számítani a világ meg nem figyelt eseményeinek valószínűségét is, és azokat használva jobb következtetésekre tud jutni, mint egy egyszerű logikai ágens.

## Irodalmi és történeti megjegyzések

Bár a szerencsejátékok már i. e. 300 körül is ismertek voltak, az esélyek és a valószínűségek matematikai analízise jóval későbbre tehető. Mahaviracarya Indiában nagyjából az i. e. 9. században kezdte meg a téma vizsgálatát. Európában az első kísérletek csak az olasz reneszánsz idejére, kb. 1500-ra tehetők. Az első jelentős következetes elemzés Girolamo Cardano nevéhez fűződik (1565), amely azonban 1663-ig nem került nyilvánosságra. Ekkorra Blaise Pascalnak a valószínűségek szisztematikus kiszámítására vonatkozó felfedezése (Pierre Fermat-val levelezve 1654-ben) megalapozta a valószínűség-számítást, mint a matematika széles körben és eredményesen vizsgált ágát. Az első valószínűségekkel kapcsolatos tankönyv a *De Ratiociniis in Ludo Aleae* volt (Huygens, 1657). Pascal bevezette a feltételes valószínűség fogalmát is, amelyről szintén szó van Huygens könyvében. Thomas Bayes anglikán lelkész (1702–1761) nevéhez fűződik a később róla elnevezett szabály, amely feltételes valószínűségek számítására alkalmas. Eredménye csak halála után jelent meg nyomtatásban (Bayes, 1763). Kolmogorov volt az első (Kolmogorov, 1950; első megjelenés németül 1933), aki a valószínűség-számítást szigorúan axiomatikus keretek között tárgyalta. Rényi (Rényi, 1970) olyan axiómarendszert vezetett be, amely nem a feltétel nélküli, hanem a feltételes valószínűség fogalmára épül.

Pascal többféleképpen használta a valószínűség fogalmát, így igényelte minden az objektívista értelmezést – amely szerint a valószínűség a világhoz tartozó, szimmetrián vagy relatív gyakoriságon alapuló sajtság –, minden pedig a valószínűség meggödődési mértéken alapuló szubjektív megközelítését: előbbi a szerencsejátékra vonatkozó valószínűség elemzéseiben, míg az utóbbi a híres pascali fogadás érvélésében Isten létezése vagy nemléte kapcsán. Mindemellett, Pascal nem látta világosan a két megközelítés közötti különbséget. Ezt jól érthetően először James Bernoulli (1654–1705) fogalmazta meg.

Leibniz nevéhez fűződik a valószínűség klasszikus fogalmának – mint a számba vett azonos esélyű esetek arányának – leírása. Ugyanezt a fogalmat használta Bernoulli is, bár igazából Laplace (1749–1827) vezette be a gyakorlatba. A fogalom azonban a frekvencionista, illetve a szubjektív értelmezési mód között nem tud különbséget tenni. Az esetek tekinthetők azonos esélyüknek akár a természetes fizikai szimmetriák miatt, akár pedig egyszerűen azért, mert nincs olyan információink, amely arra mutatna, hogy az egyik bekövetkezésének nagyobb az esélye, mint a másikénak. Ez utóbbi, amikor egyéni megfontolások alapján igazoljuk az egyenlő valószínűségek hozzárendelését, a **pártatlanság elvének (principle of indifference)** hívjuk (Keynes, 1921).

A valószínűség objektivista és szubjektivista értelmezése közötti vita a 20. században kiélesedett. A relatív gyakoriság szószólói Kolmogorov (Kolmogorov, 1963), R. A. Fisher (Fisher, 1922) és Richard von Mises (von Mises, 1928) voltak. Karl Popper (Popper, 1959; első megjelenés németül 1934) hajlam (propensity) értelmezése a mögöttes fizikai szimmetriát látja a relatív gyakoriságban. Frank Ramsey, Bruno de Finetti, R. T. Cox, Leonard Savage és Richard Jeffrey úgy értelmezték a valószínűséget, mint meghatározott egyének meggyőződésének mértékét (Ramsey, 1931; de Finetti, 1937; Cox, 1946; Savage, 1954; Jeffrey, 1983). A hiedelem mértékére vonatkozó elemzésük szorosan kötődött a hasznossághoz és a viselkedéshez, különösen ahhoz a szándékhöz, hogy fogadásokat kössünk. Rudolf Carnap, Leibnizet és Laplace-t követve, a szubjektív valószínűség egy másfajta értelmezését javasolta: nem mint az adott egyén meggyőződését, hanem mint azt a hiedelemfokot, amelyet egy eszményi személynek *kell* hinnie egy adott *a* állításról e tény fennállása esetén. Carnap megpróbált Leibniz-nél és Laplace-nál tovább menni, és ezt a **megerősítési (confirmation)** mérték fogalmat, mint *a* és *e* között fennálló logikai kapcsolatot, matematikailag is pontosan leírni. A kapcsolat tanulmányozása révén akarta a megszokott deduktív logika (Carnap, 1948; 1950) mintájára, a matematika egy **induktív logikának (inductive logic)** nevezett disziplináját megalapítani. Carnap képtelen volt az induktív logikát lényegesen kiterjeszteni az ítéletkalkuluson túlra, és Putnam mutatta meg, hogy alapvető nehézségek hiúsítják meg az aritmetikát is kifejezni képes nyelvek irányába történő szabatos kiterjesztést (Putnam, 1963).

A vonatkoztatási osztály kérdése szorosan kötődik az induktív logika megtalálására tett törekvésekhez. Az elégséges méretű legjobban specifikus vonatkoztatási osztály ki-választására való törekvést Reichenbach javasolta (Reichenbach, 1949). Erőfeszítések történtek kifinomultabb irányelvek kialakítására, amelyekkel el lehet kerülni a Reichenbach-szabályból következő, Henry Kyburg által észlelt ellentmondásokat (Kyburg, 1977; 1983), azonban ezek a törekvések leginkább *ad hoc* jellegűek maradtak. Egy újabb, Bacchus, Grove, Halpern és Koller nevéhez fűződő munka (Bacchus és társai, 1992) Carnap módszereit kiterjeszti az elsőrendű elméletekre, kiküszöbölte ezáltal sok, a vonatkoztatási osztályokkal kapcsolatos nehézséget.

A bayesi valószínűségi következetető rendszereket a hatvanas évek óta használják az MI-ben, különösen az orvosi diagnosztikában. Nemcsak azért, hogy meglevő tényekből következtetéseket vonjanak le, hanem azért is, hogy az információérték-elmélet (lásd 16.6. alfejezet) segítségével további kérdéseket és vizsgálatokat válasszanak, amikor a meglevő tények nem meggyőzők (Gorry, 1968; Gorry és társai, 1973). Az egyik rendszer felül is múulta az emberi szakértőket a heveny hasi megbetegedések diagnózisában (de Dombal és társai, 1974). Mindemellett azonban ezekkel a korai bayesi rendszerekkel igen sok gond is volt. Lévén, semmilyen elméleti modellel nem rendelkeztek azokról a körül-ményekről, amelyeket diagnosztizáltak, sebezhetők voltak mindenkoron esetekben, amikor olyan helyzetekre jellemző adatok tűntek fel, amelyeket csak kisszármú mintahalmaz képviselt (de Dombal és társai, 1981). Még ennél is alapvetőbb, hogy mivel nem volt tömör leíró módszerük (mint amilyet a 14. fejezetben ismertetünk) a feltételes függetlenség megjelenítésére és használatára, erősen függtek a hatalmas mennyiségi valószínűségi adat gyűjtésétől, tárolásától és feldolgozásától. E nehézségek miatt 1970-től a nyolcvanas évek közepéig a bizonytalanságkezelés valószínűségi módszerei kegyvesztettek lettek az MI-ben. A nyolcvanas évek vége óta bekövetkezett fejlődést a következő fejezet írja le.

Az együttes eloszlások naiv bayesi megközelítését az alakfelismeréssel foglalkozó irodalom alaposan tanulmányozta az 1950-es évektől kezdve (Duda és Hart, 1973). A megközelítést Maron munkájának megjelenése (Maron, 1961) óta gyakran használták – sokszor akaratlanul – az adatbányászat területén is. A módszer valószínűségi megalapozását, amelyet részletesen a 13.18. feladat ír le, Robertson és Sparck Jones tisztázta (Robertson és Jones, 1976). A naiv Bayes-következetesnek a még olyan tartományokban is jelentkező meglepő sikérére, ahol a függetlenségi feltételezés egyértelműen sérül, Domingos és Pazzani ad magyarázatot (Domingos és Pazzani, 1977).

A valószínűség-számítás téma körében nagyon sok kiváló tankönyv létezik, beleértve a Chung és Ross által írtakat (Chung, 1979; Ross, 1988). Morris DeGroot két könyve közül az egyik (DeGroot, 1989) a bayesi álláspont felől közelíti a közös bevezetést a valószínűség-számításba és a statisztikába, a másik pedig mélyebb ismereteket tartalmaz (DeGroot, 1970). Richard Hamming tankönyve matematikailag megalapozott bevezetést nyújt a valószínűség-számítás elméletébe a fizikai szimmetrián alapuló hajlammegközelítés felől (Hamming, 1991). Hacking és Hald a valószínűség fogalmának korai történetét tárgyal elénk (Hacking, 1975; Hald, 1990), Bernstein pedig szórakoztató népszerűsítő áttekintést ad a kockázatelemzés történetéről (Bernstein, 1996).

## Feladatok

- 13.1.** Mutassa meg az alapelvek segítségével, hogy

$$P(a|b \wedge a) = 1$$

- 13.2.** A valószínűség-számítás axiómáit használva bizonyítsa be, hogy diszkrét véletlen változók tetszőleges valószínűségi eloszlása összegének 1-et kell adnia.
- 13.3.** Ésszerű lehet-e egy ágens számára a következő három meggyőződésben hinni:  $P(A) = 0,4$ ,  $P(B) = 0,3$  és  $P(A \vee B) = 0,5$ ? Ha igen, milyen valószínűségsávot tart ésszerűnek  $A \wedge B$ -re vonatkozóan? Készítsen a 13.2. ábrához hasonló táblázatot, és mutassa meg, hogyan támasztja ez alá az Ön érvelését. Ezután készítsen egy másik változatot, amelynél  $P(A \vee B) = 0,7$ . Magyarázza el, miért racionális egy ekkora valószínűség hozzárendelése, mindenellett hogy a táblázat mutat egy olyan esetet, ami veszteség, és három olyat, ami ki van egyenlítve. (Segítség: milyen valószínűségi hozzárendelés mellett kötelezte el magát az 1. ágens a négy esetben, de különösen a veszteség esetén?)
- 13.4.** Ez a kérdés az 551–552. oldalon tárgyaltaknak megfelelően, az elemi események tulajdonságaival foglalkozik.
- (a) Bizonyítsa be, hogy az összes lehetséges elemi esemény egyesítése logikailag megegyezik az igaz állítással. (Segítség: bizonyítékként alkalmazzon indukciót a véletlen változók számára vonatkozóan.)
  - (b) Bizonyítsa be, hogy bármely állítás logikailag ekvivalens azon elemi események diszjunkciójával, amelyek maguk után vonják annak igazságát.

- 13.5.** Képzeljük el a szokványos 52 lapos kártyacsomagból osztott ötlapos póker játszmák tartományát, annak feltevéssével, hogy az osztó becsületes.
- Hány elemi eseményt tartalmaz az együttes valószínűség-eloszlás (azaz hányfélé 5 lapos leosztás létezik egy kézben)?
  - Mekkora a valószínűsége az egyes elemi eseményeknek?
  - Mekkora a valószínűsége a felül ászos színsornak (royal flush, egyszínű ász, király, dáma, bubi és tízes)? És négy egyforma lapnak?
- 13.6.** Számítsa ki az alábbiakat, ha adott a 13.3. ábra szerinti teljes együttes valószínűségi eloszlás.
- $P(\text{fogfájás})$
  - $P(\text{Lyuk})$
  - $P(\text{Fogfájás}|\text{lyuk})$
  - $P(\text{Lyuk}|\text{fogfájás} \vee \text{breakadás})$
- 13.7.** Mutassa meg, hogy a függetlenség (13.8) egyenlet szerinti három felírási módja ekvivalens egymással.
- 13.8.** Az évenként esedékes orvosi ellenőrzése után a doktor rossz és jó híreket mond. A rossz hír az, hogy komoly betegséget mutattak ki önnél, és a teszt 99%-ban megbízható (azaz, annak a valószínűsége, hogy a teszt kímutatja a meglevő betegséget, 0,99, és ugyanígy, annak a valószínűsége, hogy a teszt negatív lesz, amennyiben nem áll fenn a betegség, szintén 0,99). A jó hír az, hogy ez a betegség nagyon ritka, amelyet az Ön korában minden 10 000 emberből csak egy kap meg. Miért jó hír az, hogy a betegség ritka? Mekkora az esélye annak, hogy Ön tényleg beteg?
- 13.9.** Sokszor célszerűbb adott állítások hatását bizonyos rögzítettnek feltételezett általános háttér tények feltételezésével végiggondolni, mint az információ teljes hiányában. Az alábbi kérdések a szorzatszabály és a Bayes-tétel általános alakjainak bizonyítását célozzák, valamely e háttér tény feltételezése mellett.
- Bizonyítsa be az általános szorzatszabály valamely feltétel fennállása esetén felírható alakját:
- $$\mathbf{P}(X, Y|e) = \mathbf{P}(X|Y, e)\mathbf{P}(Y|e)$$
- Bizonyítsa be a Bayes-tétel (13.10) egyenlet szerinti, valamilyen feltétel fennállása esetére felírt változatát.
- 13.10.** Mutassa meg, hogy a
- $$\mathbf{P}(A, B|C) = \mathbf{P}(A|C)\mathbf{P}(B|C)$$
- állítás minden alábbi állítással egyenértékű
- $$\mathbf{P}(A|B, C) = \mathbf{P}(A|C) \text{ és } \mathbf{P}(B|A, C) = \mathbf{P}(B|C)$$
- 13.11.** Tegyük fel, hogy kapott egy  $n$  darab egyforma pénzérmét tartalmazó zsákokat. Azt is tudjuk, a pénzek közül  $n-1$  szabályos, azaz egyik oldalán fej, a má-

síkon írás található, ugyanakkor van egy darab hamis, amelynek minden két oldala fej.

- Tegyük fel, hogy benyül a zsákba, majd véletlenszerűen kiemel egy pénzdarabot. Ezt feldobva fejet kap eredményül. Mekkora a (feltételes) valószínűsége annak, hogy a kivett pénzdarab a hamis?
- Tegyük fel, hogy  $k$ -szor folytatva a pénz feldobását, ezután is minden alkalommal fejet kap. Most mennyi lesz a feltételes valószínűsége annak, hogy a kivett pénzdarab a hamis?
- Tegyük fel, hogy a  $k$ -szori feldobással akarta eldönteni, hogy a hamis vagy egy jó pénzérmet emelt ki a zsákból. A döntési eljárás HAMIS-at ad, ha minden  $k$  feldobás alkalmával a fej van fölül, egyébként pedig JÓ-t. Mekkora annak a (feltétel nélküli) valószínűsége, hogy az eljárás tévedni fog?

**13.12.** Ebben a feladatban befejezheti az agyhártyagyulladásra vonatkozó példa normalizálását. Először is vegyen fel egy megfelelő értéket  $P(S|\neg M)$ -re, és ennek segítségével számítsa ki  $P(M|S)$  és  $P(\neg M|S)$  normalizálatlan értékeit (azaz ne vegye figyelembe  $P(S)$ -t a Bayes-tétel kifejezésében), majd normalizálja az értékeket úgy, hogy összegük 1 legyen.

**13.13.** Ez a feladat azt vizsgálja, hogy a feltételes függetlenség fennállása hogyan befolyásolja a valószínűségi becslésekhez szükséges információ mennyiségett.

- Tegyük fel, hogy  $P(h|e_1, e_2)$ -t szeretnénk kiszámítani úgy, hogy nincs feltételes függetlenségre vonatkozó információink. Az alábbiak közül melyik az az adathalmaz, amelyikre a számításokhoz szükségünk van?
  - $\mathbf{P}(E_1, E_2), \mathbf{P}(H), \mathbf{P}(E_1|H), \mathbf{P}(E_2|H)$
  - $\mathbf{P}(E_1, E_2), \mathbf{P}(H), \mathbf{P}(E_1, E_2|H)$
  - $\mathbf{P}(H), \mathbf{P}(E_1|H), \mathbf{P}(E_2|H)$
- Tegyük fel, hogy tudjuk, hogy  $\mathbf{P}(E_1|H, E_2) = \mathbf{P}(E_1|H)$ ,  $H, E_1, E_2$  minden értékére. Hogyan változik az előbbi kérdésre adott válasz?

**13.14.**  $X, Y$  és  $Z$  legyenek logikai véletlen változók. A  $\mathbf{P}(X, Y, Z)$  együttes eloszlás nyolc lehetséges esetét jelöljük  $a, \dots, h$ -val. Fejezze ki az „ $X$  és  $Y$  adott  $Z$  mellett feltételesen függetlenek” állítást az  $a, \dots, h$ -t összekapcsoló egyenletek segítségével. Hány lesz ezek közül *redundaciamente*?

**13.15.** (A (Pearl, 1988)-ból átvéve.) Képzeljük el, hogy Ön egy éjszakai taxis cserben-hagyásos baleset tanúja Athénban. Athénban minden taxi kék vagy zöld. Ön azt vallja eskü alatt, hogy a taxi kék volt. Egy széles körű vizsgálat azt mutatja, hogy ilyen gyenge fényben a kék és zöld szín közötti tévesztés valószínűsége 75%. Kiszámítható-e, hogy milyen színű taxi volt a legvalószínűbb? (Segítség: gondosan különböztessük meg azt az állítást, hogy a taxi kék, és azt, hogy a taxi kéknek tűnik.)

Mi van akkor, ha minden 10 athéni taxiból 9 zöld?

- 13.16.** (A (Pearl, 1988)-ból átvéve.) Három fegyenc,  $A$ ,  $B$  és  $C$  celláikba vannak zárva. Közismert, hogy egyiküket másnap ki fogják végezni, míg a másik kettő kegyelmet kap. Azt, hogy kit végeznek ki, csak a börtönigazgató tudja. A fegyenc egy kéréssel fordul a börtönőrhöz: „Legyen szíves, kérdezze meg az igazatot, hogy kit fognak holnap kivégezni, és a barátaim közül az egyiknek,  $B$ -nek vagy  $C$ -nek árulja el, hogy reggel kegyelmet fog kapni.” Az őr beleegyezik, majd később azzal jön vissza, hogy  $B$ -nek szolt a kegyelemről.  
 Mekkorák  $A$  esélyei a kivégzésre a történtek után? (Ne csak erőteljes kézlegyintéssel, hanem matematikailag megalapozottan válaszoljon.)
- 13.17.** Írja meg azt az általános, naiv Bayes-eloszlást használó algoritmust, amely alkalmas  $P(Ok|e)$  alakú keresések megválaszolására. Fel kell tételeznie azt is, hogy az  $e$  bizonyíték az érintett változók *tetszőleges részhalmazához* rendel értékeket.
- 13.18.** A szövegosztályozás olyan feladat, amely egy adott dokumentumot – a benne levő szöveg alapján – előre rögzített kategóriák valamelyikébe sorol be. Erre gyakran használnak naiv Bayes-modellt. Ezekben a modellekben a lekérdezés változója a dokumentumkategória és az „okozati” változók a nyelv egyes szavainak megléte vagy hiánya. Feltételezzük azt is, hogy a szavak egymástól függetlenül, a dokumentumkategóriára jellemző sűrűsséggel jelennek meg a szövegekben.
- Magyarázza el pontosan, hogyan lehet egy ilyen modellt létrehozni, ha „stanitó mintaként” adott egy, már osztályozott dokumentumhalmaz.
  - Magyarázza el pontosan, hogyan történik egy új dokumentum besorolása.
  - Elfogadható-e a függetlenség feltételezése? Részletezze.
- 13.19.** A wumpus világ vizsgálatánál abból a tényből indultunk ki, hogy a négyzetek a többi négyzet tartalmától függetlenül 0,2 valószínűsséggel tartalmaznak csapdát. Ehelyett most tegyük fel, hogy az  $[1, 1]$ -en kívüli  $N$  négyzetben pontosan  $N/5$  csapda van egyenletes eloszlásban véletlenszerűen elhelyezve. Függetlenek-e továbbra is a  $C_{i,j}$  és a  $C_{k,l}$  változók? Hogyan fog most kinézni a  $P(C_{1,1}, \dots, C_{4,4})$  együttes eloszlását? Számolja újra az  $[1, 3]$  és a  $[2, 2]$  négyzetekben található csapda valószínűségét.

# 14. VALÓSZÍNŰSÉGI KÖVETKEZTETÉS

Ebben a fejezetben ismertetjük, hogyan építhetők olyan következető rendszerek, amelyek hálós modelleket használnak a bizonytalanságnak a valószínűség-számítás törvényeinek megfelelő kezelésére.

A 13. fejezetben bemutattuk a valószínűség-számítás szintaxisát és szemantikáját. Külföldön hangsúlyoztuk a függetlenségi és a feltételes függetlenségi kapcsolatok fontosságát a világ valószínűségi reprezentációjának az egyszerűsítésében. Ez a fejezet egy szisztematikus módszert vezet be az ilyen relációk explicit reprezentálására, a Bayes-hálók formájában. Ezeknek a hálózatoknak definiáljuk a szintaxisát és szemantikáját, és mutatjuk, hogyan használhatók fel bizonytalan tudás megragadására természetes és hatékony módon. Ezt követően ismertetjük, hogyan végezhető el a valószínűségi következtetés, ami bár legrosszabb esetben exponenciális komplexitású, a gyakorlatban sok esetben hatékonyan alkalmazható. Leírunk továbbá különböző közelítő következetet algoritmusokat, amelyek gyakran akkor is alkalmazhatók, amikor egzakt következetet nem lehetséges. Megvizsgálunk olyan módszereket is, hogy hogyan alkalmazható a valószínűség-számítás egy objektumokat és relációkat is tartalmazó világra – azaz az *ítéletelőgikával* szemben az *elsőrendű* reprezentációkra. Végül áttekintjük a bizonytalan következtetés egyéb alternatív megközelítéseit.

## 14.1. A TUDÁS REPREZENTÁLÁSA BIZONYTALANSÁG ESETÉN

A 13. fejezetben láttuk, hogy az együttes valószínűség-eloszlás függvény alapján minden kérdés megválaszolható a modellezett tárgytartománnyal kapcsolatban, de a változók számának növekedésével ez általában kivitelezhetetlené válik. Továbbá az elemi események valószínűségének megadása is igen mesterkélt és nehéz is lehet, ha csak nem áll rendelkezésre nagy mennyiségi adat a valószínűségek statisztikai becsléséhez.

A változók közötti függetlenségi és feltételesek függetlenségi relációkkal kapcsolatban azt is láttuk, hogy ezek nagyban csökkenhetik az együttes valószínűség-eloszlás függvény megadásához szükséges valószínűségek számát. Ez a fejezet bevezet egy Bayes-hálónak (*Bayesian network*<sup>1</sup>) nevezett adatstruktúrát a változók közötti függőség leírásához, és bármely együttes valószínűség-eloszlás függvény tömör megadásához.

<sup>1</sup> Az angol nyelvű szakirodalomban a Bayes-háló a legelterjedtebb megnevezés, de számos más is létezik: bizonyosságháló (*belief network*), valószínűségi háló (*probabilistic network*), okozási háló (*causal network*), tudásháló (*knowledge map*). A statisztikában a gráfos modell (*graphical model*) bővebb modellszintűt jelent, ami a Bayes-hálót is magában foglalja. A Bayes-hálók egy kiterjesztésének neve a döntési háló (*decision network*) vagy hatásdiagram (*influence diagram*) – lásd 16. fejezet.

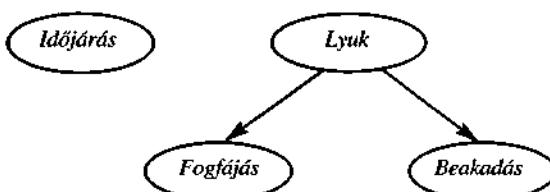
A Bayes-háló egy irányított gráf, amelyben minden *csomóponthoz* számszerű valószinűségi információk vannak csatolva. A teljes megadás a következő:

1. A háló csomópontjait valószinűségi változók egy halmaza alkotja. A változók lehetnek diszkrétek vagy folytonosak.
2. Irányított élek (nyílak) egy halmaza összeköt bizonyos csomópontpárokat. Ha létezik nyíl az  $X$  csomóponttól az  $Y$  csomópontig, azt mondjuk, hogy az  $X$  a *szülője* az  $Y$ -nak.
3. minden  $X_i$  csomóponthoz tartozik egy  $P(X_i|Szülöök(X_i))$  feltételes valószinűség-eloszlás, ami számszerűen megadja a szülők hatását a csomóponti változóra.
4. A gráf nem tartalmaz irányított kört (azaz irányított, körmentes gráf – Directed Acyclic Graph, DAG).

A háló topológiája – a csomópontok és élek halmaza – megadja a tárgyterületen fennálló feltételes függetlenségi kapcsolatokat, hogy mily módon, azt hamarosan pontosan kifejtjük. Egy helyesen létrehozott hálóban az  $X$  csomópontot az  $Y$  csomóponttal összekötő nyíl *intuitív* jelentése rendszerint az, hogy az  $X$ -nek közvetlen befolyása van az  $Y$ -ra. A tárgyterület szakértője számára általában könnyű eldönten, hogy milyen közvetlen befolyások teljesülnek egy adott területen – valójában sokkal könnyebb, mint a megfelelő valószinűségeket megadni. Ha pedig a Bayes-háló topológiája kész, már csak az egyes változókhöz tartozó feltételes valószinűség-eloszlásokat kell meghatározni a szülőkkel mint feltételekkel. Látni fogjuk, hogy a topológia és a feltételes eloszlások együttese elegendő, hogy megadja (implicit módon) az összes változó feletti együttes valószinűség-eloszlás függvényt.

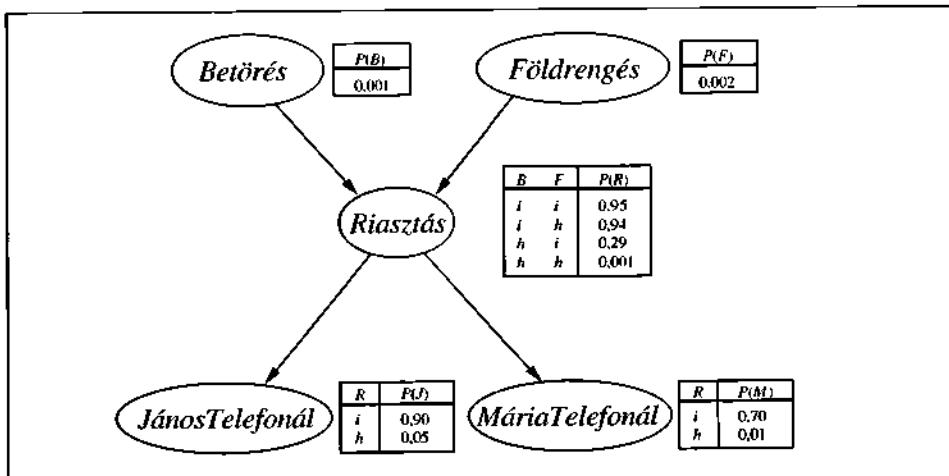
Idézzük fel a 13. fejezetben leírt egyszerű világot, amely a *Fogfájás*, a *Lyuk*, a *Beakadás* és az *Időjárás* változót tartalmazza. Úgy érvelünk, hogy az *Időjárás* független az összes többi változótól; továbbá, hogy a *Fogfájás* és a *Beakadás* feltételesen függetlenek a *Lyuk* ismeretében. Ezeket a viszonyokat a 14.1. ábrán bemutatott Bayes-háló reprezentálja. Formálisan, a *Fogfájás* és a *Beakadás* feltételes függetlenségét a *Lyuk* ismeretében a *Fogfájás* és a *Beakadás* közötti nyíl hiánya jelzi. Szemléletesen, a háló azt a tényt fejezi ki, hogy a *Lyuk* közvetlen oka a *Fogfájás*-nak és a *Beakadás*-nak, további közvetlen okozati kapcsolat azonban nem létezik a *Fogfájás* és a *Beakadás* között.

Most vizsgáljuk meg a közvetkező példát, ami csak egy árnyalatnyival összetettebb. Otthonunkban egy új betörésjelzőt szereltek fel. Ez megbízhatóan észleli a betöréseket, de idónként kisebb földrengések esetén is jelez. (Ez a példa Judea Pearl-től származik, aki Los Angeles-i lakos; innen ered a földrengések iránti külö-



14.1. ábra. Egy egyszerű Bayes-háló, amelyben az *Időjárás* független a többi három változótól, a *Fogfájás* és a *Beakadás* pedig feltételesen függetlenek a *Lyuk* ismeretében

nős érdeklődése.)<sup>2</sup> Két szomszédunk is van, János és Mária, akik megígérték, hogy felhívnak a munkahelyükön, ha meghallják a riasztót. János minden felhív mincket, ha meghallja a riasztást, de néha összekeveri a telefoncsörgést a riasztó csengésével, és ekkor is telefonál. Mária viszont, mivel szereti hangosan hallgatni a zenét, néha meg sem hallja a riasztót. Mi tehát a hívások bekövetkezése vagy hiánya alapján szeretnénk megbecsülni a betörés valószínűségét. Ezt az egyszerű problémát a 14.2. ábrán látható Bayes-háló írja le.



14.2. ábra. Egy tipikus Bayes-háló, amely a topológiát és a feltételes valószínűségi táblákat (FVT) is mutatja. Az FVT-kben  $B$ ,  $F$ ,  $R$ ,  $J$  és  $M$  szerepel Betörés, Földrengés, Riasztás, JánosTelefonál és MáriaTelefonál helyett.

Egyelőre figyelmen kívül az ábrán szereplő feltételes eloszlásokat, és összpontosítunk a háló topológiájára. A betörés háló esetén a topológia azt mutatja, hogy a betörés és a földrengés közvetlenül befolyásolja a riasztó megszólalásának valószínűségét, ellenben János vagy Mária hívásának bekövetkezése csak magán a riasztón műlik. A háló így tartalmazza azon feltevéseinket, hogy ők közvetlenül nem vesznek észre betöréseket, nem vesznek észre kisebb földrengéseket, és nem egyeztetnek hívás előtt.

Megfigyelhető, hogy a hálóban nincsenek olyan csomópontok, amelyek azt írnák le, hogy Mária éppen hangos zenét hallgat, vagy hogy a telefon cseng, és megzavarja Jánost. Ezeket a tényezőket a Riasztás csomóponttól a JánosTelefonál és MáriaTelefonál csomópontig tartó élkekhez rendelt bizonytalanság foglalja magában. Ebben mind a lusitaság, mind pedig a tudatlanság tetten érhető: rengeteg munka volna olyan tényezők meghatározása, amelyek valószínűbbé vagy kevésbé valószínűvé válnának adott esetekben, és nincs semmilyen ésszerű módszerünk, hogy ezekre vonatkozóan érdemi információt szerezzünk. A valószínűségek valójában a lehetséges körielmények egy potenciálisan végtelen halmazát összegzik, amikor is a riasztó elmulaszt megszólalni

<sup>2</sup> Judea Pearl meghatározó szerepet játszott a Bayes-hálók egzakt matematikai háttérének és gyakorlati alkalmazhatóságának a vizsgálatában. (A ford.)

(magas páratartalom, elektromos hálózat hibája, lemerült elem, elvágott drót, döglött egér beragadva a csengőbe...), vagy, hogy János és Mária elmulaszt értesíteni minket (ebédelni mentek, szabadságon vannak, időlegesen megsüketültek, egy áthaladó helikopter...). Ezen a módon egy kis ágens is képes megbirkózni egy igen bonyolult világgal, legalábbis közelítőleg. A közelítés mértéke tetszőlegesen javítható további releváns információk bevezetésével.

Most folytassuk a 14.2. ábrán mutatott feltételes eloszlásokkal. Az ábrán minden eloszlás mint **feltételes valószínűségi táblázat – FVT (conditional probability table, CPT)** van feltüntetve. (A táblázatos formát diszkrét változók esetén lehet használni; más reprezentációkat, amelyek már folytonos változóknál is használhatók, a 14.2. alfejezetben írunk le.) Az FVT-táblázatban minden sor az egyes csomóponti értékek feltételes valószínűségét tartalmazza az adott sorhoz tartozó **szülői feltétel (conditioning case)** esetén. A szülői feltétel a szülő csomópontok értékeinek egy lehetséges kombinációja (egyfajta elemi esemény, ha úgy tetszik). Az egyes sorokban szereplő számok összegének 1-et kell adnia, mivel az adott változó összes lehetséges értéke szerepel a bejegyzésekben. Bináris változók esetén, ha ismerjük, hogy az igaz érték valószínűsége  $p$ , a hamis érték valószínűségének  $1 - p$ -nek kell lennie, így gyakran a második számot elhagyjuk, ahogyan ezt a 14.2. ábrán is tettük. Általánosabban, egy  $k$  bináris szülővel rendelkező bináris változó esetén  $2^k$  valószínűség adható meg tetszés szerint. Szülő nélküli csomópontok esetén a táblázat csak egyetlen sort tartalmaz, a változó egyes értékeinek a priori valószínűségeit.

## 14.2. A BAYES-HÁLÓK SZEMANTIKÁJA

Az előző alfejezetben bemutattuk a hálót, de annak jelentését nem. A Bayes-hálók szemantikáját kétféle módon lehet megérteni. Az első szerint a háló az együttes valószínűség-eloszlás függvény egy leírása. A második szerint a háló feltételes függetlenségekről szóló állítások együttesét írja le. A két szemlélet ekvivalens, de az első inkább abban segít, hogyan *huzzunk létre* egy hálót, míg a második a következtetési eljárások tervezését segíti.

### Az együttes valószínűség-eloszlás függvény leírása

A Bayes-háló a tárgytartomány teljes leírását adja meg. A benne található információk segítségével az együttes valószínűség-eloszlás függvény bármely bejegyzése kiszámítható.<sup>3</sup> Az együttes valószínűség-eloszlás függvény egy általános bejegyzése egy teljes – minden egyes változóhoz történő – hozzárendelés konjunkciójának a valószínűsége, úgymint  $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$ . Erre a továbbiakban a  $P(x_1, \dots, x_n)$  rövidítést fogjuk használni. Egy bejegyzés értékét a következő egyenlőség adja meg:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{szülők}(X_i)) \quad (14.1)$$

<sup>3</sup> A diszkrét változóra utaló szóhasználat ellenére az állítások analógjai folytonos változók esetén is fennállnak. (A ford.)

ahol a  $Szülők(X_i)$  a  $Szülők(X_i)$ -ben szereplő változók adott értékeinek együttesét jelöli. Így az együttes valószínűség-eloszlás függvényt leíró táblázat minden bejegyzése a Bayes-hálóban szereplő, feltételes valószínűségi táblák (FVT) megfelelő elemeinek a szorzata. Ezzel az FVT-k valójában az együttes valószínűség-eloszlás függvény dekomponált leírását adják meg. Ennek illusztrálására kiszámítjuk annak az eseménynek a valószínűségét, hogy a riasztó megszólal, de nem volt sem betörés, sem földrengés, azonban János és Mária is telefonál. A kezdetűvel jelölve a változókat:

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) \\ = P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e) \\ = 0,90 \times 0,70 \times 0,001 \times 0,999 \times 0,998 = 0,00062 \end{aligned}$$

A 13.4. alfejezetben bemutattuk, hogy az együttes valószínűség-eloszlás függvény alapján a tárgytartománnyal kapcsolatos bármely kérdés megválaszolható. Ha egy Bayes-háló leírja az együttes valószínűség-eloszlás függvényt, akkor ez alapján bármely kérdés megválaszolható, összegezve a releváns együttes bejegyzéseket. A 14.4. alfejezet elmagyarázza, hogy ez hogyan végezhető el, azonban sokkal hatékonyabb módszereket is ismertet.

### Egy módszer Bayes-hálók építésére

A (14.1) egyenlet definiálja, hogy mit is jelent egy adott Bayes-háló. Azonban arra nézve nem ad felvilágosítást, hogyan építhetünk olyan Bayes-hálót, hogy az általa meghatározott együttes valószínűség-eloszlás függvény az adott tárgytartomány megfelelő leírása legyen. Most megmutatjuk, hogy a (14.1) egyenlet tartalmaz bizonyos feltételes függetlenségi relációkat, amelyeket a tudásmérnök felhasználhat a háló topológiájának meghatározásánál. Elsőként írjuk fel az együttes valószínűség-eloszlás függvényt feltételes valószínűségek szorzataként, felhasználva a szorzatszabályt (lásd 13. fejezet):

$$P(x_1, \dots, x_n) = P(x_n|x_{n-1}, \dots, x_1)P(x_{n-1}, \dots, x_1)$$

Majd ismételten alkalmazzuk ezt a lépést, minden egyes együttes valószínűséget felbontva egy feltételes valószínűségre és egy kisebb együttes valószínűségre. Végezetül egyetlen hosszú szorzatot kapunk:

$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_n|x_{n-1}, \dots, x_1)P(x_{n-1}|x_{n-2}, \dots, x_1) \cdots P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i|x_{i-1}, \dots, x_1) \end{aligned}$$

Ez a **láncszabályként (chain rule)** ismert azonosság valószínűségi változók bármely halmazára fennáll.

Összehasonlítva ezt a (14.1.) egyenettel láthatjuk, hogy az együttes valószínűség-eloszlás függvény megadása ekvivalens azzal az általános állítással, hogy a háló minden  $X_i$  változójára

$$\mathbf{P}(X_i|X_{i-1}, \dots, X_1) = \mathbf{P}(X_i|Szülők(X_i)) \quad (14.2)$$

feltéve, hogy  $Szülők(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$ . Ez utóbbi feltétel a csomópontok bármely olyan sorszámozásával teljesíthető, ami konzisztens a gráf struktúrájából adódó implicit részleges rendezéssel.

A (14.2) egyenlet szerint, egy Bayes-háló csak abban az esetben lehet helyes reprezentációja a tárgytartománynak, ha az adott szülők mellett, minden csomópont feltételesen független a csomópontot sorrendezésben elő megelőzőktől. Így a tárgytartomány struktúrájának megfelelő Bayes-háló megépítése során, minden egyes csomóponthoz úgy kell a szülőket megválasztanunk, hogy ez a feltétel teljesüljön. Szemléletesen ez azt jelenti, hogy az  $X_i$  csomópont szülei halmazának tartalmaznia kell az  $\{X_1, \dots, X_{i-1}\}$  közül minden azokat a csomópontokat, amelyek közvetlenül befolyásolják  $X_i$ -t. Például tételezzük fel, hogy a 14.2. ábrán látható hálót már teljesen befejeztük, csupán a *MáriaTelefonál* szülei kell még megválasztanunk. A *MáriaTelefonál*-t egyértelműen befolyásolja, hogy történt-e *Betörés* vagy *Földrengés*, de nem közvetlenül. A helyzettel kapcsolatos ismereteink alapján tudhatjuk, hogy ezek az események csak a riasztó által befolyásolhatják Mária telefonálással kapcsolatos viselkedését. Hasonlóan, ha a riasztó állapota ismert, akkor János hívásának bekövetkezte vagy elmaradása már nincs hatással Mária telefonálására. Formálisan fogalmazva, úgy véljük, hogy a következő feltételes függetlenség teljesül:

$$\begin{aligned} P(\text{MáriaTelefonál} | \text{JánosTelefonál, Riasztás, Földrengés, Betörés}) \\ = P(\text{MáriaTelefonál} | \text{Riasztás}) \end{aligned}$$

## Tömörség és a csomópontok sorrendje

Amellett hogy egy teljes és nem redundáns reprezentációja a tárgytartománynak, egy Bayes-háló gyakran sokkal tömörebb, mint az együttes valószínűség-eloszlás függvény. Ez a tulajdonsága teszi használhatóvá a sokváltozós tárgyterületek kezelésében. A Bayes-háló tömörsége a **lokálisan strukturált** (*locally structured*) (vagy *ritka – sparse*) rendszerek egy igen általános tulajdonságának példája. Egy lokálisan strukturált rendszerben egy komponens csak korlátos számú más komponenssel van kapcsolatban közvetlenül, függetlenül a komponensek teljes számától. Lokális struktúrához általában inkább a lineáris, mint az exponenciális komplexitásnövekedés kapcsolható. A Bayes-háló esetében jogos azt feltételezni, hogy a legtöbb tárgytartomány esetén egy valószínűségi változót csak  $k$  számú más változó befolyásol, ahol  $k$  konstans. Ha bináris változókat tételezünk fel az egyszerűség kedvéért, akkor az egy csomóponthoz tartozó feltételes valószínűségi tábla megadásához legfeljebb  $2^k$  érték szükséges, így a teljes háló  $n2^k$  értékkel megadható. Ezzel szemben az együttes valószínűség-eloszlás függvény  $2^n$  értéket tartalmaz. Egy konkrét példával élve, ha 30 csomópontunk van ( $n = 30$ ), és mindegyiknek legfeljebb 5 szülője van ( $k = 5$ ), akkor a Bayes-háló 960 számot igényel, míg az együttes valószínűség-eloszlás függvény több mint egy milliárdot.

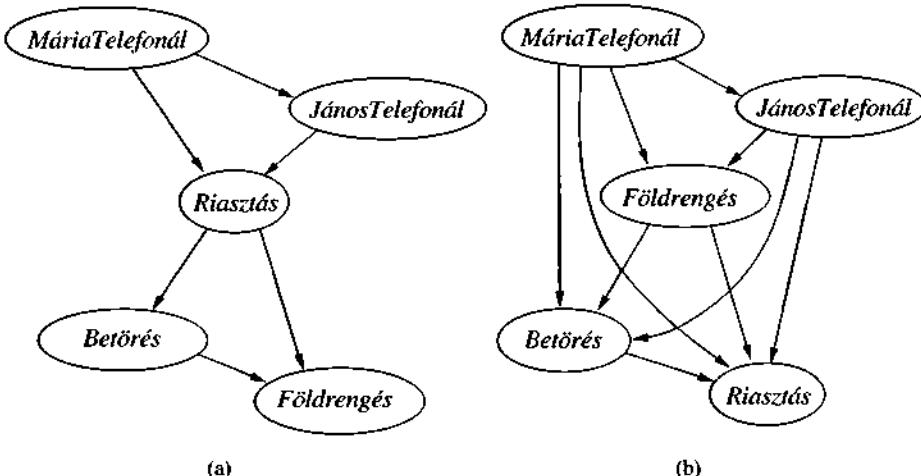
Léteznek olyan tárgytartományok, ahol a változók mindegyikét közvetlenül befolyásolhatja az összes többi, így a háló teljesen összekötött. A feltételes valószínűségi táblák megadása ekkor ugyanakkora mennyiséggű információt igényel, mint az együttes valószínűség-eloszlás függvény megadása. Bizonyos tárgytartományokban léteznek olyan gyenge függőségek, amiket feltétlenül modellezni kell egy új kapcsolat

felvételével. De ha ezek a függőségek igen gyengék, akkor lehet, hogy nem éri meg a háló komplexitását megnövelni a pontosság kismértékű növelésének érdekében. Például a betöréses hálónkkal kapcsolatban kifogásolható az, hogyha földrengés van, akkor Mária és János akkor sem telefonálna, ha hallanák a riasztót, mivel feltételezik, hogy a földrengés okozta. Az, hogy hozzákapcsoljuk-e a Földrengés-t a MáriaTelefonál-hoz és a JánosTelefonál-hoz (és így megnöveljük a táblákat) azon múlik, hogy mennyire fontos pontosabb valószínűségeket kapni, és mennyire költséges meghatározni az extra információt.

Azonban még egy lokálisan strukturált problémánál sem egyszerű lokálisan strukturált Bayes-hálót felépíteni. Ugyanis nemcsak azt követeljük meg, hogy minden egyes változót csak néhány másik befolyásoljon közvetlenül, hanem azt is, hogy a háló topológiája valóban mutassa azokat a közvetlen hatásokat a szülők megfelelő megválasztásával. A konstrukciós eljárásunk működése miatt előbb a „közvetlen befolyásolókat” kell a hálóhoz adni, ha azt szeretnénk, hogy szülőknek tudjuk őket választani az általuk befolyásolt csomópontnál. Ezért a helyes sorrend a csomópontok hozzáadásánál az, hogy először az „alapvető okokat” adjuk a hálóhoz, majd a változókat, amelyeket befolyásolnak, és ezt addig folytatjuk, amíg el nem érjük a „levelek”, amelyeknek már nincs közvetlen okozati hatása más változóra.

Mi történik, ha történetesen rossz sorrendet választunk? Vizsgáljuk meg újra a betöréses példát. Tételezzük fel, hogy a változókat a következő sorrendben adjuk a hálóhoz: MáriaTelefonál, JánosTelefonál, Riasztás, Betörés, Földrengés. Ekkor egy kicsit bonyolultabb hálót kapunk (14.3. (a) ábrán). Az eljárás a következő:

- MáriaTelefonál hozzáadása: szülők nincsenek.
- JánosTelefonál hozzáadása: ha Mária telefonál, az valószínűleg azt jelenti, hogy a riasztó megszólalt, ami természetesen valószínűbbé teszi, hogy János telefonáljon. Így a JánosTelefonál-nak szükségszerűen szülöje a MáriaTelefonál.



14.3. ábra. A háló struktúrája függ a hozzáadás sorrendjétől. A csomópontokat mindegyik hálóhoz fejlről lefelé haladva adtuk hozzá.

- Riasztás* hozzáadása: nyilvánvaló, ha mindenketten telefonálnak, akkor valószínűbb, hogy a riasztó megszólalt, mintha csak egyikük, vagy egyikük sem. Így mind a JánosTelefonál, mind a MáriaTelefonál szükséges mint szülő.
- Betörés* hozzáadása: ha ismerjük a riasztó állapotát, akkor a Jánostól vagy Máriától jövő telephívások léte vagy hiánya csupán azt jelezhetik, hogy cseng-e a telefonunk, vagy hogy Mária zenéje hangosra van-e állítva, de a betörésről nem nyújtanak további információt. Azaz

$$\mathbf{P}(\text{Betörés} | \text{Riasztás}, \text{JánosTelefonál}, \text{MáriaTelefonál}) = \mathbf{P}(\text{Betörés} | \text{Riasztás})$$

Így csak a *Riasztás* szükséges mint szülő.

- Földrengés* hozzáadása: ha a riasztó bekapcsolt, akkor valószínűbb, hogy bekövetkezett egy földrengés (mivel a riasztó valamiféle földrengészszelő). De ha tudjuk, hogy betörés történt, akkor ez megmagyarázza a riasztást, és a földrengés valószínűsége csak pár nagyobb a normálisnál. Így a *Betörés* és a *Riasztás* is szükséges mint szülő.

A kiadódó hálónak kettővel több éle van, mint az eredetinek a 14.2. ábrán, és hárommal több valószínűség meghatározását igényli. Ami még rosszabb, hogy néhány kapcsolódás megfoghatatlan viszonyt reprezentál, ami nehéz és nem természetes valószínűségi ítéleteket igényel, például a *Földrengés* feltételes valószínűsgének a megbecslését, a *Betörés* és a *Riasztás* feltételekkel. Ez a jelenség igen általános, és kapcsolódik az okozati és diagnosztikai modelleknek a 8. fejezetben bevezetett megkülönböztetéséhez. Ha úgy próbálunk meg egy diagnosztikai modellt megépíteni, hogy a kapcsolatok okozat-ok irányúak (mint például a MáriaTelefonál-tól a *Riasztás*-ig vagy a *Riasztás*-tól a *Betörés*-ig kapcsolat), akkor végül egyébként független okok között kell függést meghatároznunk (és gyakran a különállóan bekövetkező okozatok között is). *Ha ragaszkodunk az okozati modellhez, akkor kevesebb értéket kell megadnunk, és az értékeket általában könnyebb meghatározni.* Orvosi tárgytartományon például Tversky és Kahneman demonstrálta, hogy orvos szakértők szívesebben hoznak valószínűségi ítéleteket okozati, mint diagnosztikai összefüggések esetén (Tversky és Kahneman, 1982).

A 14.3. (b) ábra a változók igazán szerencsétlen sorrendjét mutatja: MáriaTelefonál, JánosTelefonál, Földrengés, Betörés, Riasztás. Ez a háló 31 önálló valószínűség meghatározását igényli – pontosan annyit, amennyi a teljes együttes valószínűség-eloszlás függvény megadása. Azonban fontos felismerni, hogy a három háló bármelyike képes pontosan ugyanannak az együttes valószínűség-eloszlás függvénynek a reprezentálására. A két utóbbi egyszerűen csak nem reprezentálja az összes feltételes függetlenségi relációt, és így rengeteg szükségtelen érték meghatározására kényszerül.



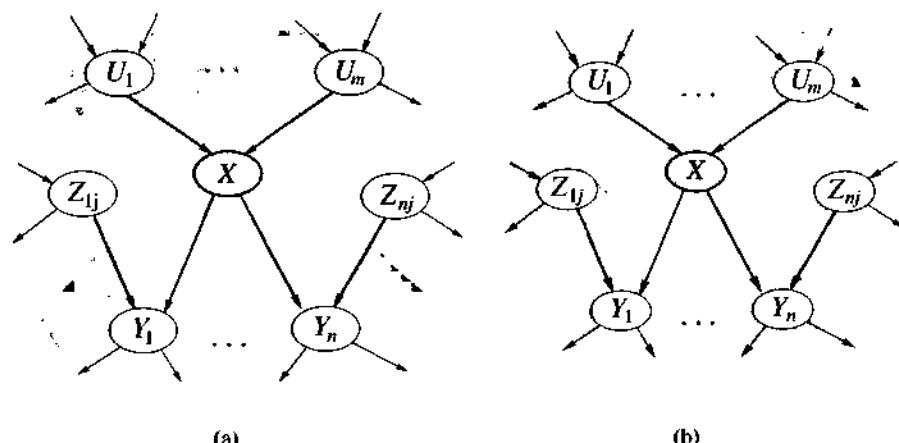
## Feltételes függetlenségi relációk Bayes-hálókban

A Bayes-hálókra egy „numerikus” szemantikát adtunk meg a teljes együttes eloszlás reprezentációjának a szempontjából, mint a (14.1) egyenletben. Ezt a szemantikát alkalmazva a Bayes-hálók konstrukciós módszereinek a származtatásánál azt a következményt kaptuk, hogy egy csomópont feltételesen független az őt megelőzőktől, ha a csomópont

szülfői adottak. Kiderül, hogy más módon is eljárhatunk. Elindulhatunk egy „topológiai” szemantikától, ami a gráf által kódolt feltételes függetlenségi relációkat adja meg, és ezekből származtatjuk a „numerikus” szemantikát. A topológiai szemantikát a következő két egymással ekvivalens meghatározás bármelyike rögzíti:<sup>4</sup>

1. Egy csomópont feltételesen független a nem leszármazottaitól (non-descendants), feltéve, hogy a szülei adottak. Például a 14.2. ábrán *JánosTelefondl* független a *Betörés*-től és a *Földrengés*-től, ismerve a *Riasztás* értékét.
2. Egy csomópont feltételesen független az összes többi csomóponttól a hálózatban, a szülei, gyermekei és gyermekei szüleinek az ismeretében – azaz a **Markov-takarójának** (**Markov blanket**) ismeretében. Például, a *Betörés* független *JánosTelefondl*-től és *MáriaTelefondl*-től, a *Betörés*-t és *Földrengés*-t ismerve.

Ezeket a meghatározásokat mutatja be a 14.4. ábra. Ezekből a feltételes függetlenségi állításokból és az FVT-kból a teljes együttes eloszlást rekonstrálni lehet: így a „numerikus” szemantika és a „topológiai” szemantika ekvivalens.



**14.4. ábra.** (a) Az  $X$  csomópont feltételesen független a nem leszármazottaitól ( $Z_{ij}$ -ktől) a szülei (a szürke területen látható  $U_i$ -k) ismeretében. (b) Az  $X$  csomópont feltételesen független a háló összes többi csomópontjától a Markov-takarójának ismeretében (a szürke terület).

<sup>4</sup> Létezik egy további általános topológiai kritérium a **d-elválasztás** (**d-separation**) annak előírására, hogy a csomópontok egy  $X$  halmaza független-e egy másik  $Y$  halmaztól egy harmadik  $Z$  halmaz feltétel esetén. A kritérium elég bonyolult, és nem szükséges a fejezetben az algoritmuskos származtatásánál, ezért nem tárgyaljuk. A részletek megtalálhatók Russell és Norvig, valamint Pearl munkáiban (Russell és Norvig, 1995; Pearl, 1988). Shachter egy sokkal szemléletesebb módszert ad a d-elválasztások meghatározásához (Shachter, 1998).

## 14.3. FELTÉTELES ELOSZLÁSOK HATÉKONY REPREZENTÁCIÓJA

Még ha a szülők maximális száma,  $k$  meglehetősen kicsi is, egy csomópont feltételes valószínűségi táblájának kitöltése akár  $O(2^k)$  számú értéket és az összes lehetséges feltételes eset figyelembe véve is nagy szakértelmet igényelhet. Valójában azonban az a legrosszabb eset, amikor a kapcsolat a szülők és a gyermek között teljesen önkényes. Általában az ilyen kapcsolatok egy **kanonikus eloszlással (canonical distribution)** írhatók le, amelyek valamelyen szabványos mintát követnek. Ilyen esetekben a teljes tábla megadható a mintázat és esetleg néhány paraméter meghatározásával – sokkal könnyebben, mint exponenciális számú paraméter megadásával.

A legegyszerűbb példát a **determinisztikus csomópontok (deterministic nodes)** szolgáltatják. Egy determinisztikus csomópont értékét a szüleinek az értéke teljesen meghatározza, mindenfajta bizonytalanságtól mentesen. A reláció lehet egy logikai kapcsolat – például ha a szülőcsomópontok azt jelentik, hogy *Kanadai*, *Egyesült Államokbeli* és *Mexikói*, a gyermekcsomópont pedig azt, hogy *Észak-Amerikai*, akkor a közöttük lévő kapcsolat egyszerűen a szülők diszjunkciója. A reláció lehet numerikus is – például ha a szülőcsomópontok egy gépkocsi különböző árai különböző forgalmazóknál, a gyermekcsomópont pedig az az ár, amit egy legolcsóbbat kereső vevő végezetül fizetne, akkor a gyermekcsomópont értéke a szülők értékeinek a minimuma. Másik példa, ha a szülőcsomópontok az egy tóba bejövő vízmennyiségek (folyók, vízlevezetők, csapadék) és az onnan eltávozó vízmennyiségek (folyók, párolgás, elszívárgás), a gyermek pedig a tó szintjének a meg változása, akkor a gyermek értéke a kifolyó és befolyó szülők értékeinek a különbsége.

Bizonytalan relációkat gyakran jellemzhetünk úgynevezett „zajos” logikai relációkkal. A mintapélda erre az úgynevezett **zajos-VAGY (noisy-OR)** reláció, ami a logikai VAGY reláció általánosítása. Ítéletlogikában kijelenthetjük, hogy a *Láz* akkor és csak akkor igaz, ha a *Megfázás* vagy az *Influenza* vagy a *Malária* igaz. A zajos-VAGY modell megengedi bizonytalanságot, hogy egyes szülők okozhatják-e a gyermekek igaz értékét – az okozati kapcsolat a szülő és gyermek között gátolt lehet, és így lehet, hogy a paciens meg van fázva, de nincs láza. A modell két feltevésre épül. Elsőként feltételezi, hogy az összes lehetséges ok fel van sorolva. (Ez nem annyira szigorú megkötés, mint amilyennek tűnik, mivel minden létrehozhatunk egy úgynevezett **szívárgáscsomópontot (leak node)**, ami „vegyes okokat” fed le.) Másodikként felteszi, hogy bármely szülő gátálása független a többi szülő gátálásától: például akármi is gátolja, hogy a *Malária* lázat okozzon, ez független attól, hogy mi gátolja az *Influenzá-t*, hogy lázat okozzon. Ezekkel a feltevésekkel a *Láz* akkor és csak akkor *hamis*, ha az összes *igaz* értékű szülő gátolt, aminek a valószínűsége a gátolás-valószínűségek szorzata. Tételezzük fel, hogy ezek az önálló gátolási valószínűségek a következők:

$$P(\neg\text{láz}|\text{megfázás}, \neg\text{influenza}, \neg\text{malária}) = 0,6$$

$$P(\neg\text{láz}|\neg\text{megfázás}, \text{influenza}, \neg\text{malária}) = 0,21$$

$$P(\neg\text{láz}|\neg\text{megfázás}, \neg\text{influenza}, \text{malária}) = 0,1$$

Ekkor – ennyi információból és a zajos-VAGY feltevésből – a teljes FVT-t fel lehet építeni. A következő táblázat azt mutatja, hogy hogyan:

Megfázás	Influenza	Malária	$P(Láz)$	$P(\neg Láz)$
H	H	H	0.0	1,0
H	H	I	0,9	0,1
H	I	H	0,8	0,2
H	I	I	0,98	$0,02 = 0,2 \times 0,1$
I	H	H	0,4	0,6
I	H	I	0,94	$0,06 = 0,6 \times 0,1$
I	I	H	0,88	$0,12 = 0,6 \times 0,2$
I	I	I	0,988	$0,012 = 0,6 \times 0,2 \times 0,1$

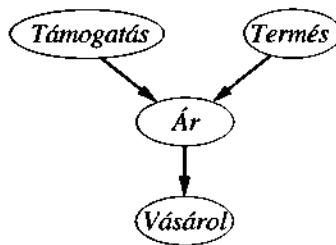
Általánosságban, a zajos logikai relációk, amelyekben egy változó  $k$  számú szülőtől függ,  $O(k)$  paraméterrel írhatók le, a teljes feltételes valószínűség-eloszlás táblázathoz tartozó  $O(2^k)$  helyett. Ez sokkal könnyebbé teszi a becslést és a tanulást. Például a CPSC-háló (Pradhan és társai, 1994) zajos-VAGY- és zajos-MAX-eloszlásokat használ a betegségek és tünetek közötti kapcsolatok modellezésére. 448 csomópont és 906 él esetén ez csak 8254 értéket igényel a 133 931 430 helyett, ami egy teljes FVT-ket használó háló esetén lenne szükséges.

## Bayes-hálók folytonos változókkal

Számos valós problémában fordulnak elő folytonos mennyiségek, mint a magasság, tömeg, hőmérséklet és pénz; valójában a statisztika nagy része olyan valószínűségi változókkal foglalkozik, amelyek értéktartománya folytonos. Definíció szerint, a folytonos változóknak végtelen számú értéke lehet, így lehetetlen feltételes valószínűségeket megadni minden egyes értékre. Egy lehetséges módszer a folytonos változók kezelésére, ha elkerüljük őket **diszkretizálással (discretization)** – azaz felosztjuk a lehetséges értékeket intervallumok adott halmaza szerint. Például, a hőmérsékletet felosztjuk ( $<0^{\circ}\text{C}$ ), ( $0^{\circ}\text{C}-100^{\circ}\text{C}$ ) és ( $>100^{\circ}\text{C}$ ) intervallumokra. A diszkretizálás néha adekvát megoldás, de gyakran eredményezi a pontosság jelentős romlását, valamint nagyon nagy FVT-ket. Egy másik megoldás, ha a valószínűség sűrűségfüggvények alapvető családjából választunk (lásd A) függelék, amelyek véges számú **paraméterrel** megadhatók. Például, a Gauss- (vagy normál) eloszláshoz  $N(\mu, \sigma^2)(x)$  a  $\mu$  átlag és  $\sigma^2$  szórásnégyzet tartozik mint paraméterek.

Egy diszkrét és folytonos változókat is tartalmazó hálót **hibrid Bayes-hálónak (hybrid Bayesian network)** nevezünk. Egy hibrid háló megadásához két újfajta eloszlást kell megadnunk: feltételes eloszlást folytonos változóhoz diszkrét és/vagy folytonos szülők esetén; továbbá feltételes eloszlást diszkrét változókhoz folytonos szülők esetén. Fontoljuk meg a 14.5. ábra egyszerű példáját, amelyben a vásárló valamilyen gyümölcsöt vásárol az ára függvényében, ami viszont a termés mennyiségétől függ és attól, hogy éppen van-e állami támogatás. Az Ár változó folytonos, a szülei pedig folytonosak és diszkrétek; a Vásárol változó diszkrét, és van egy folytonos szülője.

Az Ár változóhoz meg kell adnunk a  $P(\text{Ár}|\text{Termés, Támogatás})$  eloszlást. A diszkrét szülőt explicit felsorolással kezeljük – azaz megadjuk mind a  $P(\text{Ár}|\text{Termés, támogatás})$ , mind a  $P(\text{Ár}|\text{Termés, } \neg\text{támogatás})$  valószínűségeket. A Termés kezeléséhez



14.5. ábra. Egy egyszerű háló diszkrét (*Támogatás* és *Vásárol*) és folytonos (*Termés* és *Ár*) változókkal

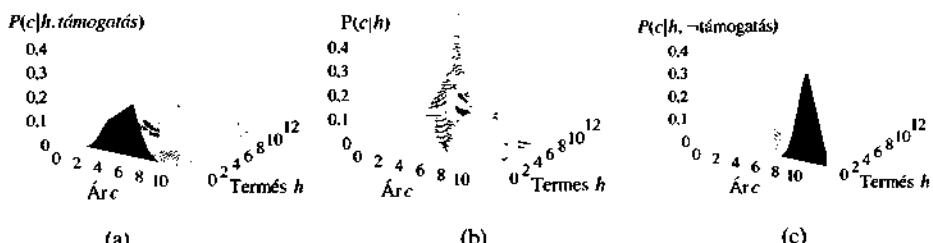
megadjuk, hogy a  $c$  ár feletti eloszlása hogyan függ a  $t$  Termés folytonos értékétől. Máshogy fogalmazva, az ár eloszlásának a paramétereit a termés  $t$  értékének függvényében adjuk meg.

A leggyakoribb választás a **lineáris Gauss-eloszlás** (linear Gaussian), amelyben a gyermekek Gauss-eloszlású, ahol a  $\mu$  várható érték lineárisan változik a szülő értékével, és ahol a  $\delta$  szórás rögzített. Két eloszlásra van szükségünk, a támogatás és a  $\neg$ támogatás esetére különböző paraméterekkel:

$$P(c|t, \text{támogatás}) = N(a_i t + b_i, \sigma_i^2)(c) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{c - (a_i t + b_i)}{\sigma_i} \right)^2}$$

$$P(c|t, \neg\text{támogatás}) = N(a_h t + b_h, \sigma_h^2)(c) = \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{c - (a_h t + b_h)}{\sigma_h} \right)^2}$$

Ebben a példában ekkor, az Ár feltételes eloszlását a lineáris normális eloszlás kiválasztával és az  $a_i, b_i, \sigma_i, a_h, b_h$  és  $\sigma_h$  paraméterekkel adhatjuk meg. A 14.6. (a) és (b) ábra mutatják ezt a két kapcsolatot. Figyeljük meg, hogy mindegyik esetben a meredekség negatív, mivel az ár csökken a kínálat növekedésével. (Természetesen a linearitás feltevése azt is jelenti, hogy az ár egy bizonyos pontnál negatív lesz; a lineáris modell csak akkor ésszerű, ha a termés mennyiséget egy szűk tartományra korlátozzuk.) A 14.6. (c) ábra a  $P(c|h)$  eloszlást mutatja, átlagolva a Támogatás két lehetséges értéke felett feltételezve, hogy mindegyik a priori valószínűsége 0.5. Ez mutatja, hogy még igen egyszerű modellekkel is, elég érdekes eloszlások reprezentálhatók.



14.6. ábra. Az (a) és (b) grafikonok az Ár valószínűség-eloszlását mutatják a Termés függvényében és a Támogatás igaz és hamis értéke mellett. A (c) diagram a  $P(\bar{A}|h)$  eloszlást mutatja, ami a két aleset összegzéseként adódik.

A lineáris normális feltételes eloszlásnak vannak bizonyos speciális tulajdonságai. Egy csak folytonos, lineáris normális feltételes eloszlású változókat tartalmazó háló együttes eloszlása egy többváltozós normális eloszlás az összes változó felett (lásd 14.5. feladat).<sup>5</sup> [A többváltozós normális eloszlás egy felület több mint egy dimenzióban, aminek van egy csúcsa az átlagnál ( $n$  dimenzióban), értéke pedig ettől távolodva minden irányban csökken.] Ha diszkrét változókat adunk a hálóhoz (feltéve, hogy egyetlen diszkrét változó sem gyermeke egy folytonos változónak), a háló egy **feltételes Gauss- (conditional Gaussian)** vagy FG-eloszlást definiál: bármilyen értéket is rendelünk a diszkrét változókhöz, a folytonos változók feletti eloszlás egy többváltozós Gauss-eloszlás lesz.

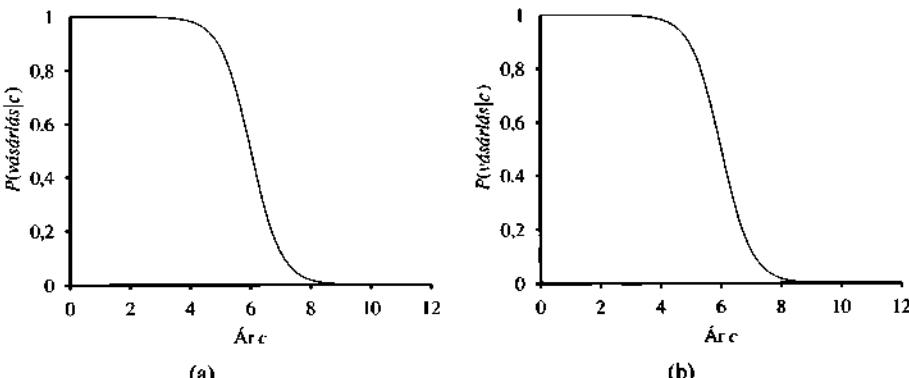
Most a folytonos szülővel rendelkező diszkrét változók eloszlásával kezdünk foglalkozni. Fontoljuk meg például a Vásárol csomópontot a 14.5. ábrán. Ésszerűnek tűnik azt feltételezni, hogy a vásárló vásárol, ha az ár alacsony, nem vásárol, ha magas, a vásárlás valószínűsége pedig folytonosan változik egy közbenső régióban. Márshogyan fogalmazza a feltételes eloszlás hasonló egy „elmosódott” („soft”) küszöbfüggvényhez. Az elmosódott küszöbök létrehozására egy módszer a standard normális eloszlás integráljának a használata:

$$\Phi(x) = \int_{-\infty}^x N(0,1)(x) dx$$

Ekkor a Vásárlás valószínűsége az Ár ismeretében ez lehet

$$P(\text{vásárol} | \text{Ár} = c) = \Phi((-c + \mu)/\sigma)$$

ami azt jelenti, hogy az ár küszöbe  $\mu$  körül van, és a küszöbrégió szélessége arányos  $\delta$ -val, illetve, hogy a vásárlás valószínűsége csökken, ahogy az ár növekszik.



**14.7. ábra.** (a) A Vásárlás valószínűségének probit eloszlása az Ár ismeretében,  $\mu = 6,0$  és  $\sigma = 1,0$  mellett. (b) Logit eloszlás hasonló paraméterekkel.

<sup>5</sup> Következésképpen lineáris normális hálókban a következetés időigénye legrosszabb esetben is csak  $O(n^3)$ , függetlenül a háló topológiájától. A 14.4. alfejezetben látni fogjuk, hogy diszkrét változók hálóiban a következetés NP-nehéz.

Ezt a **probit eloszlást (probit distribution)** a 14.7. (a) ábra illusztrálja. Alakja azzal az érveléssel igazolható, hogy az alapul szolgáló döntési folyamatnál létezik egy pontos küszöb, de ennek pontos helyét egy véletlen normális eloszlású zaj befolyásolja. A probit modell egy alternatívája a **logit eloszlás (logit distribution)**, amely a **sigmoid függvényt (sigmoid function)** használja egy elmosódott küszöb előállításához:

$$P(vásárol | Ar = c) = \frac{1}{1 + \exp\left(-2 \frac{-c + \mu}{\sigma}\right)}$$

Ezt a 14.7. (b) ábra mutatja. A két eloszlás hasonlóan néz ki, de valójában a logit sokkal lassabban tart a határértékekhez. A probit gyakran jobban illeszkedik a valódi helyzetekhez, de a logit esetenként matematikailag könnyebben kezelhető, például széles körben használatos a neurális hálókban (lásd 20. fejezet). Mind a probit, mind a logit általánosítható több folytonos szülőre a szülők értékeinek lineáris kombinációját véve. Többértékű diszkrét gyermekre történő kiterjesztéket a 14.6. feladatban taglalunk.

## 14.4. EGZAKT KÖVETKEZTETÉS BAYES-HÁLÓKBAN

Az alapvető feladat bármely valószínűségi következtető rendszer számára az, hogy ki-számitsa a **célváltozók (query variables)** egy halmazának a posteriori valószínűség-eloszlását egy adott megfigyelt **esemény (event)** esetén – azaz **bizonyítékváltozók (evidence variables)** egy halmazához történő érték-hozzárendelés esetén. A 13. fejezetben bevezetett jelölést fogjuk használni:  $X$  a célváltozót jelöli;  $E$  a bizonyítékváltozók  $E_1, \dots, E_m$  halmazát, e pedig a megfigyelt eseményt;  $Y$  fogja jelölni az (olykor **rejtett változóknak (hidden variables)** nevezett) nem-bizonyítékváltozók  $Y_1, \dots, Y_k$  halmazát. Így a változók teljes halmaza  $X = \{X\} \cup E \cup Y$ . Egy jellemző lekérdezés a  $P(X|e)$  a posteriori eloszlásra irányul.<sup>6</sup>

A riasztós hálóban például megfigyelhetnénk azt az eseményt, ahol *JánosTelefonál = igaz* és *MáriaTelefonál = igaz*. Ekkor megkérdezhetnénk mondjuk a betörés megtörténtének a valószínűségét:

$$P(\text{Betörés} | \text{JánosTelefonál} = \text{igaz}, \text{MáriaTelefonál} = \text{igaz}) = \langle 0,284, 0,716 \rangle$$

Ebben a fejezetben az a posteriori valószínűségek kiszámítására szolgáló egzakt algoritmusokat tárgyalunk, és átgondoljuk ennek a feladatnak a komplexitását. Kiderül, hogy általános esetben ez kivitelezhetetlen, ezért a 14.5. alfejezet közelítő következtetési módszereket ismertet.

<sup>6</sup> Feltesszük, hogy a célváltozó nincs a bizonyítékváltozók között; amíg  $X$  a posteriori eloszlása a megfigyelt értékre 1 valószínűséget ad. Az egyszerűség kedvéért azt is feltettük, hogy a kérdés egyetlen változóra irányul. Az algoritmusaink könnyen kiterjeszthetők több változó együttesének lekérdezésére.

## Következtetés felsorolással

A 13. fejezet elmagyarázta, hogy bármely feltételes valószínűség kiszámítható a teljes együttes eloszlás tagjainak összegzésével. Pontosabban, egy  $P(X|e)$  lekérdezés megválaszolható a (13.6) egyenlet felhasználásával, amit a jobb követhetőség kedvéért itt megismétlik:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

Továbbá, mint ismeretes, a Bayes-háló a teljes együttes eloszlás egy teljes reprezentációját nyújtja. Még pontosabban, a (14.1) egyenlet azt mutatja, hogy az együttes eloszlás  $P(x, e, y)$  tagjai felírhatók a hálóból származó feltételes valószínűségek szorzataiként. Ezért *egy lekérdezés megválaszolható a Bayes-háló felhasználásával, kiszámítva a hálóból származó feltételes valószínűségek szorzatainak az összegét.*



A 13.4. ábrán a FELSOROL-EGYÜTTES-KÉRDEZÉS algoritmust adtuk meg a teljes együttes eloszlásból felsorolással történő következtetésre. Az algoritmus bemenetként fogadja a  $P$  teljes együttes eloszlást, és értékeket néz meg benne. Egyszerű módosítással elérhető, hogy az algoritmus egy  $bn$  Bayes-hálót fogad bemenetként, és az együttes bejegyzéseket a  $bn$  megfelelő FVT-bejegyzéseinek összeszorzásával „nézi meg”.

Vegyük a  $P(\text{Betörés}|\text{JánosTelefonál} = \text{igaz}, \text{MáriaTelefonál} = \text{igaz})$  lekérdezést. A rejttett változók ennél a kérdésnél a Földrengés és a Riasztás. A (13.6) egyenletből, a kifejezések rövidítése miatt a kezdőbetűket használva, azt kapjuk, hogy<sup>7</sup>

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, e, a, j, m)$$

A Bayes-hálók szemantikája – (14.1) egyenlet – pedig az FVT-bejegyzések felhasználásával egy kifejezést ad. Az egyszerűség kedvéért ezt csak a  $\text{Betörés} = \text{igaz}$  esetére adjuk meg:

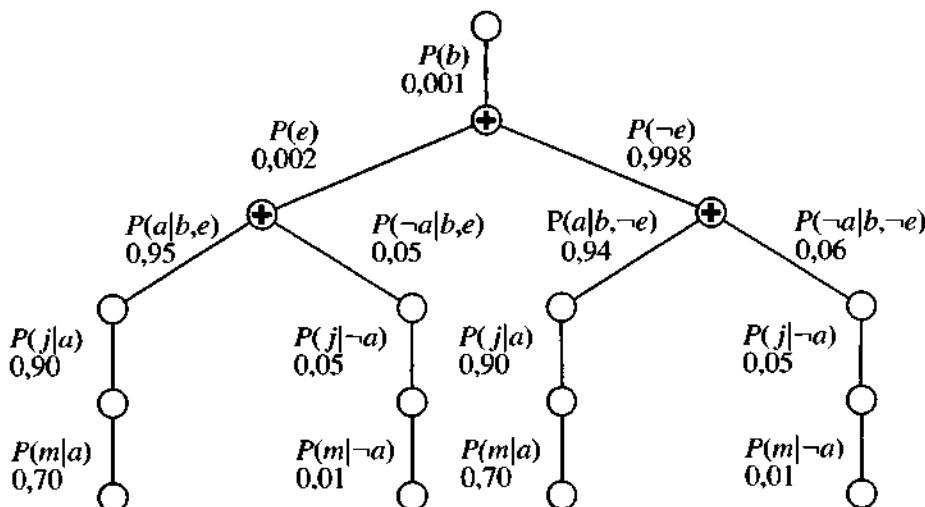
$$P(b|j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a|b, e)P(j|a)P(m|a)$$

Ennek a kifejezésnek a kiszámításához négy tagot kell összeadnunk, amelyek minden egyikötöt szám összeszorzásával kapjuk. Legrosszabb esetben, amikor majdnem minden változó felett összegezni kell, az algoritmus komplexitása egy  $n$  bináris változós háló esetén  $O(n2^n)$ .

A következő egyszerű megfigyeléssel egy javításhoz juthatunk: a  $P(b)$  tag állandó, és ki lehet vinni az  $a$  és  $e$  feletti összegzések elő, a  $P(e)$  tag pedig kivihető az  $a$  feletti összegzés elő. Így azt kapjuk, hogy

$$P(b|j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b, e)P(j|a)P(m|a) \quad (14.3)$$

<sup>7</sup> Egy olyan kifejezés mint a  $\sum_e P(a, e)$  a  $P(A = a, E = e)$ -nek az összegzését jelenti minden lehetséges  $e$  értékre. Ez többetelműséget rejti magában, mivel a  $P(e)$  jelölést használják mind a  $P(E = \text{igaz})$ , mind a  $P(E = e)$  jelentés kifejezésére, azonban a szándékolt jelentésnek a szövegkörnyezetből ki kell derülnie; egy összegzés esetében az utóbbiról van szó.



**14.8. ábra.** A (14.3) egyenletben szereplő kifejezés struktúrája. A kiértékelés fentről lefelé halad, összeszorozva az értékeket az egyes útvonalak mentén, és összegezve a „+” csomópontoknál. Vegyük észre, a  $j$ -hez és az  $m$ -hez tartozó útvonalak megismétlődését.

Ez a kifejezés egy ciklussal értékelhető ki, sorban végighaladva a változókon, miközben összeszorozzuk az FVT-bejegyzéseket. minden összegzésnél, a változó lehetséges értékei szerint is iterálnunk kell. A számítás struktúráját a 14.8. ábra mutatja. Felhasználva a 14.2. ábra értékeit, azt kapjuk hogy  $P(b|j,m) = \alpha \times 0,00059224$ . A  $\neg b$ -hez tartozó számítást átfutva az  $\alpha \times 0,0014919$ -et eredményez; így

```

function FELSOROLÁS-KÉRDEZÉS( $X, e, bn$ ) returns egy  $X$  feletti eloszlás
  inputs:  $X$ , a lekérdezés változója
   $e$ , az  $E$  változók megfigyelt értékei
   $bn$ , egy valósínűségi háló  $\{X\} \cup E \cup Y /*Y = rejtett változók*/$ 

```

```

 $Q(X) \leftarrow$  egy eloszlás  $X$  felett, kezdetben üres
for each  $x_i$  értékére  $X$ -nek do
  terjeszd ki  $e$ -t  $X$ -nek az  $x_i$  értékével
   $Q(x_i) \leftarrow$  FELSOROL-MINDET(VÁLTOZÓ[ $bn$ ],  $e$ )
return NORMALIZÁL( $Q(X)$ )
  
```

```

function FELSOROL-MINDET(változók,  $e$ ) returns egy valós szám
  if ÜRES?(változók) then return 1.0
   $Y \leftarrow$  ELSÓ(változók)
  if  $Y$  értéke  $y$   $e$ -ben
    then return  $P(y|szülök(Y)) \times$  FELSOROL-MINDET(MARADÉK(változók),  $e$ )
  else return  $\sum_y P(y|szülök(Y)) \times$  FELSOROL-MINDET(MARADÉK(változók),  $e_y$ )
    ahol  $e_y$  ki lett terjesztve  $Y = y$ -nal
  
```

**14.9. ábra.** A felsoroló algoritmus Bayes-hálós lekérdezések megválaszolására

$$P(B|j,m) = \alpha \langle 0,00059224, 0,0014919 \rangle \approx \langle 0,284, 0,716 \rangle$$

Azaz a betörés valószínűsége, ha minden szomszéd telefonál körülbelül 28%.

A 14.8. ábra a (14.3) egyenlet kiértékelési folyamatát mutatja, kifejezés fa alakban. A FELSOROL-KÉRDEZ algoritmus a 14.9. ábrán ilyen fákat mélységi rekurzióval érték el. Így a FELSOROL-KÉRDEZ tárkomplexitása csak lineáris a változók számában – az algoritmus így hatékonyan, anélkül összegez a teljes együttes eloszlás felett, hogy azt explicit módon megkonstruálná. Sajnos az algoritmus időkomplexitása egy  $n$  bináris változós hálónál minden esetben  $O(2^n)$  – ami jobb, mint az egyszerű megközelítéshez tartozó  $O(n2^n)$ , de még mindig elég nyomasztó. Egy figyelemre méltó dolog a 14.8. ábra fájával kapcsolatban, hogy nyilvánvalóvá teszi a megismételt kifejezéseket, amelyeket az algoritmus kiértékel. A  $P(j|a) P(m|a)$  és a  $P(j|\neg a) P(m|\neg a)$  szorzatok kétszer kerülnek kiszámításra,  $e$  minden értékénél. A következő fejezet egy általános módszert ír le, ami elkerüli az ilyen felesleges újraszámítások elvégzését.

## A változó eliminációs algoritmus

A felsoroló algoritmus lényegesen javítható a 14.8. ábra által szemléltetett ismétlődő számítások kiküszöbölésével. Az ötlet egyszerű: egyszer végezzük el a számítást, és őrizzük meg az eredményeket későbbi használatra. Ez egyfajta dinamikus programozás. Ennek a megközelítésnek több változata is létezik; mi a legegyszerűbbet, a **változó eliminációs (variable elimination)** algoritmust mutatjuk be. A változó eliminálás a (14.3) egyenlet típusú kifejezéseket *jobbról balra* (azaz a 14.8. ábrán *lentről felfelé*) sorrendben értékeli ki. A köztes eredményeket eltároljuk, és bármely változó feletti összegzés a kifejezésnek csak azon része felett történik meg, amely függ ettől a változótól.

Szemléltessük ezt a folyamatot a betörés háló esetén. Értékeljük ki a következő kifejezést:

$$P(B|j,m) = \alpha \underbrace{P(B)}_B \sum_e \underbrace{P(e)}_E \sum_a \underbrace{P(a|B,e)}_A \underbrace{P(j|a)}_J \underbrace{P(m|a)}_M$$

Vegyük észre, hogy a kifejezés minden részét megjelöltük a kapcsolódó változó nevével; ezeket **tényezőknek (factors)** nevezzük. A lépések a következők:

- Az  $M$  tényező,  $P(m|a)$  nem igényel  $M$  feletti összegzést (mivel  $M$  értéke már rögzített). Eltároljuk a valószínűséget  $a$  minden értékére egy kételemű vektorban,

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix}$$

(Az  $\mathbf{f}_M$  azt jelöli, hogy  $\mathbf{f}$  létrehozásában  $M$ -et felhasználtuk.)

- Hasonlóan, tároljuk a  $J$ -hez tartozó tényezőt egy kételemű  $\mathbf{f}_J(A)$  vektorban.
- Az  $A$ -hoz tartozó tényező,  $P(a|B, e)$ , ami egy  $2 \times 2 \times 2$ -es mátrix lesz,  $\mathbf{f}_A(A, B, E)$ .
- Most ki kell összegezni  $A$ -t ennek a három tényezőnek a szorzatából. Ez egy  $2 \times 2$ -es mátrixot eredményez, aminek indexei már csak  $B$  és  $E$  felett futnak. A mátrix nevében  $A$ -nál egy felülvonással jelezzük, hogy  $A$  már ki lett összegezve:

$$\begin{aligned}\mathbf{f}_{\bar{A}JM}(B, E) &= \sum_a \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &= \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &\quad + \mathbf{f}_A(\neg a, B, E) \times \mathbf{f}_J(\neg a) \times \mathbf{f}_M(\neg a)\end{aligned}$$

Az itt használt szorzást **pontonkénti szorzásnak** (pointwise product) nevezik, és hamarosan ismertetjük.

- Hasonló módon dolgozzuk fel  $E$ -t: kiösszegezzük  $E$ -t  $\mathbf{f}_E(E)$  és  $\mathbf{f}_{\bar{A}JM}(B, E)$  szorzatából:

$$\mathbf{f}_{\bar{E}\bar{A}JM}(B) = \mathbf{f}_E(e) \times \mathbf{f}_{\bar{A}JM}(B, e) + \mathbf{f}_E(\neg e) \times \mathbf{f}_{\bar{A}JM}(B, \neg e).$$

- Most pedig kiszámíthatjuk az eredményt egyszerűen összeszorozva  $B$  tényezőjét (azaz  $\mathbf{f}_B(B) = \mathbf{P}(B)$ -t) az  $\mathbf{f}_{\bar{E}\bar{A}JM}(B)$  kiadódott mátrixszal:

$$\mathbf{P}(B|j, m) = \alpha \mathbf{f}_B(B) \times \mathbf{f}_{\bar{E}\bar{A}JM}(B)$$

A 14.7. (a) feladatban azt kérjük, hogy ellenőrizze, helyes eredményt ad-e ez a folyamat.

A lépések sorozatát megvizsgálva láthatjuk, hogy két alapvető művelet szükséges: tényezőpárok pontonkénti összeszorzása és egy változó kiösszegzése tényezők szorzatából.

A pontonkénti szorzás nem mátrixszorzás és nem is egy elemenkénti szorzás. Két,  $\mathbf{f}_1$  és  $\mathbf{f}_2$  tényezőnek a pontonkénti szorzata egy új  $\mathbf{f}$  tényezőt eredményez, amelynek változít az  $\mathbf{f}_1$  és  $\mathbf{f}_2$  változóinak az *uniójá* adja. Tételezzük fel, hogy a két tényezőben az  $Y_1, \dots, Y_k$  változók közösek. Ekkor azt kapjuk, hogy

$$\mathbf{f}(X_1 \dots X_j, Y_1 \dots Y_k, Z_1 \dots Z_l) = \mathbf{f}_1(X_1 \dots X_j, Y_1 \dots Y_k) \mathbf{f}_2(Y_1 \dots Y_k, Z_1 \dots Z_l)$$

Ha minden változó bináris, akkor  $\mathbf{f}_1$ -nek és  $\mathbf{f}_2$ -nek  $2^{j+k}$  és  $2^{k+l}$  eleme van, pontonkénti szorzatuknak pedig  $2^{j+k+l+1}$ . Például az  $\mathbf{f}_1(A, B)$  és az  $\mathbf{f}_2(B, C)$  tényezők esetén a lentebb mutatott valószínűség-eloszlással, az  $\mathbf{f}_1 \times \mathbf{f}_2$  pontonkénti szorzata  $\mathbf{f}_3(A, B, C)$ -vel adott:

$A$	$B$	$\mathbf{f}_1(A, B)$	$B$	$C$	$\mathbf{f}_2(B, C)$	$A$	$B$	$C$	$\mathbf{f}_3(A, B, C)$
$I$	$I$	0,3	$I$	$I$	0,2	$I$	$I$	$I$	$0,3 \times 0,2$
$I$	$H$	0,7	$I$	$H$	0,8	$I$	$I$	$H$	$0,3 \times 0,8$
$H$	$I$	0,9	$H$	$I$	0,6	$I$	$H$	$I$	$0,7 \times 0,6$
$H$	$H$	0,1	$H$	$H$	0,4	$I$	$H$	$H$	$0,7 \times 0,4$
						$H$	$I$	$I$	$0,9 \times 0,2$
						$H$	$I$	$H$	$0,9 \times 0,8$
						$H$	$H$	$I$	$0,1 \times 0,6$
						$H$	$H$	$H$	$0,1 \times 0,4$

Egy változó kiösszegzése tényezők szorzataiból szintén egyértelmű számítás. Az egyetlen észreveendő trükk az, hogy bármely tényező, ami *nen* függ a kiösszegzendő változótól, az az összegzésen kívülre mozgatható. Például:

$$\sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) \times \mathbf{f}_J(A) \times \mathbf{f}_M(A) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e)$$

Most a pontonkénti szorzat kerül kiszámításra az összegzésen belül, a változót pedig kiösszegezzük a kiadódó mátrixból:

$$\mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \mathbf{f}_{\overline{E}, A}(A, B)$$

Vegyük észre, hogy mátrixokat *nem* szorzunk, ameddig nem kell kiösszegeznünk egy változót a kiadódó szorzból. Annál a pontnál csak azokat a mátrixokat szorozzuk össze, amelyek magukban foglalják a kiösszegzendő változót. A pontonkénti szorzás és a kiösszegzés eljárások megléte esetén maga a változó eliminálás algoritmus, amint azt a 14. 10. ábra mutatja, igen egyszerűen felírható.

Nézzünk még egy lekérdezést:  $P(\text{JánosTelefonál} | \text{Betörés} = \text{igaz})$ . Szokásos módon, az első lépés az egymásba ágyazott összegzések felírása:

$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b, e) P(J|a) \sum_m P(m|a)$$

Ha ezt a kifejezést jobbról balra kiértékeljük, valami érdekeset veszünk észre:  $\sum_m P(m|a)$  definíció szerint 1-gel egyenlő. Így ezt eleve kihagyhattuk; a változó  $M$  a kérdezre *irrelevant*. Máshogy fogalmazva, a  $P(\text{JánosTelefonál} | \text{Betörés} = \text{igaz})$  lekérdezés eredményét nem változtatja meg a *MáriaTelefonál* eltávolítása a hálóból. Általában, bár mely levélcsomópontot eltávolíthatunk, ami nem célváltozó vagy bizonyítékváltozó. Eltávolítás után lehetnek újabb levélcsomópontok, amelyek szintén irrelevantak lehetnek.

 Ezt az eljárást folytatva végül szükségszerűen arra jutunk, hogy *minden* változó, ami *nem* őse a célváltozónak vagy *egy* bizonyítékváltozónak, *irrelevant* a lekérdezésre. A változó elimináló algoritmus ezért az összes ilyen változót eltávolíthatja a lekérdezés kiértékelése előtt.

```

function ELIMINÁLÁS-KÉRDEZÉS( $X$ ,  $e$ ,  $bn$ ) returns egy  $X$  feletti eloszlás
  inputs:  $X$ , a lekérdezés változója
     $e$ , a bizonyíték mint esemény
     $bn$ , egy  $\mathbf{P}(X_1, \dots, X_n)$  együttes eloszlást megadó Bayes-háló

   $tényezők \leftarrow []$ ;  $változók \leftarrow \text{FORDÍT(VÁLTOZÓK}[bn])$ 
  for each változó in változók do
     $tényezők \leftarrow [\text{TÉNYEZŐ-LÉTREHOZÓ}(változó, e)]változók$ 
    if változó rejtett then  $tényezők \leftarrow \text{KIÖSSZEGEZ}(változó, tényezők)$ 
  return NORMALIZÁL(PONTONKÉNT-SZORZAT(tényezők))

```

14.10. ábra. A változó eliminálás algoritmus Bayes-hálós lekérdezések megválaszolására

## Az egzakt következtetés komplexitása

Megmutattuk, hogy a változóeltávolítás hatékonyabb, mint a felsorolás, mivel elkerüli a számítások megismétlését (ahogyan az irrelevant változókat is kiejt). A változó eliminálás idő- és tárigényét az algoritmus működése alatt létrejövő legnagyobb tényező mérete befolyásolja a legerőteljesebbben. Ezt viszont a változók eliminálásának sorrendje és a háló struktúrája határozza meg.

A 14.2. ábra betörős hálója a hálóknak azon családjához tartozik, ahol a háló bármely két csomópontja között legfeljebb egyetlen irányítatlan út létezik. Ezeket *egyszervesen*

**összekötött (singly connected)** hálóknak vagy **polifáknak (polytree)** nevezük, amelyeknek van egy különösen kellemes tulajdonságuk: az egzakt következetés idő- és tárkomplexitása a polifákban a háló méretében lineáris. Itt a méret az FVT-bejegyzések számával van definiálva; ha minden egyes csomópont szüleinek a száma egy konstans-sal korlátos, akkor a komplexitás a csomópontok számában is lineáris. Ezek az eredmények bármely – a háló topológiai sorrendjével konzisztens – sorrendezés esetén fennállnak (lásd 14.7. feladat).

A többszörösen összekötött (multiply connected) hálóban (lásd például 14.11. (a) ábra) a változó eliminálás legrosszabb esetben exponenciális idő- és tárkomplexitású lehet, még akkor is, ha a csomópontonkénti szülők száma korlátos. Ez nem meglepő, figyelembe véve, hogy a Bayes-hálókban való következetés NP-nehéz, mivel ez speciális alesetként tartalmazza az ítéletlogikai következetést is. Valójában megmutatható (lásd 14.8. feladat), hogy a probléma ugyanannyira nehéz, mint kiszámítani egy ítélet-logikai formula esetén a formulát kielégítő értékadások számát. Ez azt jelenti, hogy ez #P-nehéz („számossg-P-nehéz”), azaz szigorúan nehezebb, mint az NP-teljes problémák.

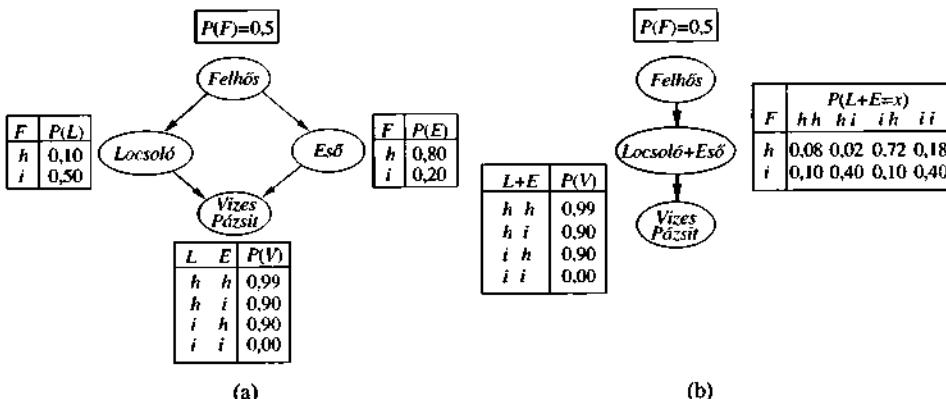
Szoros kapcsolat áll fenn a Bayes-hálókban történő következetés komplexitása és a kényszerkielégítési problémák (CSP) komplexitása között. Amint az 5. fejezetben tárgyaltuk, egy diszkrét CSP megoldásának nehézsége ahhoz kapcsolódik, hogy a kényszergráf mennyire „faszerű”. Olyan mértékek, mint a **hiperfeszélesség (hypertree width)**, amelyek a CSP megoldási komplexitását határolják, közvetlenül alkalmazhatók Bayes-hálóknál is. Ezenfelül a változó eliminálás algoritmusára általánosítható úgy, hogy kényszerkielégítési problémákat is és Bayes-hálókat is megoldjon.

## Csoportosító algoritmusok

A változó eliminálás algoritmusára egyszerű és hatékony egyedi lekérdezések megválasztására. Azonban ha a háló összes változójának az a posteriori eloszlását szeretnénk kiszámítani, akkor kevésbé hatékony is lehet. Például egy polifa hálóban  $O(n)$  egyenként  $O(n)$  költségű lekérdezést kell kiadni, összességében  $O(n^2)$  időköltséggel. **Csoportosító (clustering)** eljárásokkal (amit egyesítési fa – **join tree** – algoritmusnak is neveznek) ez az idő  $O(n)$ -re csökkenhető. Emiatt ezeket az algoritmusokat gyakran használják kereskedelmi Bayes-hálós eszközökben.

A csoportosítás alapötlete, hogy a háló önálló csomópontjait egyesítjük, klasztercsomópontokat formálva úgy, hogy a kiadódó háló polifa legyen. Például a 14.11. (a) ábrán mutatott, többszörösen összekötött háló egy polifává konvertálható a Locsoló és az Eső csomópontok egy **Locsoló+Eső**-nek nevezett **klasztercsomópontba (cluster node)** való összevonásával, ahogy azt a 14.11. (b) ábra mutatja. A két bináris csomópontot egyetlen megacsomópont váltotta fel, ami négy lehetséges értéket vesz fel: *hh*, *hi*, *ih*, *ii*. A megacsomópontnak csak egy szülője van, a bináris *Felhős* változó, így két feltételes eset létezik.

A háló polifává konvertálása után, egy speciális célú következetési algoritmust alkalmazunk. Alapvetően az algoritmus egyfajta kényszerterjesztés (lásd 5. fejezet), ahol a kényszerek biztosítják, hogy a szomszédos csoportokban megegyeznek a közös változók a posteriori eloszlásai. Gondos nyilvántartással ez az algoritmus képes  $O(n)$  időben – ahol  $n$  most a módosított háló mérete – kiszámolni a háló összes nem bizonyíték csomó-



14.11. ábra. (a) Egy többszörösen összekötött háló feltételes valószínűségi táblákkal. (b) Egy többszörösen összekötött háló csoportosított ekvivalense.

pontjának a posteriori eloszlását. Azonban a probléma NP-nehéz volta nem tűnt el: ha a háló exponenciális idő- és tárigényű a változó eliminálás esetén, akkor a csoportosított hálóhoz tartozó FVT-k megkonstruálása exponenciális idő- és tárigényű lesz.

## 14.5. KÖZELÍTŐ KÖVETKEZTETÉS BAYES-HÁLÓKBAN

Az egzakt következtetés nagy, többszörösen összekötött hálókban való kezelhetetlensége miatt, elengedhetetlen közelítő következtetési módszerek átgondolása. Ez a fejezet véletlen mintavételi, Monte Carlo-módszereknek is nevezett, eljárásokat ír le, amelyek közelítő válaszokat nyújtanak, ahol a pontosság a generált minták számától függ.

Az utóbbi években a Monte Carlo-algoritmusok széles körben elterjedtek a számítástudományban olyan mennyiségek megbecslésére, amelyeket nehéz egzakt módon kiszámolni. Például a 4. fejezetben leírt szimulált lehűtés algoritmus egy Monte Carlo-eljárás optimalizációs problémára. Ebben a fejezetben a mintavételezésnek az a posteriori valószínűségek kiszámításában történő alkalmazásában vagyunk érdekeitek. Algoritmusok két nagy családját írjuk le: a közvetlen mintavételezést és a Markovláncos mintavételezést. Két másik megközelítést – a variációs módszert és a „hurkos” („loopy”) terjesztést – a fejezet végén megjegyzésekben említiük meg.

### Közvetlen mintavételezési módszerek

Az alapelem minden mintavételi algoritmusban a minták generálása egy ismert valószínűség-eloszlásból. Például egy szabályos érme felfogható egy *Érme* valószínűségi változónak (*fej*, *irás*) értékekkel és  $P(\text{Érme}) = \langle 0,5, 0,5 \rangle$  a priori eloszlással. A mintavétel ebből az eloszlásból pontosan megfelel egy érme dobálásának: 0,5 valószínűséggel *fej*-et ad, és 0,5 valószínűséggel *irás*-t. Ha rendelkezésre áll a  $[0, 1]$  tartományba eső véletlen számoknak egy forrása, akkor bármely egyváltozós eloszlásból egyszerű dolog mintavételezni (lásd 14.9. feladat).

A véletlen mintavételezési folyamat legegyszerűbb fajtája Bayes-hálók esetén a háloból generál olyan eseményeket, amelyekhez nem kapcsolódik bizonyíték. Az ötlet az, hogy mintavételezzünk minden változót egymás után, topológiai sorrendben. A valószínűség-eloszlás, amiből az értéket mintavételezzük, feltételesen függ a változó születhez már hozzárendelt értékektől. Ez az algoritmus látható a 14.12. ábrán. A működését a 14.11. (a) ábrán látható hálón szemléltetjük, feltételezve egy [Fehős. Locsoló. Eső, VizesPázsit] sorrendezést:

1. Sorsoljunk a  $P(Fehős) = (0,5, 0,5)$  eloszlásból; tegyük fel, hogy *igaz*-at kapunk.
2. Sorsoljunk a  $P(Locsoló|Fehős = \text{igaz}) = (0,1, 0,9)$  eloszlásból; tegyük fel, hogy *hamis*-at kapunk.
3. Sorsoljunk a  $P(Eső|Fehős = \text{igaz}) = (0,8, 0,2)$  eloszlásból; tegyük fel, hogy *igaz*-at kapunk.
4. Sorsoljunk a  $P(VizesPázsit|Locsoló = \text{hamis}, Fehős = \text{igaz}) = (0,9, 0,1)$  eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

Ebben az esetben a PRIOR-MINTA a következő eseményt adja [*igaz*, *hamis*, *igaz*, *igaz*].

Könnyen látható, hogy a PRIOR-MINTA a háló által meghatározott a priori együttes eloszlásból generál mintákat. Először, legyen  $S_{PS}(x_1, \dots, x_n)$  annak a valószínűsége, hogy egy adott eseményt legenerál a PRIOR-MINTA algoritmus. *Pusztán a mintavételi folyamat alapján* felírható, hogy

$$S_{PS}(x_1 \dots x_n) = \prod_{i=1}^n P(x_i | \text{szülők}(X_i))$$

mivel minden mintavételi lépés a szülő értékektől függ.

```
function PRIOR-MINTA(bn) returns egy esemény a bn által definiált a priori eloszlásból
inputs: bn, egy  $P(X_1, \dots, X_n)$  együttes eloszlást megadó Bayes-háló
    x  $\leftarrow$  egy esemény n elemmel
    for i = 1 to n do
         $x_i \leftarrow$  egy véletlen minta  $P(X_i | \text{szülők}(X_i))$  szerint
    return x
```

14.12. ábra. Egy mintavételező algoritmus, amely eseményeket generál egy Bayes-hálóból

Ez a kifejezés ismerős, mivel – ahogy ezt a (14.1) egyenlet kimondja – ez egyben a valószínűsége is az együttes eloszlás Bayes-hálós reprezentációja szerinti eseménynek is. Arra jutottunk tehát, hogy

$$S_{PS}(x_1 \dots x_n) = P(x_1 \dots x_n)$$

Ez az egyszerű tény nagyon könnyűvé teszi lekérdezések – minták felhasználásával történő – megválaszolását.

Bármilyen mintavételi algoritmusban a válasz kiszámítása a generálás során előálló minták megszámlálása alapján történik. Tételezzük fel, hogy *N* teljes mintánk van, és

jelölje  $N(x_1, \dots, x_n)$  az  $x_1, \dots, x_n$  esemény gyakoriságát. Azt várjuk, hogy ez a gyakoriság határértékben konvergáljon a várható értékhez a mintavételei valószínűség szerint:

$$\lim_{N \rightarrow \infty} \frac{N_{PS}(x_1, \dots, x_n)}{N} = S_{PS}(x_1, \dots, x_n) = P(x_1, \dots, x_n) \quad (14.4)$$

Például gondoljuk meg az előbb generált eseményt: [igaz, hamis, igaz, igaz]. Ennek az eseménynek a mintavételei valószínűsége:

$$S_{PS}(\text{igaz}, \text{hamis}, \text{igaz}, \text{igaz}) = 0,5 \times 0,9 \times 0,8 \times 0,9 = 0,324$$

Így,  $N$  nagy értékeinél azt várjuk, hogy a minták 32,4%-a ez az esemény legyen.

A következőkben, amikor a közelítő egyenlőség jelet ( $\approx$ ) használjuk, pontosan ilyen értelemben használjuk – azaz a becsült valószínűség egzakttá válik a nagy mintaszámú határesetben. Az ilyen becsléseket **konzisztenstek** (consistent) nevezzük. Például bármely részlegesen meghatározott  $x_1, \dots, x_m$  esemény valószínűségének egy konzisztenst becslése a következőképpen kapható meg, ha  $m \leq n$ :

$$P(x_1, \dots, x_m) \approx N_{PS}(x_1, \dots, x_m)/N \quad (14.5)$$

Azaz, az esemény valószínűsége megbecsülhető a mintavételei folyamat által generált összes teljes esemény azon hánnyadával, amelyek illeszkednek a részlegesen meghatározott eseményre. Például ha generálunk 1000 mintát a locsolós hálóból, és 511-ben közülük az  $Eső = \text{igaz}$ , akkor az eső becsült valószínűsége  $\hat{P}(Eső = \text{igaz}) 0,511$ .

### Elutasító mintavételezés Bayes-hálókban

**Az elutasító mintavételezés (rejection sampling)** általános módszer minták előállítására egy nehezen mintavételezhető eloszlásból, felhasználva egy könnyen mintavételezhető eloszlást. Legegyszerűbb formájában feltételes valószínűségek kiszámítására – azaz  $P(X|e)$  meghatározására használható fel. Az ELUTASÍTÓ-MINTAVÉTELEZÉS algoritmusa a 14.13. ábrán látható. Először a háló által megadott a priori eloszlásból generál mintákat, majd elutasítja azokat, amelyek nem illeszkednek a bizonyítékhöz. Végül, a  $\hat{P}(X = x|e)$  becslés megkapható az  $X = x$  előfordulásainak megszámolásával a megmaradt mintában.

Legyen  $\hat{P}(X|e)$  az algoritmus által kiadott becsült eloszlás. Az algoritmus definíciója szerint fennáll, hogy

$$\hat{P}(X|e) = \alpha N_{PS}(X, e) = \frac{N_{PS}(X, e)}{N_{PS}(e)}$$

A 14.5 egyenletből kapjuk, hogy

$$\hat{P}(X|e) \approx \frac{P(X, e)}{P(e)} = P(X|e)$$

Azaz az elutasító mintavételezés az igazi valószínűség konzisztenst becslését adja.

Folytatva a 14.11. (a) ábra szerinti példánkat, tételezzük fel, hogy meg szeretnénk becsülni a  $P(Eső|Locsoló = \text{igaz})$ -t, 100 minta felhasználásával. A 100 általunk generált mintából, tegyük fel, hogy 73-ban teljesül, hogy  $Locsoló = \text{hamis}$ , és így elutasított,

```

function ELUTASÍTÓ-MINTAVÉTELEZÉS( $X, e, bn, N$ ) returns  $P(X|e)$  egy becslése
  inputs:  $X$ , a lekérdezés változója
     $e$ , a bizonyíték mint esemény
     $bn$ , egy Bayes-háló
     $N$ , a generálandó minták száma
  local variables:  $N, X$  értékeihez tartozó számlálók vektora, kezdetben nulla
  for  $j = 1$  to  $N$  do
     $x \leftarrow$  PRIOR-MINTA( $bn$ )
    if  $x$  konzisztens  $e$ -vel then
       $N[x] \leftarrow N[x] + 1$ , ahol  $x$  az  $X$  értéke  $x$ -ben
  return NORMALIZÁL( $N[X]$ )

```

**14.13. ábra.** Az elutasító mintavétel algoritmusá, ami Bayes-hálós lekérdezéseket válaszol meg bizonyítékok esetén

míg 27-ben fennáll, hogy  $Locsoló = igaz$ ; a 27-ből 8-ban  $Eső = igaz$ , 19-ben pedig  $Eső = hamis$ . Így a

$$P(Eső|Locsoló = igaz) \approx \text{NORMALIZÁL}(\langle 8, 19 \rangle) = \langle 0,296, 0,704 \rangle$$

A helyes érték  $\langle 0,3, 0,7 \rangle$ . A begyűjtött minták szaporodásával a becslés konvergálni fog a helyes értékhez. A becslés szórása az egyes valószínűségeknél arányos lesz  $1/\sqrt{n}$ -nel, ahol  $n$  a becsléshez felhasznált minták száma.

Az elutasító mintavétel legnagyobb hibája, hogy nagyon sok mintát utasít el! Az  $e$  bizonyítékkal konzisztens minták aránya exponenciálisan egyre kevesebb, ahogy a bizonyítékváltozók száma nő, így az eljárás egyszerűen használhatatlan komplex problémákban.

Vegyük észre, hogy az elutasító mintavétel nagyon hasonló a feltételes valószínűségek becsléséhez, ha közvetlenül a valódi világból történik a becslés. Például a  $P(Eső|VörösAzÉgEste = igaz)$  becslésénél egyszerűen megszámoljuk, milyen gyakran esik az után, hogy az előző este az ég vörös volt – figyelmen kívül hagyva azokat az estéket, amikor az ég nem volt vörös. (Itt a világ maga játszsa a mintageneráló algoritmus szerepét.) Nyilvánvalóan, ha az ég nagyon ritkán vörös, akkor ez hosszú időt vehet igénybe, és ez az, ami az elutasító mintavétel gyengesége.

### Valószínűségi súlyozás

A **valószínűségi súlyozás** (**likelihood weighting**) elkerüli az elutasító mintavételezés gyengeségét azáltal, hogy csak az  $e$  bizonyítékkal konzisztens eseményeket generál. Az algoritmus működésének a lefrásával kezdjük; majd megmutatjuk, hogy helyesen működik – azaz, hogy konzisztens valószínűség becsléseket generál. A **VALÓSZÍNŰSÉGI-SÚLYOZÁS** (lásd 14.14. ábra) rögzíti az  $E$  bizonyítékváltozók értékeit, és csak a maradék  $X$  és  $Y$  változókat mintavételezi. Ez garantálja, hogy minden generált esemény konzisztens a bizonyítékkal. Azonban nem minden esemény egyenlő. Mielőtt megállapítanánk a számlálási eredményeket a célváltozó eloszlásában, minden eseményt súlyozunk azsal a valószínűséggel, amely megadja, hogy az esemény mennyire van összhangban a bizonyítékkal. Ezt a valószínűséget az egyes bizonyítékváltozók feltételes valószínű-

ségeinek a szorzatával mérjük, a szüleik ismeretében. Szemléletesen, azoknak az eseményeknek, ahol a bizonyíték valószínűtlennek tűnik, kisebb súlyt kell adni.

Alkalmazzuk az algoritmust a 14.11. (a) ábrán látható háló esetén a  $P(Eső|Locsoló = igaz, VizesPázsit = igaz)$  kérdésre. A folyamat a következőképpen halad: először a w súlyt 1,0-ra állítjuk. Aztán generálunk egy eseményt:

1. Sorsoljunk a  $P(Felhős) = \langle 0,5, 0,5 \rangle$  eloszlásból; tegyük fel, hogy *igaz*-at kapunk.
2. A *Locsoló* egy bizonyítékváltozó *igaz* értékkel. Ezért beállítjuk, hogy

$$w \leftarrow w \times P(Locsoló = igaz | Felhős = igaz) = 0,1$$

3. Sorsoljunk a  $P(Eső|Felhős = igaz) = \langle 0,8, 0,2 \rangle$  eloszlásból; tegyük fel, hogy *igaz*-at kapunk.

4. A *VizesPázsit* egy bizonyítékváltozó *igaz* értékkel. Ezért beállítjuk, hogy

$$w \leftarrow w \times P(VizesPázsit = igaz | Locsoló = igaz, Eső = igaz) = 0,099$$

Itt a SÚLYOZOTT-MINTA az [*igaz, igaz, igaz, igaz*] eseményt adja ki 0,099 súlyjal, és ezt az *Eső = igaz* esetnél vesszük számításba. A súly alacsony, mivel az esemény egy felhős napot ír le, amikor valószínűtlen, hogy a locsoló be van kapcsolva.

Hogy megértsük, miért is működik a valószínűségi súlyozás, azzal kezdjük, hogy vizsgáljuk a SÚLYOZOTT-MINTA  $S_{WS}$  mintavételi eloszlását. Emlékezzünk, hogy az **E** bizonyítékváltozók értéke (**e**) rögzített. A többi változót **Z**-vel jelöljük, azaz **Z** = {**X**} ∪ **Y**. Az algoritmus **Z** minden változóját mintavételezi, szülei adott értékei mellett:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^l P(z_i | szülők(Z_i)) \quad (14.6)$$

Vegyük észre, hogy a *Szülők*( $Z_i$ ) tartalmazhat minden rejtett, minden bizonyítékváltozókat is. A  $P(z)$  a priori eloszlástól eltérően az  $S_{WS}$  eloszlás szentel valamennyi figyelmet a bizonyítéknak is: a mintavételezett értékekkel minden  $Z_i$  esetén befolyásolják a  $Z_i$  ösei között levő bizonyítékok. Másfelől,  $S_{WS}$  kevesebb figyelmet szentel a bizonyítékoknak, mint a  $P(\mathbf{z}|\mathbf{e})$  valódi a posteriori eloszlás, mivel a mintavételezett értékek minden  $Z_i$  esetén *figyelmen kívül hagyják* azokat a bizonyítékokat, amelyek  $Z_i$ -nek nem ősei.<sup>8</sup>

A  $w$  valószínűségi súly pótja ki a különbséget a valódi és a kívánt mintavételi eloszlás között. Egy adott **z**-ből és **e**-ből álló **x** minta súlya az összes bizonyítékváltozó valószínűségének szorzata a szülei értékei mellett (amelyek közül néhány vagy akár az összes a  $Z_i$ -k között lehet):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^m P(e_i | szülők(E_i)) \quad (14.7)$$

A (14.6) és (14.7) egyenleteket összeszorozva láthatjuk, hogy egy minta *súlyozott* valószínűsége különösen kényelmes alakú

$$S_{WS}(\mathbf{z}, \mathbf{e})w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^l P(z_i | szülők(Z_i)) \prod_{i=1}^m P(e_i | szülők(E_i)) = P(\mathbf{z}, \mathbf{e}) \quad (14.8)$$

<sup>8</sup> Ideálisan, a  $P(\mathbf{z}|\mathbf{e})$  valódi a posteriori eloszlással megegyező mintavételi eloszlást szeretnénk használni, hogy az összes bizonyíték figyelembe vegyük. Ez azonban nem tehető meg hatékonyan. Ha lehetne, akkor a kívánt valószínűséget tetszőleges pontossággal közelíthetnénk polinomiális számú mintával. Megmutatható, hogy nem létezhet ilyen polinom idejű közelítő séma.

```

function VALÓSZÍNÜSEGI-SÚLYOZÁS( $X, e, bn, N$ ) returns  $P(X|e)$  egy becslése
  inputs:  $X$ , a lekérdezés változója
     $e$ , a bizonyíték mint esemény
     $bn$ , egy Bayes-háló
     $N$ , a generálandó minták száma
  local variables:  $W, X$  értékeihez tartozó súlyozott számítások vektora, kezdetben nulla
  for  $j = 1$  to  $N$  do
     $x, w \leftarrow$  SÚLYOZOTT-MINTA( $bn$ )
     $W[x] \leftarrow W[x] + w$ , ahol  $x$  az  $X$  értéke  $x$ -ben
  return NORMALIZÁL( $W[X]$ )

function SÚLYOZOTT-MINTA( $bn, e$ ) returns egy esemény és egy súly
   $x \leftarrow$  egy esemény  $n$  elemmel;  $w \leftarrow 1$ 
  for  $i = 1$  to  $n$  do
    if  $X_i$  értéke szerepel  $e$ -ben
      then  $w \leftarrow w \times P(X_i = x_i | \text{szüilik}(X_i))$ 
      else  $x_i \leftarrow$  egy véletlen minta  $P(X_i = x_i | \text{szüilik}(X_i))$  szerint
  return  $x, w$ 

```

**14.14. ábra.** A valószínűségi súlyozás algoritmusa Bayes-hálóban történő következtetéshez

mivel a két szorzat lefedi az összes változót a hálóban, lehetővé téve, hogy az együttes valószínűségre a (14.1) egyenletet használjuk.

Most már könnyen megmutatható, hogy a valószínűségi súlyozásos mintavétel konzisztens. Az  $X$  bármely konkrét  $x$  értékére a becsült a posteriori valószínűség a következőképpen számítható ki:

$$\begin{aligned}
 \hat{P}(x|e) &= \alpha \sum_y N_{WS}(x, y, e) w(x, y, e) && \text{a VALÓSZÍNÜSEGI-SÚLYOZÁS-ból} \\
 &\approx \alpha' \sum_y S_{WS}(x, y, e) w(x, y, e) && \text{nagy } N\text{-ekre} \\
 &= \alpha' \sum_y P(x, y, e) && \text{a (14.8) egyenlet miatt} \\
 &= \alpha' P(x, e) = P(x|e)
 \end{aligned}$$

Így a valószínűségi súlyozás konzisztens becslésekkel szolgáltat.

Mivel a valószínűségi súlyozás az összes generált mintát felhasználja, sokkal hatékonyságban lehet, mint az elutasításos mintavétel. Azonban a teljesítménye leromlik, amint a bizonyítékváltozók száma növekszik. Ez azért következik be, mert a legtöbb mintának nagyon kis súlya lesz, és így a súlyozott becslést főként a minták azon csekély töredéke határozza meg, amelyek egy elenyésző valószínűségnél jobban illeszkednek a bizonyítékokhoz. A probléma még erőteljesebben jelentkezik, ha a bizonyítékváltozók később fordulnak elő a változó sorrendben, mivel ekkor a minták olyan szimulációk, amelyek kevés hasonlóságot mutatnak a bizonyítékok által sugalmazott valósághoz.

## Következtetés Markov-lánc szimulációval

Ebben a fejezetben a **Markov lánc Monte Carlo** (MCMC, **Markov chain Monte Carlo**) algoritmust ismertetjük, hogy Bayes-hálókban következtethessünk. Először leírjuk, mit csinál az algoritmus, majd elmagyarázzuk, hogy miért is működik, és miért van ilyen bonyolult neve.

### Az MCMC algoritmus

Az előző két mintavételelő algoritmustól eltérően, amelyek az egyes eseményeket a semmiből generálják, az MCMC minden eseményt az azt megelőző esemény véletlen módosításával generál. Ezért a hálót hasznos úgy elképzelní, mint aminek van egy konkrét *jelenlegi állapota*, ami minden változóra meghatároz egy értéket. A következő állapot generálása egy  $X_i$  nem bizonyítékváltozóhoz tartozó érték véletlenszerű mintavételezésével történik, az  $X_i$  Markov-takarójába tartozó változók *jelenlegi értékeinek feltétele mellett*. (Emlékezzünk vissza az 586. oldalra: egy változó Markov-takarója a szüleiből, gyermekeiből és gyermekei szüleiből áll.) Az MCMC így véletlen bolyongást végez az állapottérben – a lehetséges teljes érték hozzárendelések terében –, egyszerre egy változót billentve át, de rögzítetten tartva a bizonyítékváltozókat.

Gondoljuk meg a  $P(Eső|Locsoló = \text{igaz}, VizesPázsit = \text{igaz})$  kérdést a 14.11. (a) ábrán látható háló esetén. A Locsoló és a VizesPázsit bizonyítékváltozók rögzítettek a megfigyelt értékeikre, míg a rejtejtett Fehős és Eső változók véletlenszerűen inicializáltak – mondjuk az egyik *igaz*-ra, a másik *hamis*-ra. Így a kezdeti állapot [*igaz, igaz, hamis, igaz*]. Ekkor a következő lépéseket hajtjuk végre ismétlődően:

1. A *Fehős*-t mintavételezzük a Markov-takarójába eső változók jelenlegi értékeinek ismeretében: ebben az esetben a  $P(Fehős|Locsoló = \text{igaz}, Eső = \text{hamis})$  szerint mintavételezünk. (Hamarasan megmutatjuk, hogyan számítható ki ez az eloszlás.) Tegyük fel, hogy az eredmény *Fehős = hamis*. Ekkor az új állapot [*hamis, igaz, hamis, igaz*].
2. Az *Eső*-t mintavételezzük a Markov-takarójába eső változók jelenlegi értékeinek ismeretében: ebben az esetben a  $P(Eső|Fehős = \text{hamis}, Locsoló = \text{igaz}, VizesPázsit = \text{igaz})$  szerint mintavételezünk. Tegyük fel, hogy ennek eredménye *Eső = igaz*. Ekkor az új állapot [*hamis, igaz, igaz, igaz*].

A folyamat során meglátogatott minden egyes állapot egy olyan minta, ami hozzájárul az *Eső* célváltozó megbecsléséhez. Ha a folyamat 20 állapotot látogatott meg, ahol az *Eső* igaz, és 61 állapotot, ahol az *Eső* hamis, akkor a kérdésre a választ a NORMALIZÁL( $\langle 20, 60 \rangle$ ) =  $\langle 0,25, 0,75 \rangle$  adja. A teljes algoritmus a 14.15. ábrán látható.

### Miért működik az MCMC?

Most megmutatjuk, hogy az MCMC konzisztens becslésekkel szolgáltat az a posteriori valószínűségekre. A fejezet anyaga elégé teknikai jellegű, de az alapállítás egyszerű:

```

function MCMC-KÉRDEZÉS( $X, e, bn, N$ ) returns  $P(X|e)$  egy becslése
local variables:  $N[X]$ ,  $X$  értékeihez tartozó számlálók vektora, kezdetben nulla
     $Z$ : a nem bizonyíték változók  $bn$ -ben
     $x$ : a háló jelenlegi állapota, kezdetben  $e$ -re állítva

    a  $Z$ -beli változók kezdeti értékét  $x$ -ben válasszuk meg véletlenszerűen
    for  $j = 1$  to  $N$  do
        for each  $Z_i$   $Z$ -ben do
            sorsoljuk  $Z_i$  értékét  $x$ -ben  $P(Z_i|mb(Z_i))$ -ből, ahol az  $MB(Z_i)$  értéke  $x$  szerinti
             $N[x] \leftarrow N[x] + 1$ , ahol  $x$  az  $Z_i$  értéke  $x$ -ben
    return NORMALIZÁL( $N[X]$ )

```

14.15. ábra. Az MCMC algoritmus Bayes-hálóban történő közelítő következtetéshez

*a mintavételei folyamat egy olyan „dinamikus egyensúlyban” állapotok meg, amelyben az egyes állapotokban töltött idő hosszú távon számolt hányada pontosan az a posteriori valószínűséggel arányos. Ez a figyelemre méltó tulajdonság a speciális átmenet-valószínűség (transition probability) miatt áll fent, amely szerint a folyamat egyik állapotból a másikba lép át, ahogyan ezt a feltételes eloszlást a mintavételezett változó Markov-takarója meghatározza.*

Legyen  $q(x \rightarrow x')$  az a valószínűség, hogy a folyamat  $x$  állapotból  $x'$  állapotba lép át. Ez az átmenet-valószínűség definíálja az úgynevezett **Markov-láncot** (**Markov chain**) az állapponton. (A Markov-láncok a 15. és 17. fejezetben is kiemelkedő szerepet játszanak.)

Tegyük fel most, hogy  $t$  lépéssnyit futtatjuk a Markov-láncot, és legyen  $\pi_t(x)$  annak a valószínűsége, hogy a rendszer a  $t$  időpillanatban az  $x$  állapotban van. Hasonlóan, legyen  $\pi_{t+1}(x')$  annak a valószínűsége, hogy a rendszer  $t+1$  időpillanatban  $x'$  állapotban van. Ismerve  $\pi_t(x)$ -et a  $\pi_{t+1}(x')$  kiszámítható úgy, hogy minden olyan állapotra, amiben a rendszer  $t$  időpillanatban lehet, összegezzük az állapot valószínűségének és az ebből az állapotból az  $x'$ -be való átlépés valószínűségének szorzatát:

$$\pi_{t+1}(x') = \sum_x \pi_t(x) q(x \rightarrow x')$$

Akkor mondjuk, hogy egy lánc elérte a **stacionárius eloszlását** (**stationary distribution**), ha  $\pi_t = \pi_{t+1}$ . Jelöljük ezt a stacionárius eloszlást  $\pi$ -vel; az ezt definiáló egyenlet így

$$\pi(x') = \sum_x \pi(x) q(x \rightarrow x') \text{ minden } x' \text{-re} \quad (14.9)$$

Ha a  $q$  átmenet-valószínűség eloszlás eleget tesz néhány alapvető feltevésnek,<sup>9</sup> akkor bármely  $q$  esetén pontosan egyetlen  $\pi$  eloszlás elégíti ki ezt az egyenletet.

A (14.9) egyenlet úgy tekinthető, mint ami azt állítja, hogy az egyes állapotokból a „kilépések” várható értéke (azaz a jelenlegi „populáció”) egyenlő az összes állapotból a „belépések” várható értékével. Ez a kapcsolat nyilvánvalóan teljesíthető, ha a várható átlépések minden két irányban ugyanakkorán bármely állapotpár esetén. Ez a tulajdonság a **teljes egyensúly** (**detailed balance**):

<sup>9</sup> A  $q$  által definiált Markov-láncnak **ergodikusnak** (**ergodic**) kell lennie – azaz lényegében, minden állapotnak elérhetőnek kell lennie bármely másik állapotból, és nem lehetnek szigorúan periodikus ciklusok.

$$\pi(\mathbf{x})q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}')q(\mathbf{x}' \rightarrow \mathbf{x}) \quad \text{ minden } \mathbf{x}, \mathbf{x}'\text{-re} \quad (14.10)$$

Megmutathatjuk, hogy a teljes egyensúly maga után vonja a stacionaritást, egyszerűen az  $\mathbf{x}$ -ek felett összegezve a (14.10) egyenletben. Azt kapjuk, hogy

$$\sum_{\mathbf{x}} \pi(\mathbf{x})q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}'} \pi(\mathbf{x}')q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

ahol az utolsó lépés abból következik, hogy  $\mathbf{x}'$ -ből biztosan történik átmenet.

Most megmutatjuk, hogy az MCMC-KÉRDEZ mintavételei lépéseiben megadott  $q(\mathbf{x} \rightarrow \mathbf{x}')$  átmenet-valószínűség eleget tesz a teljes egyensúly egyenletének, ahol a stacionárius eloszlás a  $P(\mathbf{x}|\mathbf{e})$ , a rejtegett változók valódi a posteriori eloszlása. Ezt két lépésben teszszük meg. Először, egy Markov-láncot definiálunk, amelyben minden változót az összes többi változóval feltételesen mintavételezünk, és megmutatjuk, hogy ez eleget tesz a teljes egyensúly egyenletének. Majd egyszerűen megállapítjuk, hogy Bayes-hálóknál ez ekvivalens azzal a feltételes mintavételezéssel, hogy a feltétel a változó Markov-takarója (lásd 586. oldal).

Legyen  $X_i$  a mintavételezett változó, és jelölje  $\bar{\mathbf{x}}_i$  az összes többi rejtegett változót. A jelenlegi állapotban az értékük  $x_i$  és  $\bar{\mathbf{x}}_i$ . Ha az  $X_i$ -nek új  $x'_i$  értéket mintavételezünk feltételesen az összes többi változóval, beleértve a bizonyítékot is, akkor

$$q(\mathbf{x} \rightarrow \mathbf{x}') = q((x_i, \bar{\mathbf{x}}_i) \rightarrow (x'_i, \bar{\mathbf{x}}_i)) = P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e})$$

Ezt az átmenet-valószínűséget nevezik Gibbs-mintavételezőnek (**Gibbs sampler**), ami az MCMC egy nagyon kényelmes alakja.

Most megmutatjuk, hogy a Gibbs-mintavételező teljes egyensúlyban van a helyes a posteriori eloszlása szerint:

$$\begin{aligned} \pi(\mathbf{x})q(\mathbf{x} \rightarrow \mathbf{x}') &= P(\mathbf{x}|\mathbf{e})P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x_i, \bar{\mathbf{x}}_i | \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \\ &= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(\bar{\mathbf{x}}_i | \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \quad (\text{alkalmazva a láncszabályt az első tagon}) \\ &= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \quad (\text{alkalmazva a láncszabály megfordítását}) \\ &= \pi(\mathbf{x}')q(\mathbf{x}' \rightarrow \mathbf{x}) \end{aligned}$$

Ahogy az 586. oldalon kimondtuk, egy változó független az összes többi változótól a Markov-takarójának a feltételében; így:

$$P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x'_i | mb(X_i))$$

ahol  $mb(X_i)$  jelöli az  $X_i$  Markov-takarójában,  $MB(X_i)$ -ben lévő változók értékeit. Amint a 14.10. feladat mutatja, egy változó valószínűsége a Markov-takarójának ismeretében arányos a változó szüleivel vett feltételes valószínűségének és az egyes gyermekek azok szüleivel vett feltételes valószínűségeinek a szorzatával:

$$P(x'_i | mb(X_i)) = \alpha P(x'_i | szülők(X_i)) \times \prod_{Y_j \in Gyermekek(X_i)} P(y_j | szülők(Y_j)) \quad (14.11)$$

Így egy változó átléptetéséhez az  $X_i$  gyermekéinek számával megegyező számú szorzás szükséges.

Itt csupán az MCMC-nek egy egyszerű változatát tárgyalunk, nevezetesen a Gibbs-mintavételezőt. Legáltalánosabb formájában az MCMC hatékony módszer valószínűségi modellekkel való számolásra, és számos változatát fejlesztették ki, közöttük

a 4. fejezetben bemutatott szimulált lehűtés algoritmust, a 7. fejezetben a sztochasztikus kielégíthetőség algoritmust, valamint a 15. fejezetben a Metropolis–Hastings-minta-vételezőt.

## 14.6. ELSŐRENDŰ REPREZENTÁCIÓK VALÓSZÍNÜSÉGI KITERJESZTÉSE

A 8. fejezetben kifejtettük az elsőrendű logika reprezentációs előnyeit az ítéletlogikához képest. Az elsőrendű logika objektumok és közöttük lévő relációk létezését veszi alapul, és képes tényeket kifejezni a tárgyterület néhány vagy összes objektumáról. Ez gyakran vezet olyan reprezentációkra, amelyek összehasonlítathatóan tömörebbek, mint az ekvivalens ítéletlogikai leírások. Márpedig a Bayes-hálók alapvetően az ítéletlogikához kapcsolódnak: a változók halmaza rögzített és véges, és minden változónak rögzített értékkészlete van. Ez a tény korlátozza a Bayes-hálók felhasználhatóságát. *Ha sikerül találnunk egy módszert arra, hogy az elsőrendű reprezentációk kifejezőerejét a valószínűség-számítással kombináljuk, várható, hogy ugrásszerűen megnöveljük a kezelhető problémák körét.*

Ennek a célnak az eléréséhez a következő alapvető felismerés szükséges: az ítéletlogika szemszögéből nézve a Bayes-háló elemi események feletti valószínűségeket határozz meg, amelyek mindegyike a hálózat minden változójára meghatároz egy értéket. Azaz, az ítéletlogika terminológiája szerint egy elemi esemény egy **modell** vagy egy **lehetséges világ**. Az elsőrendű felfogás szerint egy modell (az értelmezésével együtt) meghatározza az objektumok egy tárgykörét, a közöttük fennálló kapcsolatokat, és a tudásbázisbeli konstansok és predikátumok egy megfeleltetését a modellbeli objektumokra és kapcsolatokra. Ezért *egy elsőrendű valószínűségi tudásbázisnak az összes lehetséges elsőrendű modell valószínűségét definiálnia kell*. Legyen  $\mu(M)$  a tudásbázis által az  $M$  modellhez rendelt valószínűség. Ekkor bármely  $\phi$  elsőrendű állításnak a  $P(\phi)$  valószínűségét a szokásos módon kapjuk, összegezve azon lehetséges világok felett, ahol  $\phi$  igaz:

$$P(\phi) = \sum_{M: \phi \text{ igaz } M\text{-ben}} \mu(M) \quad (14.12)$$

Eddig rendben is volna. Azonban van egy probléma: az elsőrendű modellek halmaza végtelen. Ez azt jelenti, hogy (1) az összegzés kivitelezhetetlen, és (2) egy teljes, konziszens eloszlás meghatározása a világok egy végtelen halmazán igen bonyolult lehet.

Fogadjunk el ezért bizonyos korlátozásokat, legalábbis időlegesen. Nevezetesen, hogy tervezünk egy olyan korlátozott nyelvet, amelyhez csak véges modell tartozik. Ez számos módon megtehető. Itt a **relációs valószínűségi modellek** (**relational probability models, RVM**) vagy **RVM-eket** ismertetjük, amelyek a szemantikus hálók (lásd 10. fejezet) és objektumrelációs adatbázisok elméletéből is vettek át elemeket. Egyéb megközelítések az irodalmi és történeti megjegyzésekben szerepelnek.

Az RVM-ek megengednek konstans szimbólumokat, amelyek objektumokat neveznek meg. Például legyen *KovácsProf* egy professzor neve, *János* pedig egy diák neve. minden objektum az osztály egy példánya: például *KovácsProf* egy *Professzor* és *János* egy *Diák*. Feltessük, hogy minden konstans szimbólum osztálya ismert.

Függvényszimbólumaink kétfelé lesznek osztva. Az első fajtába tartozók, az **egyszerű függvények (simple functions)**, egy objektumot nem egy másik strukturált objektumra képeznek le, hanem egy értékre egy rögzített értéktartományból, pontosan úgy, ahogy egy valószínűségi változónál. Például az *Intelligencia(János)* és a *Finanszírozás(KovácsProf)* lehet *magas* vagy *alacsony*; a *Siker(János)* és a *Hírnév(KovácsProf)* lehet *igaz* vagy *hamis*. Függvényszimbólumok nem alkalmazhatók olyan értékekre, mint az *igaz* vagy a *hamis*, így nem lehetséges egyszerű függvények egymásba ágyazása. Ezzel elkerüljük a végtelenség egyik forrását. Egy adott objektumon alkalmazott egyszerű függvény értéke lehet megfigyelt vagy ismeretlen; reprezentációnkban ezek lesznek az elemi valószínűségi változók.<sup>10</sup>

Megengedünk emellett **komplex függvényeket (complex functions)** is, amelyek objektumokat képeznek le más objektumokra. Például a *Konzulens(János)* lehet *KovácsProf*. minden komplex függvénynek van egy megadott értéktartománya és értékkészlete, amik osztályok. Például a *Konzulens* értéktartománya a *Diák* és érték-készlete a *Professzor*. A függvények csak helyes osztályokra alkalmazhatók; például a *KovácsProf Konzulens-e* nem definiált. Komplex függvények egymásba ágyazhatók: a *TanszékVezető(Konzulens(János))* lehet *MórProf*. Egyelőre feltesszük, hogy minden komplex függvény értéke ismert az összes konstans szimbólumra. Mivel a *TB* véges, ez maga után vonja, hogy komplex függvények alkalmazásának minden láncolata elvezet a véges számú objektum egyikéhez.<sup>11</sup>

Az utolsó elem, amire szükségünk van, az a valószínűségi információ. minden egyszerű függvényhez specifikáljuk a szülők egy halmazát, csakúgy, mint a Bayes-hálóknál. A szülők lehetnek ugyanannak az objektumnak más egyszerű függvényei; például egy *Professzor*-nak a *Finanszírozás-a* függhet a *Hírnév-től*. A szülők lehetnek még *kapcsolódó* objektumok egyszerű függvényei – például egy diák *Sikere* függhet a diák *Intelligenciá-jától* és a diák konzulensének *Hírnév-től*. Ezek valóban *univerzálisan kvantifikált* állítások egy osztály minden objektumának szüleiről. Így írhatjuk azt, hogy

$$\forall x \ x \in \text{Diák} \Rightarrow \text{Szülők}(\text{Siker}(x)) = \{\text{Intelligencia}(x), \text{Hírnév}(\text{Konzulens}(x))\}$$

(Kevésbé formálisan a 14.16. (a) ábrához hasonló diagrammokat rajzolhatunk.) Most megadjuk a gyermek feltételes valószínűség eloszlását, ahol a feltételek a szülei jelentik. Például mondhatjuk, hogy:

$$\forall x \ x \in \text{Diák} \Rightarrow$$

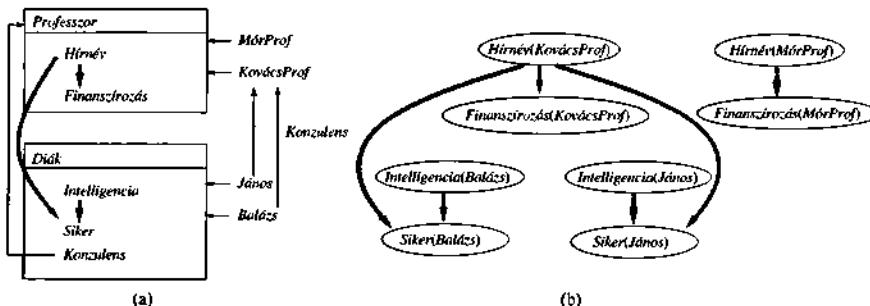
$$P(\text{Siker}(x) = \text{igaz} | \{\text{Intelligencia}(x) = \text{magas}, \text{Hírnév}(\text{Konzulens}(x)) = \text{igaz}\}) = 0,95$$

Csakúgy, mint a szemantikus hálókban, csatolhatunk feltételes eloszlásokat magához az osztályhoz is, így a példányok öröklik (*inherit*) a függéseket és a feltételes valószínűségeket az osztálytól.

Az RVM nyelv szemantikája felteszi, hogy minden konstans szimbólum egy külön-álló objektumra hivatkozik – az **egyedi név feltételezést (unique names assumption)**

<sup>10</sup> Nagyon hasonló szerepet töltenek be, mint a változómentes atomi mondatok, amelyek a 9.1. alfejezetben leírt ítéletlogikai állításokra való visszavezetési folyamatban generálódnak.

<sup>11</sup> Ez a megkötés azt jelenti, hogy nem használhatunk olyan komplex függvényeket, mint *Apa* vagy *Anya*, amelyek potenciálisan végtelen láncokat eredményezhetnének egy ismeretlen objektummal végződve. Ezt a megkötést később újra meggondoljuk.



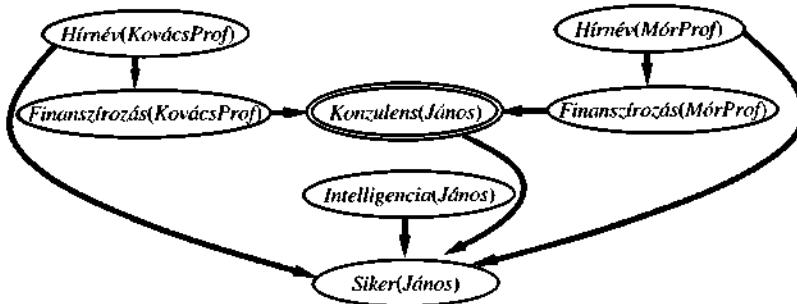
**14.16. ábra.** (a) Egy két osztályt (*Professzor* és *Diák*) leíró RVM. Két professzor és két diák van, minden két diák konzulense *KovácsProf*. (b) Az (a)-beli RVM Bayes-háló ekvivalense.

a 10. fejezetben írtuk le. Ezen feltevés mellett és a korábban felsorolt megkötésekkel, meg lehet mutatni, hogy minden RVM valószínűségi változók egy rögzített, véges halma-zát generálja, amelyek mindegyike egy egyszerű függvény egy konstans szimbólumra alkalmazva. Ekkor – feltéve, hogy a szülő-gyermekek függések *körmentesek* – megkonstruálhatunk egy ekvivalens Bayes-hálót. Azaz az RVM és a Bayes-háló azonos valószínűségeket határoz meg minden lehetséges világra. A 14.6. (b) ábra mutatja azt a Bayes-hálót, ami a 14.16. (a) ábra RVM-jéhez tartozik. Vegyük észre, hogy az RVM-ben szereplő *Konzulens* kapcsolatok nincsenek jelen a Bayes-hálóban. Ennek oka, hogy rögzítettek és ismertek. Azonban *közvetetten* feltűnnek a háló topolójában; például a *Siker(János)* szülője a *Hírnév(KovácsProf)*, mivel a *Konzulens(János)* értéke a *KovácsProf*. Általában, az objektumok között fennálló relációk meghatározzák az ezen objektumok tulajdonságai között fennálló függőségeket.

Az RVM-ek kifejezési erejének megnövelésére számos módszer kínálkozik. Megengedhetünk **rekurzív függéseket** (*recursive dependencies*) a változók között, hogy bizonyosfajta visszatérő kapcsolatokat kezelnél tudunk. Például tegyük fel, hogy a gyorsétermi ételektől való függőséget egy *McGén* okozza. Ekkor minden  $x$ -re  $McGén(x)$  függ a  $McGén(Apa(x))$  és  $McGén(Anyá(x))$ -től, amelyek viszont függnek a  $McGén(Apa(Apa(x)))$  és a  $McGén(Anyá(Apa(x)))$ -től és így tovább. Bár az ilyen tudásbázisok végtelen változót tartalmazó Bayes-hálókhhoz tartoznak, fixpont-egyenletekből olykor megoldásokhoz juthatunk. Például a kiszámítható a *McGén* egyensúlyi eloszlása, az öröklődés adott feltételes valószínűsége mellett. A rekurzív tudásbázisok egy másik igen fontos családja a 15. fejezetben leírt **időbeli valószínűségi modellek** (*temporal probability models*) tartalmazza. Ezekben a modellekben a  $t$  időpillanatbeli állapot tulajdonságai a  $t - 1$  időpillanatbeli állapot tulajdonságaitól függnek és így tovább.

Az RVM-ek szintén kiterjeszhetők, hogy megengedjenek **kapesolati bizonysalanságot** (*relational uncertainty*) is – azaz bizonysalanságot a komplex függvények értékeinél. Például lehet, hogy nem tudjuk, kicsoda a *Konzulens(János)*. Ekkor a *Konzulens(János)* egy valószínűségi változó lesz, aminek lehetséges értékei *KovácsProf* és *MórProf*. A megfelelő hálót a 14.17. ábra mutatja.

Szintén jelen lehet **azonossági bizonysalanság** (*identity uncertainty*): például esetleg nem tudjuk, hogy *Mária* és *KovácsProf* ugyanaz a személy-e. Azonossági bizonysalanság esetén az objektumok és az állítások száma változhat a lehetséges világokban.



**14.17. ábra.** Egy Bayes-háló részlete, ami egy olyan RVM-hez tartozik, amelyben a *Konzulens(János)* ismeretlen, de vagy *KovácsProf*, vagy *MórProf*. A konzulens választása az egyes professzorok finanszírozásától függ. Vegyük észre, hogy a *Siker(János)* most *mind a két professzor Hírnév-étől* függ, bár a *Konzulens(János)* értéke meghatározza, hogy valójában melyiknek is van hatása.

Annak a világnak, ahol *Mária* és *KovácsProf* ugyanaz a személy eggyel kevesebb objektuma van, mint annak a világnak, ahol ők különböző személyek. Ez a következetési eljárást bonyolultabbá teszi, de a (14.12) egyenlet által lefektetett alapelvek érvényben marad: bármely kijelentés valószínűsége jól definiált és kiszámítható. Az azonossági bizonytalanság különösen fontos robotok és beágyazott érzékelőrendszerek esetén, amelyeknek több objektumot is követniük kell. Erre a problémára a 15. fejezetben viszszatérünk.

Vizsgáljuk meg most a következetés kérdését. Világos, hogy a következetés elvégzhető az ekvivalens Bayes-hálóban, feltéve, hogy az RVM nyelvet úgy korlátozzuk, hogy az ekvivalens háló véges és rögzített struktúrájú. Ez analóg azzal a módszerrel, ahogy elsőrendű logikai következetést végezhetünk el ítéletlogikai következetéssel, az ekvivalens ítéletlogikai tudásbázisban (lásd 9.1. alfejezet). Ahogy a logikai esetben is, az ekvivalens háló túlságosan nagy ahhoz, hogy megkonstruáljuk, különösen ahhoz, hogy kiértékeljük. A sűrű összeköttség szintén probléma (lásd 14.12. feladat). A közelítő algoritmusok, mint az MCMC (lásd 14.5. alfejezet), ezért igen hasznosak RVM-kben való következetésre.

Amikor az MCMC-t egy egyszerű RVM-tudásbázisának az ekvivalens Bayes-hálójára alkalmazzuk, ahol a tudásbázis nem tartalmaz kapcsolati vagy azonossági bizonytalanságot, az algoritmus a lehetséges világok teréből mintavételez, amit az objektumok egyszerű függvényeinek értékei határoznak meg. Világosan látható, hogy ez a megközelítés kiterjeszthető a kapcsolati és az azonossági bizonytalanságok kezelésére is. Ebben az esetben a lehetséges világok közötti átmenet lehet, hogy egyegyszerű függvény értékét vagy egy komplex függvényt változtat meg, ami a függési struktúra megváltozásához is vezethet. Az átmenetek szintén megváltoztathatják a konstans szimbólumok közötti azonossági relációkat. Így az MCMC elegáns módszernek tűnik következetések elvégzésére igen nagy kifejezőerejű, elsőrendű valószínűségi tudásbázisokban.

A kutatás ezen a területen még a kezdeteknél tart, de az egyre nyilvánvalóbb, hogy az elsőrendű valószínűségi érvelés az MI-rendszerek óriási hatékonyságnövekedését eredményezi a bizonytalanság kezelése kapcsán. A potenciális alkalmazások között

jelen van a számítógépes látás, a számítógépes nyelvészeti, az információ-visszakeresés és a szituációértékelés. Ezen tételek mindegyikén az objektumok halmaza – és így valószínűségi változók halmaza – nem ismert előre, így a tisztán „kijelentésekben” alapuló módszerek, mint például a Bayes-hálók, nem képesek a helyzetet teljesen leírni. Ezen módszereket kiegészítették a modellek tere felett kereséssel, de az RVM-ek egyetlen modellben teszik lehetővé a következtetést ilyen típusú bizonytalanság esetén is.

## 14.7. EGYÉB MÓDSZEREK A BIZONYTALAN KÖRNYEZETBEN TÖRTÉNŐ KÖVETKEZTETÉSHEZ

Más tudományágak (például fizika, genetika, közgazdaságtan) régóta a valószínűségszámítást részesítették előnyben a bizonytalanság modellezésére.<sup>12</sup> Pierre Laplace 1819-es megfogalmazásában „A valószínűség-számítás semmi egyéb, mint a józan ész számokban megfogalmazva”. James Maxwell 1850-ben tett kijelentése szerint „a világ igazi logikája a valószínűségek kalkulusa, ami figyelembe veszi a valószínűség nagyságát, ahogy azt minden gondolkodó ember megtesszi vagy meg kellene tennie”.

Ezt a régi tradíciót látva talán meglepő, hogy az MI a valószínűség-számítás helyett számos alternatívával kísérletezett. Az 1970-es évekbeli első szakértői rendszerek teljesen figyelmen kívül hagyták a bizonytalanságot, és szigorúan logikai következtetést használtak, de hamar kiderült, hogy ez a legtöbb valós problémánál nem alkalmazható. A szakértői rendszerek következő generációja (különösen az orvosi tárgytartományokban) valószínűségi technikákat használt. A kezdeti eredmények biztatók voltak, de nem voltak skálázhatók nagyobb problémákra a teljes együttes valószínűség-eloszlás megadásához szükséges exponenciális számú érték miatt. (Hatékony Bayes-hálós algoritmusok ekkor még nem voltak ismertek.) Ennek eredményeképpen a valószínűségi megközelítések elvesztették vonzerejüket 1975 és 1988 között, és számos egyéb alternatívát próbáltak ki:

- Egy elterjedt nézet szerint a valószínűség-számítás alapvetően numerikus jellegű, míg az emberi színtérhozatal sokkal „kvalitatívabb”. Az bizonyos, hogy nem vagyunk tudatában, hogy különböző fokú bizonyossággal végünk numerikus számításokat. (Ahogyan a logikai egyesítés elvégzésének sem vagyunk tudatában, mégis úgy tűnik, hogy képesek vagyunk valamelyenfajta logikai érvelésre.) Lehetséges, hogy a bizonyosság valamilyen numerikus fokát közvetlenül tárolják az idegsejtek kapcsolatainak és aktivációinak erősségei. Ebben az esetben ezen erősségek tudatos elérésének a nehézsége nem meglepő. Azt is el kell ismerni, hogy a kvalitatív következtetési mechanizmus a valószínűség-számítási alapokra is felépíthető, így a „nincsenek számok” érv a valószínűségek ellen nem meggyőző. Mindazonáltal néhány kvalitatív módszer igen hasznosnak tűnhet a sajátos tulajdonságai miatt. Az egyik legjobban tanulmányozott módszer az alapértelmezésekben alapuló következtetés (**default reasoning**), ami nem úgy kezeli a következtetéseket, mint amikben „bizonyos fokig hiszünk”, hanem mint „amikben addig hiszünk, amíg nincs jobb indok valami másra hinni”. Az alapértelmezésekben alapuló következtetés a 10. fejezetben szerepel.

<sup>12</sup> A bizonytalan ismereteket és/vagy bizonytalan következtési formákat tartalmazó következtetéseket a többiakban egységesen „bizonytalansági következtetéseknek” nevezik. (A *ford.*)

- **Szabályalapú (rule-based)** megközelítésekkel is megpróbáltak bizonytalanságot kezelní. Ezek a módszerek a logikai szabályrendszerek sikereit akarták felhasználni, de hozzáadva minden szabályhoz valamilyen „kiagyalt” *ad hoc* tényezőt, ami a bizonytalanságot kezeli. Ezeket az eljárásokat az 1970-es évek közepén fejlesztették ki, és igen nagy számú szakértői rendszer alapjául szolgáltak, különösen orvosi és egyéb területeken.
- Az **ismerethiány (ignorance)** (ami a bizonytalansággal bizonyos értelemben szembeállítható) kérdéseit eddig még nem érintettük. Fontoljuk meg egy érme feldobását. Ha tudjuk, hogy az érme szabályos, akkor ésszerűnek tűnik, hogy a fej valószínűsége 0,5. Ha tudjuk, hogy az érme szabálytalan, de nem tudjuk, milyen módon, akkor 0,5 az egyetlen ésszerűnek tűnő valószínűség. Nyilvánvaló, hogy a két eset különböző, de a valószínűségtük láthatóan nem különbözteti meg őket. A **Dempster-Shafer-elmélet (Dempster–Shafer theory)** **intervallumértékeket (interval-valued)** használ a bizonyosság fokozataira, így reprezentálja az ágens tudását egy állítás valószínűségéről. Más, másodrendű valószínűségeket használó módszereket is tárgyalunk majd.
- A valószínűség-számítás ugyanazt a lételemeti feltevést használja, mint a logika: az események vagy igazak, vagy hamisak a világban, még ha az ágens bizonytalan is, melyik is áll fenn. A **fuzzy logika (fuzzy logic)** kutatói egy másik ontológiát javasoltak, ami megengedi a **meghatározatlanságot (vagueness)** is: azt, hogy egy esemény lehet „valamennyire” igaz. A meghatározatlanság és a bizonytalanság valójában ortogonális témaikörök, ahogy azt látni fogjuk.

A következő három alfejezet ezen megközelítések mindegyikét egy kicsit részletesebben mutatja be. Nem tartalmaznak részletes technikai leírásokat, de a megadott hivatkozások segítséget nyújtanak további részletek eléréshöz.

## Bizonytalansági következtetés szabályalapú eljárásokkal

A szabályalapú rendszerek a logikai következtés gyakorlati és szemléletes rendszerein való korai munkákból alakultak ki. A logikai rendszerek általában, és a logikai szabályalapú rendszerek speciálisan, rendelkeznek az alábbi három tulajdonsággal:

- **Lokalitás (locality):** logikai rendszerekben, ha van  $A \Rightarrow B$  formájú szabály, akkor a  $B$  kikövetkeztethető, ha az  $A$  bizonyíték adott, *bármely más szabályra való tekintet nélkül*. Valószínűségi rendszerekben a Markov-takaróban rendelkezésre álló összes tényt figyelembe kell venni.
- **Leválasztás (detachment):** ha egyszer találtunk egy logikai bizonyítást  $B$  állítás kikövetkeztetéséhez, az állítást tetszés szerint felhasználhatjuk, a származtatásától függetlenül. Vagyis **leválasztható (detached)** a megokolásaitól. Valószínűségek esetén azonban a bizonyosság bizonyítékának a forrása fontos az elkövetkező következtésekben.
- **Igazságfüggvény (truth-functionality):** a logikában az összetett kifejezések igazságértéke kiszámítható az alkotóinak igazságértékéből. Valószínűségek kombinálása nem lehetséges ezen a módon, csak szigorú, teljes függetlenségi feltételezések mellett.

Számos kísérlet történt olyan bizonytalansági következtetések kifejlesztésére, ahol ezek az előnyös tulajdonságok megmaradnak. Az elképzelés szerint a bizonyosság mértékét hozzárendeljük állításokhoz és szabályokhoz, és kifejlesztünk tisztán lokális sémákat a bizonyosságmértékek kombinálására és terjesztésére. A sémák egyben igazságfüggvények, például az  $A \vee B$  bizonyosságmértéke, az  $A$ - és  $B$ -beli bizonyosságok függvénye.

A rossz hír a szabályalapú rendszerek számára, hogy a *lokálitás*, a *leválasztás* és az *igazságfüggvény egyszerűen nem alkalmas a bizonytalansági következtetés számára*. Vizsgáljuk először az igazságfüggvényt. Legyen  $F_1$  az az esemény, hogy fejet kapunk egy szabályos érme feldobásakor, legyen  $I_1$  az az esemény, hogy ugyanekkor írást kapunk, és legyen  $F_2$  az az esemény, hogy fejet kapunk egy szabályos érme második feldobásakor. Világos, hogy minden esemény valószínűsége 0,5, és egy igazságáltató rendszernek ugyanazt az értéket kell hozzárendelnie bármely kettő konjunkciójához. De láthatjuk, hogy a konjunkció valószínűsége függ maguktól az eseményektől is, nem csak a valószínűségektől.

$P(A)$	$P(B)$	$P(A \vee B)$
$P(F_1) = 0,5$	$P(F_1) = 0,5$	$P(F_1 \vee F_1) = 0,50$
	$P(I_1) = 0,5$	$P(F_1 \vee I_1) = 1,00$
	$P(F_2) = 0,5$	$P(F_1 \vee F_2) = 0,75$

A helyzet még rosszabb, ha bizonyításokat láncolunk. Az igazságfüggvényeken alapuló rendszerekben léteznek  $A \rightarrow B$  alakú **szabályok (rules)**, amelyek lehetővé teszik, hogy kiszámítsuk  $B$ -beli bizonyosságunk értékét, mint annak függvényét, hogy mennyire hiszünk a szabályban, és mennyire hiszünk  $A$ -ban. Ekkor természetesen mind előre-, mind hátrahaladó rendszereket is ki lehet fejleszteni. A szabálybeli bizonyosságunkat konstansnak tételezzük fel, amit általában a tudásmérnök határoz meg, például  $A \rightarrow_{0,9} B$ .

Gondoljuk a 14.11. (a) ábrán látható vizes-pázsit szituációra. Ha képesek szeretnék lenni mind okozati, mind diagnosztikai következtetés elvégzésére, a két alábbi szabályra lesz szükségünk:

$$Eső \mapsto VizesPázsit$$

és

$$VizesPázsit \mapsto Eső$$

Ha nem vigyázunk, ez a két szabály visszacsatolásos módon fog működni úgy, hogy az  $Eső$ -re vonatkozó bizonyíték megnöveli a  $VizesPázsit$ -hoz tartozó bizonyosságot, ami viszont még inkább megnöveli az  $Eső$ -höz tartozó bizonyosságot. Nyilvánvaló, hogy bizonytalansági következtető rendszereknek nyilván kell tartaniuk, hogy milyen utakon terjedt a bizonyíték.

Az okok közötti következtetés (vagy kimagyarázás) szintén bonyodalmas. Gondoljuk meg, mi is történik, ha a következő két szabályunk van:

$$Locsoló \mapsto VizesPázsit$$

és

$$VizesPázsit \mapsto Eső$$

Tételezzük fel, hogy látjuk a locsolót bekapcsolva. Előreláncolással a szabályainkon keresztül ez megnöveli azt a bizonyosságot, hogy a pázsit vizes, ami viszont megnöveli az esőhöz tartozó bizonyosságot. De ez képtelenség: a locsoló bekapcsolt állapota megmagyarázza a pázsit vizességét, így *csökkentenie* kell az esőhöz tartozó bizonyosságot. Egy igazságfüggvényen alapuló rendszer úgy viselkedik, mintha a  $Locsoló \rightarrow Eső$  szabályban is hinne.

Ezen nehézségek mellett hogyan lehetséges, hogy az igazságfüggvényen alapuló rendszereket valaha is hasznosnak tartották? A megoldás abban rejlik, hogy korlátozták a feladatokat körét, és a szabálybázist gondosan úgy terveztek meg, hogy nemkívánatos kölcsönhatások ne következzenek be. A leghíresebb igazságfüggvényen alapuló rendszer bizonytalansági következtetésre a **bizonyossági tényező** (*certainty factor*) modell, amit a MYCIN orvosi diagnosztikai program számára fejlesztettek ki, és széles körben használták szakértői rendszerekben az 1970-es és az 1980-as években. A bizonyossági tényezők majdnem minden alkalmazása vagy tisztán diagnosztikai (mint a MYCIN), vagy tisztán okozati szabálybázist tartalmazott. Továbbá a tények csak a szabálybázis „gyökereinél” léptek be, és a legtöbb szabálybázis egyszeresen összekötött volt. Heckerman megmutatta, hogy ezen feltételek mellett a bizonyossági tényezőkön alapuló következtetés kicsit eltérő változata ekvivalens a polifabeli Bayes-következtetéssel (Heckerman, 1986). Más körülmények között, a bizonyossági tényezők katasztfálisan helytelen bizonyosságot eredményezhetnek a tények túlzott figyelembevétele miatt. Amint a szabályhalmaz nagyválik a szabályok között, a nemkívánatos interakciók egyre gyakoribbak, és a felhasználók azt találták, hogy számos szabálynak kell „hangolni” a bizonyossági faktorát, amikor új szabályokkal bővítenek. Szükségtelen mondani, hogy ez a megközelítés már nem ajánlott.

## Az ismerethiány reprezentálása: a Dempster–Shafer-elmélet

A Dempster–Shafer-elméletet (Dempster–Shafer theory)azzal a céllal hozták létre, hogy megkülönböztethetővé váljon a **bizonytalanság** (*uncertainty*) és az **ismerethiány** (*ignorance*). Ez az elmélet egy állításnak nem a valószínűségét számítja ki, hanem helyette azt, hogy mennyi annak a valószínűsége, hogy a bizonyíték támogatja az állítást. A bizonyosságnak ezt a mértékét **bizonyosságfüggvénynek** (*belief function*) nevezik, és  $Bel(X)$ -szel jelölik.

Most visszatérünk a pénzérmés példánkhöz, a bizonyosságfüggvény bemutatásához. Tételezzük fel, hogy egy rossz kinézetű alak odamegy önhöz és felajánljá, hogy fogadjon 1000 forintba, hogy a következő esetben fej lesz az eredmény. Mivel az érme lehet szabályos, de lehet szabálytalan is, milyen bizonyosságot kell ahhoz az eseményhez rendelni, hogy a következő eredmény fej lesz? A Dempster–Shafer-elmélet szerint minden nincs bizonyíték egyik esetre sem, így azt kell mondani, hogy a bizonyosság  $Bel(Fej) = 0$  és  $Bel(\neg Fej) = 0$ . Ez szkeptikussá teszi a Dempster–Shafer-elméletet használó következetető rendszereket, aminek van egy intuitív vonzereje. Most tegyük fel, hogy rendelkezésre áll egy szakértő, aki 90%-os bizonyossággal tanúsítja, hogy az érme szabályos (azaz 90%-ban biztos, hogy  $P(Fej) = 0,5$ ). Ekkor a Dempster–Shafer-elmélet azt adja, hogy  $Bel(Fej) = 0,9 \times 0,5 = 0,45$ , és hasonlóan  $Bel(\neg Fej) = 0,9 \times 0,5 = 0,45$ . Ekkor még mindig 10% az a „hiány”, amit a tényeink nem fednek le. „Dempster szabálya” (Dempster, 1968) megmutatja, hogy hogyan kombinálunk tényeket, hogy új  $Bel$  értéket kapunk, Shafer munkája pedig ezt egy teljes számítási modellként fogalmazta meg.

Mint az alapértelmezéses következtetésnél, itt is probléma a bizonyosság és a cselekvések összekapsolása. Valószínűségek esetén a döntéselmélet szerint, ha  $P(Fej) = 0,5$  és  $P(\neg Fej) = 0,5$ , akkor (feltéve, hogy nyerni és veszteni 1000 forintot pontosan egy-

más ellenettségei) a következtető számára közömbös lesz a tét elfogadása vagy elutasítása. Egy Dempster–Shafer-következtető esetén  $Bel(-Fej) = 0$ , és így nincs oka elfogadni, de ugyanakkor  $Bel(Fej) = 0$ , és így nincs oka vonakodni sem. Így látható, hogy ebben az esetben a Dempster–Shafer-következtető ugyanarra a következtetésre jutott, hogy hogyan cselekedjen. Sajnos a Dempster–Shafer-elmélet számos más esetben is megenged nem definiált döntéseket, amikor a valószínűségi következtetés konkrét választást eredményez. Valójában a hasznosság fogalma a Dempster–Shafer-modellben még nem kellően tisztázott.

Dempster–Shafer-elmélet egyik értelmezése, hogy egy valószínűségi intervallumot definiál – az intervallum a *Fej* esetén a szakértő tanúságtétele előtt [0, 1], míg utána [0,45, 0,55]. Az intervallum nagysága hasznos segítséget adhat annak megtíelésében, hogy mikor van szükségünk további tényekre: elárulhatja, hogy a szakértő tanúságtétele akkor fog segíteni, ha nem tudjuk, hogy szabályos-e az érme, de nem fog segíteni, ha már tudjuk, hogy az érme szabályos. Azonban nincsenek világos irányelvek ennek megtételére vonatkozóan, mivel nincs világos értelmezés arra, hogy mit is jelent az intervallum szélessége. A Bayes-megközelítésben ez a fajta következtetés könnyen elvégezhető, ha megvizsgáljuk azt, hogy újabb tények begyűjtésével hogyan változna egy bizonyosság. Például a fejek megjelenésével kapcsolatos bizonyosságunkat jelentősen befolyásolná, ha tudjuk, hogy az érme szabályos, és aszimmetrikus súly észrevétele jelentősen befolyásolná azon bizonyosságunkat, hogy az érme szabályos. Egy teljes Bayes-modell magában foglalna ehhez hasonló tényezőkre vonatkozó valószínűségi becsléseket is, megengedve, hogy a „tudatlanságunkat” bizonyosságaink azon megváltozásának mértékével fejezzük ki, amelyek jövőbeli megfigyelések esetén következnének be.

## A meghatározatlanság reprezentálása: fuzzy halmazok és logikák

A **fuzzy halmazok elmélete** (*fuzzy set theory*) egy eszköz annak specifikálására, hogy egy objektum milyen mértékben illeszkedik egy bizonytalan leíráshoz. Például fontoljuk meg azt az állítást, hogy „Nóri magas”. Igaz ez akkor, ha Nóri 175 cm magas? A legtöbb ember vonakodna „igent” vagy „nem” mondani, és inkább azt választaná, hogy „olyasmi”. Látható, hogy ez nem egy különböző világra vonatkozó bizonytalanság – mi biztosak vagyunk Nóri magasságában. A probléma az, hogy a „magas” nyelvi kifejezés nem az objektumok éles kettéosztását jelenti – a magasságnak fokozatai vannak. Ezért a legtöbb szerző szerint a fuzzy halmazelmélet *egyáltalán nem használható* bizonytalansági következtetésekre. Ehelyett a fuzzy halmazelmélet a *MagasSzemély*-t egy fuzzy predikátnak tekinti, aminek az igazságértéke inkább egy 0 és 1 közötti érték, mint hogy *igaz* vagy *hamis* lenne. A „fuzzy halmaz” név a predikátum értelmezéséből származik, ahogyan impliciten definíálja a halmaz tagjait – egy halmazt, aminek nincsenek éles határai.

A **fuzzy logika** (*fuzzy logic*) egy eljárás olyan logikai kifejezésekkel való következetésre, amelyek fuzzy halmazbeli tagsági állításokat írnak le. Például annak az összetett kifejezésnek, hogy *MagasSzemély(Nóri)*  $\vee$  *Nehéz(Nóri)* van fuzzy igazságértéke, ami a kifejezésben lévő komponensek igazságértékeinek a függvénye. Egy összetett



kifejezés  $T$  fuzzy igazságértékének a kiértékelésére a következő alapvető szabályok szolgálnak:

$$\begin{array}{lll} T(A \wedge B) & = & \min(T(A), T(B)) \\ T(A \vee B) & = & \max(T(A), T(B)) \\ T(\neg A) & = & 1 - T(A) \end{array}$$

A fuzzy logika tehát egy igazságfüggvényen alapuló rendszer, és ez komoly nehézségeket okoz. Tegyük fel például, hogy  $T(\text{MagasSzemély}(Nóri)) = 0,6$  és  $T(\text{Nehéz}(Nóri)) = 0,4$ . Ekkor azt kapjuk, hogy  $T(\text{MagasSzemély}(Nóri) \wedge \text{Nehéz}(Nóri)) = 0,4$ , ami elfogadhatónak tűnik, de az is adódik, hogy  $T(\text{MagasSzemély}(Nóri) \wedge \neg \text{MagasSzemély}(Nóri)) = 0,4$ , ami már nem. Világos, hogy a probléma amiatt lép fel, hogy az igazságfüggvényen alapuló megközelítés nem képes kezelní a pozitív és negatív korrelációkat az állítás részei között.

A **fuzzy szabályozás (fuzzy control)** egy metodológia olyan szabályozási rendszerek alkotására, amelyekben a valós értékű bemenetek leképezését a kimeneti paramétereikre fuzzy szabályok reprezentálják. A fuzzy szabályozás nagyon sikeresnek bizonyult olyan kereskedelmi termékekben, mint az automatikus sebességváltók, a videokamerák és a villanyborotvák. A kritikusok [lásd például (Elkan, 1993)] úgy érveltek, hogy ezek az alkalmazások azért sikeresek, mert kicsi a szabálybázisuk, nem végeznek többelépes következetést, és hangolható paramétereik vannak, amelyek állításával a rendszer teljesítménye javítható. Az a tény, hogy fuzzy műveleteket használnak, mellékes a sikereik szempontjából; a magyarázat egyszerűen az, hogy egy tömör és szemléletes módszert biztosítanak egyenletesen interpoláló, valós értékű függvények megadására.

Voltak arra irányuló kísérletek, hogy a fuzzy logikát a valószínűsg-számítás keretein belül értelmezzék. Az egyik elgondolás az, hogy az olyan kijelentéseket, mint „Nóri magas” úgy fogjuk fel, mint egy folytonos rejtett változó, Nóri valodi Magasságának diszkrét megfigyeléseit. A valószínűségi modell meghatározza a  $P(\text{„Megfigyelő állítja, hogy Nóri magas”}| \text{Magasság})$  valószínűséget, például az 591. oldalon leírt probit eloszlás (probit distribution) felhasználásával. Ekkor az a posteriori eloszlás Nóri magassága felett a megszokott módon számolható ki, például ha a modell része egy hibrid Bayes-hálónak. Egy ilyen megközelítés természetesen nem igazságfüggvényen alapuló. Például a feltételes eloszlás

$$P(\text{„Megfigyelő állítja, hogy Nóri magas”} | \text{Magasság}, Síly)$$

kapcsolatot enged meg a magasság és a síly között, ahogyan a megfigyelést okozzák. Így nagyon valószínűtlen, hogy valakit, aki két és fél méter magas és kilencven kilogramm, „magas és nehéz”-nek nevezünk, bár a „két és fél méter” „magas”-nak és a „kilencven kilogramm” „nehéz”-nek minősül.

A fuzzy predikátumoknak szintén adható egy valószínűségi értelmezés a **valószínűségi halmazokat (random sets)** felhasználva – azaz olyan valószínűségi változókat, amiknek lehetséges értékei objektumok halmazai. Például a *MagasSzemély* egy valószínűségi halmaz, amelynek lehetséges értékei emberek halmazai. A  $P(\text{Magas} = S_1)$  valószínűség, ahol  $S_1$  az embereknek valamely konkrét halmaza, annak a valószínűsége, hogy pontosan ezt a halmazt minősíténé „magas”-nak a megfigyelő. Ekkor a „Nóri magas” valószínűsége minden olyan halmaz valószínűségének az összege, amelyben Nóri elemként jelen van.

Mind a hibrid Bayes-hálós megközelítés, mind a valószínűségi halmaz megközelítés úgy tűnik, hogy modellezzi a meghatározatlanság egyes vetületeit bizonyosságok felhasználása nélkül. Mindazonáltal számos megválaszolatlan kérdés van a nyelvi megfigyelések és folytonos mennyiségek helyes reprezentációjával kapcsolatban, amely kérdezésekkel a fuzzy kutatói közösségen kívül a többség nem foglalkozik.

## 14.8. ÖSSZEFoglalás

Ez a fejezet a Bayes-hálókat (*Bayesian networks*) mutatta be, amelyek egy jól kidolgozott reprezentációt jelentenek a bizonytalan tudás számára. A Bayes-hálók nagyjából hasonló szerepet töltenek be, mint amit az ítéletlogika a biztos tudás számára.

- A Bayes-háló egy irányított körmentes gráf, aminek csomópontjai valószínűségi változókhöz tartoznak; minden csomóponthoz tartozik egy feltételes eloszlás, ahol a feltételt a csomópont szülei jelentik.
- A Bayes-hálók tömör módot adnak a tárgyterület **feltételes függetlenségi (conditional independence)** kapcsolatainak a reprezentálására.
- Egy Bayes-háló egy teljes együttes eloszlást specifikál; egy együttes bejegyzést a megfelelő lokális feltételes eloszlásokbeli bejegyzések szorzata határozza meg. Egy Bayes-háló gyakran exponenciálisan kisebb méretű, mint a teljes együttes eloszlás.
- Számos feltételes eloszlást lehet tömören reprezentálni eloszlások kanonikus családjával. A **hibrid Bayes-hálók (hybrid Bayesian networks)**, amelyek mind diszkrét, mind folytonos változókat tartalmaznak, sokféle kanonikus eloszlást használnak.
- A következtetés Bayes-hálókban a célváltozók egy halmaza valószínűség-eloszlásának kiszámítását jelenti, feltéve a tényváltozók egy halmazát. Az egzakt következtetési algoritmusok, mint a **változó eliminálás (variable elimination)** feltételes valószínűségek szorzatainak összegét értékelik, amilyen hatékonyan csak lehet.
- **Polifákban (polytrees)** (egyszeresen összekötött hálókban) az egzakt következtetés számítási ideje a háló méretével lineárisan nő. Általános esetben a probléma kezelhetetlen.
- Sztochasztikus közelítő számítási technikák, mint a **valószínűségi súlyozás (likelihood weighting)** és a **Markov lánc Monte Carlo (Markov chain Monte Carlo)** módszerek elfogadható becsléseket adnak a hálóbeli valódi a posteriori valószínűségekre, és sokkal nagyobb hálókkal is megbirkóznak, mint az egzakt algoritmusok.
- A valószínűség-számítás kombinálható az elsőrendű logikából vett reprezentációs ötletekkel, hogy nagyon hatékony rendszereket hozzunk létre bizonytalanság esetén történő következtetéshez. A relációs valószínűségi modellek (RVM-ek) eleget tesznek olyan reprezentációs megkötéseknek, amelyek egy olyan jól definiált valószínűség eloszlást garantálnak, amely egy ekvivalens Bayes-hálóval kifejezhető.
- Számos egyéb rendszert is javasoltak a bizonytalanság esetén történő következtetéshez. Nagy általánosságban azt lehet mondani, hogy az **igazságfüggvényen (truth-functional)** alapuló rendszerek az ilyen érveléshez nem a legmegfelelőbb eszközök.

## Irodalmi és történeti megjegyzések

A hálók felhasználása valószínűségi információk reprezentálására Sewall Wright munkájával kezdődött el a 20. század elején a genetikai öröklődés analízisével és az állatok növekedési tényezőivel kapcsolatban (Wright, 1921; 1934). Az ő hálóinak egyike látszik ennek a könyvnek a borítóján. I. J. Good, Alan Turinggal együttműködve, olyan valószínűségi reprezentációkat és Bayes következtetési módszereket fejlesztett ki, amelyeket a modern Bayes-hálók előfutárának lehet tekinteni – bár a cikkre nem hivatkoznak gyakran ebben a kontextusban (Good, 1961).<sup>13</sup> Ugyanez a cikk az eredeti forrása a zajos-VAGY modellnek.

A döntési problémák **hatásdiagram** (**influence diagram**) reprezentációját – ami a valószínűségi változók reprezentációjához egy irányított körmentes gráfot tartalmazott – döntéselemezések során használták az 1970-es évek vége felé (lásd 16. fejezet), de a kiértékelés csak felsorolást használtak. Judea Pearl fejlesztette ki az üzenetváltó eljárást, a fa hálókban (Pearl, 1982a) és az egyszeresen összekötött (polifa) hálókban (Kim és Pearl, 1983) történő következtetésre, és szemben az akkor népszerű bizonyossági tényezős rendszerekkel, a diagnosztikai helyett a kazuális valószínűségi modellek létrehozásának fontossága mellett érvelt. Az első szakértői rendszer, ami Bayes-hálókat használt, a CONVINCE volt (Kim, 1983; Kim és Pearl, 1987). A későbbi rendszerek között találjuk a mozgatóidegekkel kapcsolatos rendellenességek diagnosztizálására szolgáló MUNIN (Andersen és társai, 1989) és a patológiai helyzetek felderítésére szolgáló PATHFINDER rendszert (Heckerman, 1991). Messze a legtöbbet használt Bayes-hálós rendszerek a Microsoft Windows diagnóziskijavítás moduljai (például a Nyomtató vázárló) (Breese és Heckerman, 1996) és az Office segéd a Microsoft Office-ban (Horvitz és társai, 1998).

Általános Bayes-hálókban történő egzakt következtetésre Pearl fejlesztett ki egy csoportosításon alapuló algoritmust, felhasználva egy konverziót változók csoportjai feletti irányított polifába, amelyben üzenetváltások biztosítják a csoportok közös változói feletti konziszenciát. Egy hasonló módszer, amit David Spiegelhalter és Steffen Lauritzen fejlesztett ki (Spiegelhalter, 1986; Lauritzen és Spiegelhalter, 1988), egy irányítatlan Markov-hálóba vivő konverzió alapul. Az eljárást a HUGIN rendszerben valósították meg, ami egy hatékony és széles körben használt eszköz a bizonytalansági következtetések esetén (Andersen és társai, 1989). Ross Shachter, a hatásdiagramok kutatói közösségeből, egy egzakt módszert fejlesztett ki, ami a háló célirányos csökkentésén alapul a posteriori megőrző átalakításokat felhasználva.

A fejezetben leírt változó eliminációs módszer Shachter módszeréhez áll legközelebb, amiből a szimbolikus valószínűségi következtetés (SzVK) módszere alakult ki (Shachter, 1990). Az SzVK megkísérli optimalizálni az olyan kifejezési fák kiértékelését, mint amilyen a 14.8. ábrán látható. Az általunk ismertetett algoritmus a Zhang és Poole által kifejlesztetthez áll a legközelebb (Zhang és Poole, 1994; 1996). Irreleváns változók eltávolítására szolgáló kritériumokat Geiger és társai, illetve Lauritzen és társai fejlesztettek ki (Geiger és társai, 1990; Lauritzen és társai, 1990); az általunk

<sup>13</sup> I. J. Good vezető statisztikus volt Turing kódfejtő csapatában a második világháború során. A 2001: úrodísszeia c. művében Arthur C. Clarke Goodnak és Minskynek tulajdonította azokat az áttöréseket, amelyek a HAL számítógép kifejlesztéséhez vezettek (Clarke, 1968a).

megadott kritérium ezeknek egy egyszerű speciális esete. Rina Dechter (Dechter, 1999) mutatta meg, hogy a változó eliminálás ötlete lényegében megegyezik a **nem folytatós dinamikus programozással (nonserial dynamic programming)** (Bertele és Brioschi, 1972). Ez egy algoritmikus megközelítés, amit Bayes-hálókban előálló következetési problémák széles körére lehet sikerrel alkalmazni. Ilyen probléma például megfigyelések egy adott halmazához a legvalószínűbb magyarázat megkeresése. Ez összekapcsolja a Bayes-hálós algoritmusokat a kényszerkielégítési problémákat megoldó kapcsolódó módszerekkel, és közvetlen módszereket szolgáltat az egzakt következetés komplexitására a háló hipergráfjának a szélessége alapján.

Folytonos valószínűségi változók jelenlétéit Bayes-hálókban Pearl, illetve Shachter és Kenley vizsgálta (Pearl, 1988; Shachter és Kenley, 1989); ezek a cikkek kizárolag folytonos változókat tartalmazó hálókat tárgyalnak, amelyek lokális függési modelljei lineáris normális eloszlásúak. Diszkrét változók vezetését Lauritzen és Wermuth vizsgálta (Lauritzen és Wermuth, 1989), amit a cHUGIN rendszerben meg is valósítottak (Olesen, 1993). A probit eloszlást először Finney vizsgálta (Finney, 1947), sigmoid eloszlásnak nevezve. Az eloszlást széles körben használják diszkrét választási jelenségek modellezésére, és kiterjeszhető több mint két választási lehetőség kezelésére (Daganzo, 1979). A logit eloszlás használatához Bishop ad indoklást (Bishop, 1995).

Cooper mutatta meg, hogy a következetés általános problémája tetszőleges Bayes-hálóban NP-teljes (Cooper, 1990), Paule Dagum és Mike Luby pedig azt, hogy ennek a közelítő megoldása is NP-teljes (Dagum és Luby, 1993). A tárkomplexitás szintén komoly probléma mind a csoportosító, mind a változó eliminációs módszereknél. A **vágóhalmaz feltételezésekben (cutset conditioning)** alapuló eljárás, amit a kényszerkielégítési problémákra az 5. fejezetben mutattunk be, elkerüli az exponenciális méretű táblák létrehozását. Egy Bayes-hálóban a vágóhalmaz a csomópontok azon halmaza, amelyek értékeinek rögzítése esetén a fennmaradó csomópontok függései egy fagráfra egyszerűsödnek, ami már lineáris időben és téren megoldható. A lekérdezést összegzéssel adhatjuk meg a vágóhalmaz minden lehetséges értékadása mellett, így az összes tárígyen még mindig lineáris (Pearl, 1988). Darwiche (Darwiche, 2001) egy rekurzíván feltételező algoritmust ad meg, ami tetszőleges tár/idő egyensúlyt tesz lehetővé.

Gyors közelítő algoritmusok Bayes-hálókra való kifejlesztése nagyon aktív terület, a statisztika, a számításelmélet és a fizika határterületén. Az elutasító mintavételezés módszere egy általános, a statisztikában régebben ismert technika; Bayes-hálóknál Max Henrion alkalmazta először, aki **logikai mintavételezésnek (logic sampling)** nevezte (Henrion, 1988). A valószínűségi súlyozás, amit Fung és Chang (Fung és Chang, 1989), illetve Shachter és Peot fejlesztettek ki (Shachter és Peot, 1989), egy példája a **fontossági mintavétel (importance sampling)** jól ismert statisztikai módszerének. A valószínűségi súlyozás nagymérvű felhasználását az orvosi diagnosztikában Shwe és Cooper jelentí (Shwe és Cooper, 1991). Cheng és Druzdzel a valószínűségi súlyozás adaptív verzióját közlik (Cheng és Druzdzel, 2000), ami akkor is jól működik, ha a bizonyíték a priori valószínűsége nagyon alacsony.

A Markov lánc Monte Carlo (MCMC) módszerek a Metropolis algoritmussal kezdődtek, ami Metropolisnak köszönhető (Metropolis és társai, 1953), ami a 4. fejezetben leírt szimulált lehűtés algoritmusának szintén a forrása. A Gibbs-mintavételezést Geman és Geman fejlesztette ki irányítatlan Markov-hálókban történő következetésre (Geman és Geman, 1984). Az MCMC Bayes-hálókban történő alkalmazása Pearlnek köszönhető

(Pearl, 1987). A Gilks és társai által összegyűjtött cikkek MCMC-alkalmazások széles skáláját ölelik fel, számos közülük a jól ismert BUGS csomaggal lett kifejlesztve (Gilks és társai, 1996).

A közelítési módszereknek két nagyon fontos családja létezik, amelyeket nem tárgyalunk a fejezetben. Az első család a **variációs közelítés (variational approximation)** módszere, ami mindenfajta komplex számítás egyszerűsítésére felhasználható. Az alapötlet az, hogy az eredeti problémának előállítjuk egy olyan redukált verzióját, ami annyira egyszerű, hogy dolgozni lehessen vele, de amennyire lehet hasonló az eredeti problémához. A redukált probléma leírása tartalmazza a  **$\lambda$  variációs paramétereit (variational parameters)**, amelyeket úgy állítunk be, hogy minimalizáljon egy  $D$  távolságfüggvényt az eredeti és a redukált problémák között, gyakran a  $\partial D / \partial \lambda = 0$  egyenletrendszer megoldásával. Számos esetben pontos felső és alsó korlátok kaphatók. Variációs módszereket régóta használnak a statisztikában (Rustagi, 1976). A statisztikus fizikában a **mezőátlag (mean field)** módszer egy speciális variációs módszer, amelyben a modellt alkotó egyes változókat teljesen függetlennek tételezzük fel. Ezt az ötletet felhasználva sikerült kezelní nagy irányítatlan Markov-hálókat (Peterson és Anderson, 1987; Parisi, 1988). A variációs módszerek Bayes-hálókra történő alkalmazásának matematikai megalapozását Saul és társai fejlesztették ki, és ők kaptak a sigmoid hálókra pontos alsó korlát közelítéseket a mezőátlag módszerrel (Saul és társai, 1996). Jaakkola és Jordan úgy terjesztették ki a módszert (Jaakkola és Jordan, 1996), hogy mind az alsó, mind a felső korlát kinyerhető legyen. A variációs módszereket Jordan tekintette át (Jordan, 1999).

A közelítő algoritmusok második családja Pearl fagráfon futó üzenetváltásos algoritmusán alapszik (Pearl, 1982a). Az algoritmust általános hálókra is lehet alkalmazni, ahogy Pearl javasolta (Pearl, 1988). Az eredmények lehetnek helytelenek, és lehet, hogy az algoritmus nem áll le, de számos esetben a szolgáltatott értékek közel vannak a helyes értékekhez. Ez az úgynevezett **bizonyosságterjesztés (belief propagation)** vagy **hurkos terjesztés (loopy propagation)** megközelítés kevés figyelmet kapott, mindaddig, amíg McEliece és társai fel nem figyeltek (McEliece és társai, 1998) arra, hogy az üzenetváltás többszörösen összekötött Bayes-hálókban pontosan megegyezik a **turbó dekódolás (turbo decoding)** algoritmus által végzett számítással (Berrou és társai, 1993), ami jelentős áttörést eredményezett a hibajavító kódok tervezésében. Ebből az a következtetés vonható le, hogy a bizonyosságterjesztés gyors is és pontos is a nagyon nagy méretű és nagyon sűrűn összekötött dekódolásra használt hálókban, és ezért lehet, hogy általánosabban is hasznos lehet. Murphy és társai egy kísérletről számoltak be, ahol ez valóban működött (Murphy és társai, 1999). Yedidia és társai további kapcsolatokra mutattak rá a bizonyosságterjesztés és a statisztikus fizika között (Yedidia és társai, 2001).

A valószínűség és az elsőrendű nyelvek közötti kapcsolatot Carnap tanulmányozta (Carnap, 1950). Gaifman, illetve Scott és Krauss megadott egy nyelvet, amelyben valószínűségeket elsőrendű mondatokhoz lehet kapcsolni, és amelyre a modellek valószínűségi mértékek voltak lehetséges világokon (Gaifman, 1964; Scott és Krauss, 1966). Az MI-n belül ezt az ötletet az ítéletlogikára Nilsson, a predikátumlogikára pedig Halpern fejlesztette ki (Nilsson, 1986; Halpern, 1990). A tudásreprezentációs kérdések első kiterjedt vizsgálatát ilyen nyelvek esetén Bacchus végezte (Bacchus, 1990), Wellman és társainak a tanulmánya a korai implementációkat tekinti át (Wellman és társai, 1992),

amelyek egy ekvivalens (ítéletlogikai) Bayes-háló létrehozásán alapulnak. Napjainkra a kutatók elkezdték megérteni a teljes tudásbázisok fontosságát, azaz olyan tudásbázisok fontosságát, amelyek akár a Bayes-hálók, egy egyértelmű együttes eloszlást definiálnak az összes lehetséges világ felett. Az erre szolgáló eljárások a logikai programozás valószínűségi verzióin (Poole, 1993; Sato és Kameya, 1997) vagy szemantikus hálókon (Koller és Pfeffer, 1998) alapultak. A fejezetben ismertetett típusú relációs valószínűségi modelleket Pfeffer vizsgálta részletesebben (Pfeffer, 2000). Pasula és Russell mind a relációs, mind az azonossági bizonytalanságot vizsgálta az RVM-ekben és az MCMC következtetés felhasználásában (Pasula és Russell, 2001).

Amint a 13. fejezetben kifejtettük, a korai valószínűségi rendszerek kegyvesztettek lettek az 1970-es években, helyt adva az alternatív módszerek megjelenésének. A bizonyossági tényezőket a MYCIN orvosi szakértői rendszerbeli felhasználásra fejlesztették ki (Shortliffe, 1976), amit egyrészt mérnöki megoldásnak is, másrészről az emberi ítélezéshez használt modelljének is javasolták, bizonytalanság esetében. A *Rule-Based Expert Systems* gyűjtemény (Buchanan és Shortliffe, 1984) teljes áttekintést ad a MYCIN-ról és leszármazottairól (lásd még [Stefik, 1995]). David Heckerman megmutatta, hogy a bizonyossági tényezős számítások kicsit módosított verziójára már helyes valószínűségeket ad bizonyos esetekben, de más esetekben a bizonyítékok hibát okozó, túlzott figyelembevételére vezet (Heckerman, 1986). A PROSPECTOR szakértői rendszer (Duda és társai, 1979) szabályalapú megközelítést használt, amiben a szabályokat (ritkán tartható) globális függetlenségi feltételezésekkel igazolták.

A Dempster–Shafer-elmélet Arthur Dempster publikációjával kezdődött, amiben a pontszerű valószínűségi értékek általánosítását javasolta intervallumértékekre, és szabályokat javasolt a kombinálásukra (Dempster, 1968). Glenn Shafer későbbi munkája vezetett el a Dempster–Shafer-elmélet és a valószínűség-számítás egymással versengő szemléletéhez (Shafer, 1976). Ruspini a Dempster–Shafer-elmélet és a valószínűség-számítás kapcsolatát elemzte (Ruspini és társai, 1992). Shenoy a Dempster–Shafer bizonyosságfüggvény alapján egy eljárást javasolt a döntéshozatalra (Shenoy, 1989).

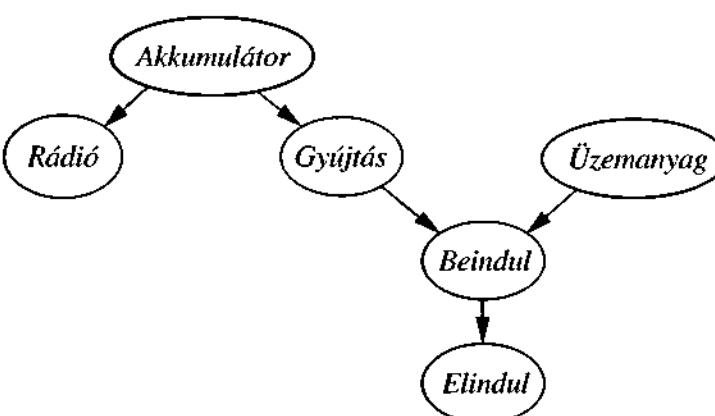
A fuzzy halmazokat Lotfi Zadeh dolgozta ki válaszul arra a nehézségre (Zadeh, 1965), hogy egzakt bemeneti értékeket adjunk az intelligens rendszerek számára. Zimmermann leírása (Zimmermann, 2001) részletes bevezetést nyújt a fuzzy halmazok elméletébe; a (Zimmermann, 1999) pedig fuzzy alkalmazásokról szóló cikkek gyűjteménye. Amint korábban említettük, a fuzzy logikát gyakran tekintik a valószínűség-számítás versenytársának, pedig valójában eltérő kérdésekkel foglalkozik. A lehetőséglemelet (possibility theory) (Zadeh, 1978) a bizonytalanság kezelésére vezették be fuzzy rendszerekben, és sok közös vonása van a valószínűség-számítással. Dubois és Prade alapos áttekintést nyújt a lehetőséglemelet és a valószínűség-számítás kapcsolatáról (Dubois és Prade, 1994).

A valószínűség-számítás MI-n belüli feltámadása főként a Bayes-hálók felfedezésén múlt, ami módszert adott a feltételes függetlenségek reprezentálására és kihasználására. Ez az újjászületés igen küzdelmes volt; Peter Cheeseman harcias *In defense of Probability* (Cheeseman, 1985) és a későbbi *An Inquire into Computer Understanding* (Cheeseman, 1988, megjegyzésekkel) c. cikke ízelítőt ad a vitából. A logika művelőinek egyik legfőbb kifogása az volt, hogy a valószínűség-számítás miatt szükségesnek vélt számítások önelemzéssel nem érhetők el, és egy nem reális pontossági szintet tételeznek fel a bizonytalan tudásunkban. A kvalitatív Bayes-hálók (qualitative probabilistic

**networks)** (Wellman, 1990a) lehetőséget adnak a Bayes-hálók tisztán kvalitatív absztraktiójára, csupán a változók közötti hatások pozitív és negatív jellegét kihasználva. Wellman megmutatta, hogy számos esetben ennyi információ is elegendő az optimális döntéshozatalhoz a valószínűségi értékek precíz meghatározása nélkül. Adnan Darwiche és Matt Ginsberg munkája kivonatolja a valószínűség-számítás elméletében szerepelő feltételesség és a tények kombinálásának alapvető tulajdonságait, és megmutatja, hogyan lehet ezeket alkalmazni logikai és alapértelmezési következetésekben (Darwiche és Ginsberg, 1992).

A fejezetben leírt szívbetegség-kezelő rendszer Lucastól származik (Lucas, 1996). A Bayes-hálók más területen történő alkalmazásai között találjuk egyebek között a Microsoftnál végzett fejlesztéseket a felhasználó céljainak kikövetkeztetésére (Horvitz és társai, 1998) és a kéretlen elektronikus levelek szűrésére (Sahami és társai, 1998), az Electric Power Research Institute fejlesztését áramgenerátorok figyelésére és a NASA fejlesztését időkritikus információk megjelenítésére a Mission Controlnál Houstonban (Horvitz és Barry, 1995).

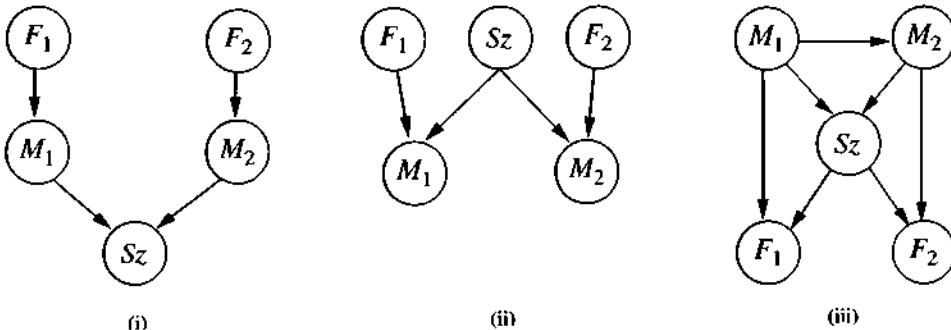
Az MI-n belüli bizonytalansági következtetéssel kapcsolatos számos fontos korai publikáció megtalálható a *Readings in Uncertain Reasoning* (Shafer és Pearl, 1990) és az *Uncertainty in Artificial Intelligence* (Kanal és Lemmer, 1986) antológiákban. A Bayes-hálók témakörének fejlődésében a legfontosabb egyedülálló publikáció vitathatatlanul a *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Pearl, 1988). Több kiváló munka tartalmazza az újabb anyagokat, mint például (Lauritzen, 1996; Jensen, 2001; Jordan, 2003). A valószínűségi következtetéssel kapcsolatos új eredmények mind az MI fő irányait meghatározó főbb lapokban, mint pl. az *Artificial Intelligence*-ben, mind a specializáltabb folyóiratokban, mint amilyen az *International Journal of Approximate Reasoning* jelent meg. A gráfos modellek ről (graphical models), amely modellosztály magában foglalja a Bayes-hálókat is, számos cikk statisztikai újságokban jelenik meg. Az *Uncertainty in Artificial Intelligence* (UAI), a *Neural Information Processing Systems* (NIPS) és az *Artificial Intelligence and Statistics* (AISTATS) konferenciák kiadványai kiváló forrásai a legfrissebb kutatásoknak.



**14.18. ábra.** Egy gépkocsi elektromos és motorikus rendszereinek részeit leíró Bayes-háló. Minden változó bináris, és az igaz érték jelzi a megfelelő funkció helyes működését.

## Feladatok

- 14.1.** Tekintsük a 14.18. ábrán látható gépkocsi-diagnosztikai hálót.
- Egészítse ki a hálót a *JegesIdő* és az *IndítóMotor* bináris változókkal.
  - Adj meg ésszerű feltételes valószínűségi táblákat az összes csomópontra.
  - Hány független értéket tartalmaz a teljes együttes valószínűség-eloszlás függvény 8 bináris csomópont esetén, feltételezve, hogy nincs közöttük feltételes függetlenségi reláció?
  - Hány független valószínűségi értéket tartalmaznak az ön hálójának táblázatai?
  - A *Beindul* feltételes valószínűség eloszlása megadható egy zajos-ÉS eloszlással. Írja le ezt az eloszláscsaládot általában és kapcsolatát a zajos-VAGY eloszláshoz.
- 14.2.** A helyi nukleáris erőműben van egy riasztó, ami érzékeli, ha a hőmérsékletmérő egy adott küszöbértéket meghalad. A mérő a reaktormag hőmérsékletét méri. Tekintse a következő bináris változókat:  $R$  (riasztó hangja),  $H_R$  (hibás riasztó),  $H_M$  (hibás mérő) és a következő többértékű változókat:  $M$  (mérő) és  $H$  (egy aktuális belső hőmérséklet).
- Rajzoljon fel egy Bayes-hálót a tárgytartományra, feltételezve, hogy valószínűbb, hogy a mérő akkor hibásodik meg, ha a hőmérséklet túl magas.
  - A hálózata polifa lett?
  - Tételezze fel, hogy a hőmérsékletnek csak két lehetséges értéke van: normális és magas; és hogy a mérő helytelen hőmérsékletet mér az esetek  $x\%-ában$ , ha működik, és  $y\%-ában$ , ha hibás. Adja meg az  $M$  feltételes valószínűségtábláit.
  - Tételezze fel, hogy a riasztó tökéletesen működik, hacsak nem hibás, mely utóbbi esetben egyáltalán nem riaszt. Adja meg az  $R$ -hez tartozó feltételes valószínűség-táblát.
  - Tételezze fel, hogy a mérő és a riasztó működik, és a riasztó megszólal. Fejezze ki annak a valószínűségét, hogy a reaktormag hőmérséklete magas a háló feltételes valószínűségeinek a függvényében.
- 14.3.** Két csillagász – távcsöveiket használva – a világ különböző részein  $M_1$  és  $M_2$  méréseket végeznek az égbolt egy kis részén látható csillagok számáról ( $Sz$ ). Általában egy kis  $e$  valószínűsége a hibának, ami egy csillagnyi eltérést jelent. Bárminelyik távcsővel megtörténhet az is (egy sokkal kisebb  $f$  valószínűséggel), hogy nincs rendesen fókuszálva ( $F_1$  és  $F_2$  események), amely esetben a tudós 3 vagy több csillaggal kevesebb csillagot számlál (ha  $Sz$  kisebb mint 3, akkor egyetlen csillagot sem észlel). Tekintsük a 14.19. ábrán látható három hálót.
- Melyik Bayes-háló reprezentálja helyesen (de nem feltétlenül hatékonyan) a fenti ismereteket?
  - Melyik a legjobb háló? Adj meg magyarázatot.
  - Adj meg egy ésszerű feltételes valószínűségi táblát a  $P(M_1|Sz)$  értékeire, amikor  $M_1 \in \{0, 1, 2, 3, 4\}$  és  $Sz \in \{1, 2, 3\}$ . A feltételes eloszlás minden értékét az  $e$  és az  $f$  paraméterek függvényében fejezze ki.



**14.19. ábra.** Három lehetséges háló a távcsőproblémára

- (d) Tételezz fel, hogy  $M_1 = 1$  és  $M_2 = 3$ . Mik a csillagok lehetséges számai, ha a priori nem korlátozzuk Sz értékét?

(e) Mi a csillagok *legvalószínűbb* száma ezen megfigyelések esetén? Magyarázza el ennek kiszámítását, vagy ha nem lehetséges kiszámítani, magyarázza el, milyen további információkra van szükség, és hogy az miként befolyásolná az eredményt.

**14.4.** Tekintsük a 14.19. (ii) ábrán látható hálót, és tegyük fel, hogy a két teleszkóp azonosan működik. Legyen  $M_1, M_2 \in \{0, 1, 2, 3, 4\}$  és  $Sz \in \{1, 2, 3\}$  a 14.3. feladatban megadott szimbolikus FVT-kkel. A felsoroló algoritmus felhasználásával számolja ki a következő valószínűség-eloszlást:  $P(Sz|M_1 = 2, M_2 = 2)$ .

**14.5.** Tekintsük a lineáris Gauss lokális eloszlású hálók családját, amit az 589. oldalon illusztrál.

(a) Egy kétváltozós hálóban legyen  $X_1$  az  $X_2$  szülője,  $X_1$ -nek legyen normális a priori eloszlása,  $P(X_2|X_1)$  pedig egy lineáris normális eloszlás. Mutassa meg, hogy a  $P(X_1, X_2)$  együttes eloszlás egy többváltozós Gauss-eloszlás, és számolja ki ennek a kovarianciamátrixát.

(b) Bizonyítsa be indukcióval, hogy egy általános lineáris Gauss-háló  $X_1, \dots, X_n$ -en vett együttes eloszlása szintén többváltozós Gauss-eloszlás.

**14.6.** Az 591. oldalon definiált probit eloszlás egy bináris gyermek valószínűség-eloszlását írja le, egyetlen folytonos szülő esetén.

(a) Hogyan terjeszthető ki a definíció több folytonos szülő esetére?

(b) Hogyan terjeszthető ki, hogy többéretékkü gyermekváltozóra is alkalmazható legyen? Gondolja végig mind a két esetet, amikor a gyermek értékei sorrendezettek (ahogyan a vezetésnél a sebességfokozat megválasztása a sebesség, a lejtés, a kívánt gyorsulás stb. függvénye), és amikor nem állíthatók sorba (mint például a busz, a vonat vagy a kocsi választása munkába menetelnél). (Segítség: gondoljunk a lehetséges értékek két csoportra osztására, hogy bináris változót szimuláljunk.)

**14.7.** Ez a feladat a 14.10. ábrán látható változó eliminációs algoritmusra irányul.

- (a) A 14.4. alfejezet a változó eliminálást alkalmazza arra a lekérdezésre, hogy

$$P(\text{Betörés} | \text{JánosTelefonál} = \text{igaz}, \text{MáriaTelefonál} = \text{igaz})$$

Végezze el a jelzett számításokat, és ellenőrizze, hogy helyes-e a válasz.

- (b) Számolja össze az elvégzett aritmetikai műveletek számát, és hasonlítsa össze a felsorolási algoritmus által elvégzettek számával.
- (c) Tegyük fel, hogy a háló egy láncot alkot: bináris változók szekvenciáját, ahol  $Szüleök(X_i) = \{X_{i-1}\}$   $i = 2, \dots, n$  esetén. Mi a  $P(X_1 | X_n = \text{igaz})$  kiszámításának a komplexitása a felsorolást használva? És mi, ha változó eliminálást használunk?
- (d) Bizonyítsa be, hogy a változó eliminálás futásának komplexitása egyszere sen összekötött hálóban lineáris a háló méretében bármely, a háló struktúrájával konzisztens változó sorrendezés mellett.

**14.8.** Vizsgáljuk meg az egzakt következtetés komplexitását általános Bayes-hálókban.

- (a) Bizonyítsa be, hogy bármely 3-SAT probléma redukálható egy egzakt következtetésre egy olyan Bayes-hálóban, amely a konkrét problémát reprezentálja, és így az egzakt következtetés NP-nehéz. (Segítség: gondoljon egy olyan hálóra, amiben minden kijelentésszimbólumhoz, minden klózhöz és a klózok minden konjunktciójához rende egy-egy csomópont tartozik.)
- (b) A 3-SAT problémában a kielégítő érték-hozzárendelések számának meghatározása #P-teljes. Bizonyítsa be, hogy az egzakt következtetés legalább ennyire nehéz.

**14.9.** Tekintsük egy véletlen minta egy megadott egyváltozós eloszlásból történő generálásának a problémáját. Feltételezzük, hogy rendelkezésre áll egy véletlenszám-generátor, ami egyenletes eloszlás szerint 0 és 1 közötti véletlen számokat ad.

- (a) Legyen  $X$  egy diszkrét valószínűségi változó  $P(X_i = x_i) = p_i$   $i \in \{1, \dots, k\}$  tömegfüggvénnyel. Az  $X$  eloszlásfüggvénye (*cumulative distribution*) minden lehetséges  $j$ -re megadja annak valószínűségét, hogy  $X \in \{x_1, \dots, x_j\}$ . Magyarázza el, hogyan számolható ki az eloszlásfüggvény  $O(k)$  időben, és hogyan generálhatunk vele egy  $X$  eloszlása szerinti mintát. Ez utóbbi megtehető kevesebb mint  $O(k)$  időben?
- (b) Most tegyük fel, hogy  $N$  mintát szeretnénk generálni  $X$  szerint, ahol  $N \gg k$ . Magyarázza el, hogyan tehető ez úgy meg, hogy a mintánkénti várható futási idő *konstans* (azaz független  $k$ -tól).
- (c) Most tekintsünk egy folytonos értékű változót egy parametrikus eloszlással (például normálissal). Hogyan generálhatunk mintákat egy ilyen eloszlásból?
- (d) Tegyük fel, hogy egy folytonos értékű változót szeretne lekérdezni, és egy olyan mintavételező algoritmust használ a következtetésre, mint a VALÓSZÍNÜSÉGISÜLYOZÁS. Hogyan kellene módosítania a lekérdezés-válaszadás folyamatát?

- 14.10.** Egy változó **Markov-takaróját (Markov blanket)** az 586. oldalon definiáltuk.
- Bizonyítsa be, hogy a változó független a háló összes többi változójától, ha Markov-takarója ismert.
  - Vezesse le a (14.11) egyenletet.
- 14.11.** Tekintsük a  $P(Eső|Locsoló = \text{igaz}, VizesPázsit = \text{igaz})$  lekérdezést a 14.11. (a) ábra szerint, és annak az MCMC módszerrel való megválaszolását.
- Hány állapota van a Markov-láncnak?
  - Számítsa ki a **Q állapotátmenet-mátrixot (transition matrix Q)**, ami tartalmazza az összes  $q(y \rightarrow y')$  értéket minden  $y$ -ra és  $y'$ -ra.
  - Mit reprezentál  $Q^2$ , az állapotátemeneti mátrix négyzete?
  - Mi lesz  $Q^n$ , ha  $n \rightarrow \infty$ ?
  - Magyarázza el, hogy hajtsunk végre valószínűségi következtetést Bayes-hálókban, ha  $Q^n$  elérhető. Hatékony módja ez a következtetésnek?
- 14.12.** Hárrom focicsapat,  $A$ ,  $B$  és  $C$  játszik egymás ellen. minden meccsen két csapat vesz részt, az eredmény pedig győzelem, vereség vagy döntetlen lehet. minden csapatnak egy rögzített, ismeretlen fokozatú játékszintje van – ami egy 0 és 3 közötti egész érték –, és egy meccs eredménye a két csapat játékszintje közötti különbségtől függ sztochasztikusan.
- Hozzon létre egy relációs valószínűségi modellt a tárgyterületre, és javasoljon értékeket az összes szükséges valószínűség-eloszlásra.
  - Hozzon létre egy ekvivalens Bayes-hálót.
  - Tegyük fel, hogy az első két meccsen  $A$  megveri  $B$ -t,  $C$ -vel pedig döntetlen játszik. Egy tetszőleges egzakt következtetést felhasználva, számítsa ki a harmadik meccs a posteriori eloszlását.
  - Tegyük fel, hogy  $n$  csapat vesz részt a bajnokságon, és minden eredményünk megvan, kivéve az utolsót. Hogyan változik az utolsó meccs megjósításának a komplexitása az  $n$  függvényében?
  - Vizsgálja meg az MCMC alkalmazását ezen a problémán. Milyen gyorsan konvergál az MCMC a gyakorlatban, és hogyan skálázható fel?

# 15. IDŐBELI VALÓSZÍNŰSÉGI KÖVETKEZTETÉS

*Ebben a fejezetben megpróbáljuk értelmezni a jelent, megérteni a múltat és esetleg megjósolni a jövőt, még akkor is, ha igencsak kevésbé kristálytisztta.*

A bizonytalan környezetben lévő ágenseknek – a logikai ágensekhez hasonlóan – képesnek kell lenniük környezetük aktuális állapotának nyomon követésére. Ezt a feladatot nehezebbé teszi a részleges és zajos érzékelés, és az a bizonytalanság, ahogyan a környezet az idő előrehaladtával változik. Az ágens a legjobb esetben is a jelenlegi helyzetnek csak egy valószínűségi értékeléséhez képes hozzájutni. Ez a fejezet olyan reprezentációkat és következtetési algoritmusokat ír le, amelyek lehetővé teszik ezt az értékelést, a 14. fejezetben bemutatott ötletekre építve.

Az alapvető megközelítés leírása a 15.1. alfejezetben szerepel: egy változó világot úgy modellezünk, hogy a világ állapotának minden vonatkozására, *minden időpillanatban* egy valószínűségi változót használunk. Ezen változók közti kapcsolatok írják le az állapot fejlődését. A 15.2. alfejezet meghatározza az alapvető következtetési feladatokat, és leírja az időbeli modellekhez tartozó következtető algoritmusok általános struktúráját. Ezután három különbözőfajta modellt írunk le: a **rejtett Markov-modelleket** (**hidden Markov model**), a **Kalman-szűrőket** (**Kalman filters**) és a **dinamikus Bayes-hálókat** (**dynamic Bayesian networks**) (ami mint speciális aleseteket magában foglalja a rejtett Markov-modelleket és a Kalman-szűrőket). Végül a 15.6. alfejezet bemutatja, miként alkotják az időbeli valószínűségi modellek a modern beszédfelismerő rendszerek magját. Ezen modellek mindegyikének létrehozásában központi szerepet tölt be a tanulás, de a tanulási algoritmusok részletes vizsgálata a VI. részre marad.

## 15.1. IDŐ ÉS BIZONYTALANSÁG

A valószínűségi következtetésre szolgáló technikáinkat **változatlan (static)** világok esetén fejlesztettük ki, amelyekben minden egyes valószínűségi változónak egyetlen rögzített értéke van. Például egy gépkocsi javításánál feltesszük, hogy ami meghibásodott, az a diagnosztizálás alatt is hibás marad; a feladatunk a gépkocsi állapotának kikövetkeztetése a megfigyelt bizonyítékokból, amelyek szintén változatlanok maradnak.

Most vegyük fontolóra egy másféle problémát: egy cukorbeteg páciens kezelését. Akárcsak a gépkocsijavítás esetén, rendelkezünk bizonyítékokkal: a jelenlegi inzulin-adagok, az élelmiszer-bevitel, a vérekorszint mérésének eredményei és egyéb testi tünetek. A feladat a páciens jelenlegi állapotának értékelése, beleértve az aktuális vércukorszintet és inzulinszintet. Ezeknek az információknak az alapján az orvos (vagy a páciens) döntést hoz a páciens élelmiszer-beviteli és inzulinadagjáról.

A gépkocsijavítás esetétől eltérően, itt a probléma *dinamikai* vonatkozásai alapvetőek. A vércukorszintek és méréseik idővel gyorsan változhatnak, amit befolyásol a paciens aktuális élelmiszer-bevitel és inzulinadagja, az anyagszere aktivitása, a napszak és egyéb tényezők. Ahhoz, hogy a bizonyítékok időbeli alakulásából a jelenlegi állapotot megbecsüljük, és egy kezelés kimenetelét megjósoljuk, modellezünk kell ezeket a változásokat.

Ugyanezek a szempontok számos más esetben is jelentkeznek, egy nemzet gazdasági aktivitásának közelítő és részleges statisztikák alapján történő követésétől, beszédszekvenciák zajos és többértelmű akusztikai mérésekkel történő megértéséig. Adódik a kérdés: hogyan lehet az ezekhez hasonló dinamikai helyzeteket modellezni?

## Állapotok és megfigyelések

Az alapvető megközelítés, amit követünk, hasonló a 10. fejezetben leírt szituációkalkulus alapjáról szolgáló ötlethez: a változás folyamatát pillanatfelvételek sorozatának tekinthetjük, amelyek mindegyike egy adott pillanatban írja le a világ állapotát. minden pillanatfelvétel vagy **időpont** (*time slice*) valószínűségi változók egy halmazát tartalmazza, amelyek némelyike megfigyelhető, némelyike pedig nem.

Az egyszerűség kedvéért a továbbiakban fel fogjuk tenni, hogy a változóknak ugyanazz a részhalmaza minden időponthan megfigyelhető (bár az elkövetkezőkben ez teljes mértékben sehol sem szükségszerű). A  $t$  időpillanatban nem megfigyelhető változók halmazának a jelölésére  $X_t$ -t fogjuk használni, és  $E_t$ -t a megfigyelhető változók halmazának a jelölésére. A  $t$  időpontbeli megfigyelés  $E_t = e_t$  az értékek valamely  $e$ , halmazára.

Gondoljuk át a következő leegyszerűsített példát: tegyük fel, hogy egy titkos, földalatti létesítmény biztonsági őrei vagyunk. Szeretnénk tudni, hogy vajon aznap esik-e, de a külvilághoz való egyetlen hozzáférésünket az jelenti, hogy reggelenként látjuk, hogy az igazgató esernyővel vagy esernyő nélkül jön be. minden egyes  $t$  napon az  $E_t$  halmaz így egyetlen bizonyítékváltozót tartalmaz, az  $U_t$ -t (van-e esernyő), és  $X_t$ , az  $R_t$ -t, az egyetlen állapotváltozót tartalmazza (esik-e). Más problémákhöz változók nagyobb halmaza tartozhat. A cukorbetegség példájában a bizonyítékváltozók lehetnek a *MértVércukor*, és a *Pulzusszám*, és az állapotváltozók lehetnek a *Vércukor*, és a *Gyomortartalom*.<sup>1</sup>

Az időpontok közötti intervallum szintén problémafüggő. A cukorbetegség követésénél az alkalmas intervallum inkább egy óra, mint egy nap. Ebben a fejezetben általában egy rögzített véges intervallumot tételezünk fel, ami azt jelenti, hogy az időpillanatok egész számokkal felcímkézhetők. Feltesszük, hogy az állapotSOROZAT  $t = 0$ -nál kezdődik; és jelentéktelen okokból feltesszük, hogy a bizonyítékok  $t = 1$ -nél kezdenek beérkezni, és nem  $t = 0$ -nál. Így az esernyők világát az  $R_0, R_1, R_2, \dots$  állapotváltozók és az  $U_1, U_2, \dots$  bizonyítékváltozók reprezentálják. Az  $a : b$  jelölést fogjuk használni az egészek  $a$ -tól  $b$ -ig tartó sorozatának jelölésére (a határokat beleérte), és  $X_{a:b}$  jelöli a változók megfelelő halmazát  $X_a$ -tól  $X_b$ -ig. Például  $U_{1:3}$  az  $U_1, U_2, U_3$  változóknak felel meg.

<sup>1</sup> Vagyük észre, hogy a *Vércukor*, és a *MértVércukor* két különböző változó, így kezeljük az aktuális mennyiségek zajos mérését.

## Stacionárius folyamatok és a Markov-feltétel

Egy adott problémánál az állapotváltozók és a bizonyítékváltozók halmazának meghatározása után a következő lépés a változók közötti függőségek megadása. Követhetnénk a 14. fejezetben megállapított eljárást, valahogyan sorrendezve a változókat, és kérdéseket feltéve az elődöktől való feltételes függetlenségre, adott szülői halmaz esetén. Egy nyilvánvaló választás, hogy a változókat a természetes idősorrendjük szerint sorrendezzük, mivel az ok általában megelőzi a hatást, és a változókat lehetőleg az ok-okozati sorrendjük szerint vesszük.

Azonban hamar beleütközhetünk egy akadályba: a változók halmaza nem korlátos, mivel minden időpontra tartalmazza az állapot- és bizonyítékváltozókat. Valójában ez két problémát is felvet: (1) korlátlan számú feltételes valószínűségi táblát kell megadnunk minden változóra, minden időpillanatban; (2) ezek korlátlan számú szülőt tartalmazhatnának.

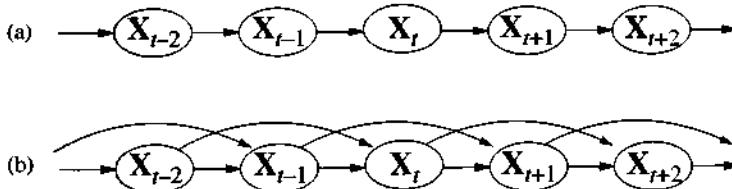
Az első problémát megoldja annak feltételezése, hogy a világ állapotának a változásait egy **stacionárius folyamat** (*stationary process*) okozza – azaz egy változási folyamat, amit olyan törvények határoznak meg, amik maguk nem változnak az idővel. (Ne keverjük össze a *stacionaritást* a *statikussággal*: egy *statikus* folyamatban maga az állapot nem változik.) Az esernyős világban ekkor a  $P(U_t|Szülők(U_i))$  feltételes valószínűség, hogy az esernyő feltűnik, minden  $t$ -re azonos. A stacionaritás feltevésénél ezért csak egy „*repräsentatív*” időpillanathoz tartozó változók feltételes valószínűségeit kell megadnunk.

A második problémát, a potenciálisan végtelen számú szülő kezelését, az úgynevezett **Markov-feltétel** (**Markov assumption**) elfogadása oldja meg – azaz, hogy a jelenlegi állapot a korábbi állapotoknak csak véges történetétől függ. Ennek a feltevésnek eleget tévő folyamatokat elsőként Andrei Markov orosz matematikus tanulmányozta részletesen. Ezeket a folyamatokat **Markov-folyamatoknak** (**Markov processes**) vagy **Markov-láncoknak** (**Markov chains**) neveznek. (A Markov-folyamatok feloszthatók aszerint, hogy az állapotter folytonos vagy diszkrét, illetve hogy az idő folytonos vagy diszkrét. Markov-láncoknak a diszkrét idejű és/vagy diszkrét állapotterű Markov-folyamatokat nevezik – *a ford.*) A Markov-folyamatok különböző jellemzőkkel rendelkezhetnek; a legegyszerűbb az **elsőrendű Markov-folyamat** (**first-order Markov process**), amelyben a jelenlegi állapot csak az előző állapototól függ, és nem függ egyetlen korábbitól sem. Máshogy fogalmazva, egyetlen állapot ismeretére van szükség, hogy a jövő a múlttól feltételesen független legyen az állapot ismeretében. A jelölésünket felhasználva, a megfelelő feltételes valószínűségi állítás azt mondja ki, hogy minden  $t$ -re

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-1}) \quad (15.1)$$

Így egy elsőrendű Markov-folyamatban az állapotok időbeli változását leíró szabályokat teljes mértékben tartalmazza a  $P(X_t|X_{t-1})$  feltételes eloszlás, amit az elsőrendű folyamat **állapotátmennet-modelljének** (**transition model**) nevezünk.<sup>2</sup> A 15.1. ábra elsőrendű és másodrendű Markov-folyamatokhoz tartozó Bayes-hálóstruktúrát mutat.

<sup>2</sup> Az állapotátmennet-modell a valószínűségi analógja a 7. fejezet logikai frissítő áramkörének és a 10. fejezet követő állapot axiómáinak.



**15.1. ábra.** (a) Egy elsőrendű Markov-folyamathoz tartozó Bayes-hálóstruktúra. A reprezentált Markov-folyamatban az állapotot az  $X_t$  változók definiálják. (b) Egy másodrendű Markov-folyamat.

Az  $X_t$  állapotváltozók szüleinek a korlátozásán túl korlátoznunk kell az  $E_t$  bizonyítékváltozók szüleit is. Tipikusan feltesszük, hogy a bizonyítékváltozók egy  $t$  időpillanatban csak az aktuális állapottól függnek:

$$P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t) \quad (15.2)$$

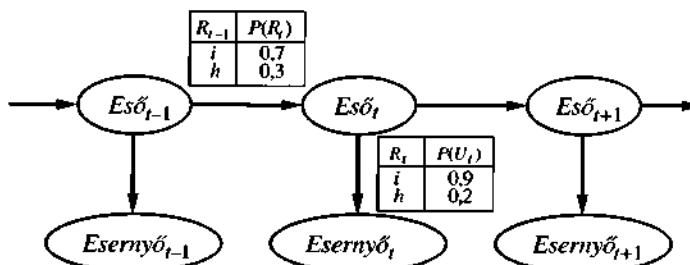
A  $P(E_t | X_t)$  feltételes eloszlást **érzékelő modellnek (sensor model)** nevezik (vagy néha **megfigyelési modellnek [observational model]**), mivel leírja, hogy az „érzékelőket” – azaz a bizonyítékváltozókat – hogyan befolyásolja a világ aktuális állapota. Vegyük észre a függés irányát: a „nyíl” az állapottól az érzékelő értékére mutat, mivel a világ állapota okozza azt, hogy az érzékelők bizonyos értékeket vegyenek fel. Az esernyős világban például az eső okozza az esernyő feltűnését. (A következetési folyamat természetesen a másik irányban halad; a modellezett függések iránya és a következetés iránya közötti különbségtétel a Bayes-hálók egyik fő előnye.)

Az állapotátmenet-modellhez és az érzékelő modellhez még meg kell adnunk egy  $P(X_0)$  a priori eloszlást a 0. időpontbeli állapotok felett. Ez a három eloszlás, kombinálva a (15.1) és a (15.2) egyenletekben megfogalmazott feltételes függetlenségi állításokkal, biztosítja számunkra a teljes együttes eloszlás meghatározását az összes változó felett. Bármely véges  $t$ -re azt kapjuk, hogy

$$P(X_0, X_1, \dots, X_t, E_1, \dots, E_t) = P(X_0) \prod_{i=1}^t P(X_i | X_{i-1}) P(E_i | X_i)$$

A függetlenségi állítások egy nagyon egyszerű Bayes-hálóstruktúrának felelnek meg, ami az egész rendszert leírja. A 15.2. ábrán látható a háló struktúrája az esernyős példa esetén, beleértve az állapotátmenet- és az érzékelő modellekhez tartozó feltételes eloszlásokat.

Az ábrán lévő struktúra elsőrendű Markov-folyamat feltételezésen alapul, mivel feltesszük, hogy az eső valószínűsége csak attól függ, hogy az előző nap esett-e. Egy ilyen feltevésnek a helyévalósága magától a tárgyterülettől függ. Az elsőrendű Markov-feltétel kimondja, hogy az állapotváltozók az összes olyan információt tartalmazzák, amely a következő időpontbeli valószínűség-eloszlások megadásához szükséges. Néha ez a feltevés pontosan teljesül – például ha egy részecske **véletlen bolyongást (random walk)** végez az  $x$  tengelyen,  $\pm 1$ -gyel változtatva meg a pozíóját minden időpontban, ekkor az  $x$  koordinátát használva állapotként, egy Markov-lánc adódik. Gyakran a feltevés csak közelítő, mint amikor az eső jóslása csak az alapján történik, hogy esett-e az előző napon. Két lehetséges javítás létezik, ha a közelítés túlságosan pontatlannak bizonyul:



15.2. ábra. Az esernyős világot leíró Bayes-hálóstruktúra és feltételes eloszlások. Az álapotámenet-modell a  $P(Eső_t|Eső_{t-1})$  feltételes valószínűség-eloszlás, az érzékelő modell a  $P(Esernyő_t|Eső_t)$ .

1. A Markov-folyamat rendjének a megnövelése. Például létrehozhatnánk egy másodrendű modellt, felvéve egy  $Eső_{t-2}$ -t mint az  $Eső_t$  szülőjét, ami lehet, hogy valamivel pontosabb predikciót adna (például Palo Altóban nagyon ritkán esik több mint két napig egyfolytában).
2. Az állapotváltozók halmazának megnövelése. Például felvehetnénk az  $\text{Évszak}_t$ , változót, hogy ez lehetővé tegye számunkra az esős évszakok történeti feljegyzéseinak beépítését, vagy hozzáadhatnánk a  $Hőmérséklet_t$ , a  $\text{Páratartalom}_t$  és a  $\text{Légnymás}_t$  változókat, hogy az eső feltételeinek fizikai modelljeit felhasználhassuk.

A 15.1. feladat annak megmutatását kéri, hogy az első megoldás – a folyamat rendjének a megnövelése – minden átfogalmazható az állapotváltozók halmazának megnövelésére, változatlanul hagyva a rendet. Vegyük észre, hogy állapotváltozók hozzáadása javíthatja a rendszer előrejelző erejét, de megnöveli a predikciós követelményeket is: ekkor már az új változókat is jósolni kell. Így a változóknak egy „önmagában elégséges” halmazát keressük, ami valójában azt jelenti, hogy meg kell értenünk a modellezett folyamat „fizikáját”. A folyamat pontos modellezése iránti követelmény nyilvánvalóan mérsékeltebb, ha új érzékelőket vehetünk fel (például a hőmérséklet és a nyomás mérése), amelyek közvetlenül az új állapotváltozókról szolgáltatnak információt.

Gondoljuk meg például egy X–Y síkon véletlenszerűen sétáló robot követésének a problémáját. Egy javaslat az állapotváltozók egy elégséges halmazára ekkor a pozíció és a sebesség lehet: Newton törvényeit felhasználva kiszámolható az új pozíció, és a sebesség megjósolhatatlanul változik. Ha azonban a robot akkumulátorról üzemel, akkor az elem lemerülésének tipikusan van egy szisztematikus hatása a sebesség megváltozárára. Mivel ez viszont függ attól, hogy mennyi energia használódott el az összes korábbi manőverben, a Markov-tulajdonság sértől. A Markov-tulajdonságot helyreállíthatjuk egy Akkumulátor, feltöltöttségi szintnek mint az X-t alkotó állapotváltozók egyikének a felvételével. Ez segít a robot mozgásának a jóslásában, de ugyanakkor megköveteli az Akkumulátor, jóslását az Akkumulátor<sub>t-1</sub>-ból és a sebességből. Bizonyos esetekben ez megbízhatóan megtethető: a pontosság azonban javulna egy, az akkumulátor feltöltöttsegét mérő új érzékelő felvételével.

## 15.2. KÖVETKEZTETÉS IDŐBELI MODELLEKBEN

Egy általános időbeli modellstruktúra kidolgozása után, most már megfogalmazhatjuk a megoldandó alapvető következetési feladatokat:

- **Szűrés (filtering)** vagy **ellenőrző megfigyelés (monitoring)**. Ez a **bizonyossági állapot (belief state)** kiszámításának a feladata – ami a jelenlegi állapot feletti a posteriori eloszlás, az adott időpontig vett összes bizonyíték ismeretében. Azaz szeretnénk kiszámítani a  $P(X_t | e_{1:t})$  mennyiséget, feltéve, hogy a bizonyítékok folyamatos sorozatban érkeznek kezdve a  $t = 1$  időponttól. Az esernyős példában ez az aznapi eső valószínűségének a kiszámítását jelentené, az esernyőhordozó eddigi összes megfigyelésének az ismeretében. A szűrés az, amit egy racionális ágensnek el kell végeznie ahhoz, hogy a jelenlegi állapotot követni tudja, és így racionális döntéseket hozhasson (lásd 17. fejezet). Kiderül, hogy majdnem azonos számítás szolgáltatja a bizonyítéksorozat megfigyelésének a **valószínűségét (likelihood)**,  $P(e_{1:t} | \cdot)$ .
- **Előrejelzés (prediction)**. Ez egy **jövőbeli állapot** feletti a posteriori eloszlás kiszámításának a feladata, az adott időpontig vett összes bizonyíték ismeretében. Azaz, szeretnénk kiszámítani a  $P(X_{t+k} | e_{1:t})$  mennyiséget valamely  $k > 0$  esetén. Az esernyős példában ez jelentheti az eső valószínűségének a kiszámítását három napra előre, az esernyőhordozó eddigi összes megfigyelésének az ismeretében. Az előrejelzés hasznos a cselekedetek lehetséges sorozatainak a kiértékelésében.
- **Simítás (smoothing)** vagy **visszatekintés (hindsight)**. Ez egy múltbeli állapot feletti a posteriori eloszlás kiszámításának a feladata, a jelen időpontig vett összes bizonyíték ismeretében. Azaz szeretnénk kiszámítani a  $P(X_k | e_{1:t})$  mennyiséget valamely  $0 \leq k < t$  esetén. Az esernyős példában ez jelentheti az eső valószínűségének a kiszámítását múlt szerdára, ha ismerjük az esernyőhordozónak a mai napig történt összes megfigyelését. A visszatekintés az állapotnak egy jobb becslését adja, mint ami akkor elérhető volt, mivel több bizonyítékot használ fel.
- **Legvalószínűbb magyarázat (most likely explanation)**. A megfigyelések egy sorozatának ismeretében lehet, hogy szeretnénk megtalálni azt az állapot-sorozatot, ami a leginkább valószínű, hogy az adott megfigyeléseket generálta. Azaz szeretnénk kiszámítani az  $\operatorname{argmax}_{x_{1:t}} P(x_{1:t} | e_{1:t})$  értékét. Például ha az esernyő feltűnik az első három nap mindenkorán, és hiányzik a negyediken, akkor a legvalószínűbb magyarázat az, hogy az első három napon esett és a negyediken nem esett. Az erre a feladatra szolgáló algoritmusok számos alkalmazásban hasznosak, ideértve a beszédfelismerést – ahol a cél a szavak legvalószínűbb sorozatának a megtalálása hangok sorozatának ismeretében – és egy zajos csatornán továbbított bináris szekvenciák rekonstrukcióját.

Ezeken a feladatokon túl szükség van még módszerekre az állapotámenet- és érzékelő modellek megfigyelésekkel történő *meganulására*. Csakúgy, mint a statikus Bayes-hálóknál, a dinamikus Bayes-hálós tanulás elvégezhető mint a következetés mellékterméke. A következetés egy becslést ad arra, hogy milyen átmenetek következtek be valójában, és milyen állapotok generálták az érzékelők értékeit, és ezek a becslések felhasználhatók a modell frissítésére. A frissített modellek jobb becsléseteket adnak, és a folyamat a konvergálásig iterálódik. A teljes folyamat a várható érték-maximálás vagy **EM algoritmus** egy esete (lásd 20.3. alfejezet). Figyelemre méltó részlet, hogy a tanulás

teljes simítós következtetést igényel szűrés helyett, mivel ez jobb becslésekkel ad a folyamat állapotára. Lehet, hogy a szűressel való tanulás nem konvergál helyesen; gondoljunk például a gyilkosságok felderítésének a megtanulására: a visszatekintés *mindig* szükséges annak kikövetkeztetéséhez, hogy megfigyelhető bizonyítékok alapján, hogy mi történt a gyilkosság helyszínén.

Az előző szakaszban felsorolt négy következtetési feladatot megoldó algoritmusok először leírhatók általános szinten, függetlenül az alkalmazott modell konkrét típusától. Az egyes modellekhez igazodó javításokat a következő fejezetekben ismertetjük.

## Szűrés és előrejelzés

Kezdjük a szűréssel. Megmutatjuk, hogy ez menet közben egyszerűen, online módon elvégezhető: a  $t$ -edik időpillanatig tartó szűrés eredményét ismerve, a  $t+1$ -edik időpillanatra az eredmény könnyen kiszámítható az új  $e_{t+1}$  bizonyítékból. Azaz

$$P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))$$

valamely  $f$  függvényel. Ezt a folyamatot gyakran nevezik **rekurzív becslésnek (recursive estimation)**. Tekinthetjük a számítást úgy, mint ami valójában két részből áll: elsőként a jelenlegi állapot eloszlását terjesztiük előre  $t$ -ről  $t+1$ -re; azután fogjuk frissíteni, felhasználva az új  $e_{t+1}$  bizonyítékot. Ez a kétlépéses folyamat elég egyszerűen alakul:

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= P(X_{t+1}|e_{1:t}, e_{t+1}) && \text{(felosztva a bizonyítékot)} \\ &= \alpha P(e_{t+1}|X_{t+1}, e_{1:t}) P(X_{t+1}|e_{1:t}) && \text{(a Bayes-szabályt használva)} \\ &= \alpha P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t}) && \text{(a bizonyíték Markov-tulajdonsága} \\ &&& \text{miatt)} \end{aligned}$$

Itt és végig a fejezetben, az  $\alpha$  egy normalizációs konstans, ami a valószínűségek 1-re összegzését biztosítja. A második tényező  $P(X_{t+1}|e_{1:t})$  reprezentálja a következő állapot egylépéses előrejelzését, és az első tényező ezt frissíti az új bizonyítékkal. Végyük észre, hogy  $P(e_{t+1}|X_{t+1})$  közvetlenül kinyerhető az érzékelő modellből. Most a jelenlegi  $X_t$  állapotot feltételnek véve a következő állapot egylépéses előrejelzéséhez juttunk:

$$\begin{aligned} P(X_{t+1}|e_{1:t+1}) &= \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1:t}) \\ &= \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t}) \quad \text{(kihasználva a Markov-tulajdonságot)} \end{aligned} \quad (15.3)$$

Az összegzésen belül az első tényező egyszerűen az állapotátmenet-modell és a második a jelenlegi állapot eloszlása. Így megvan a kívánt rekurzív képlet. A szűrt  $P(X_t|e_{1:t})$  becslésre úgy gondolhatunk, mint egy  $f_{1:t}$  „üzenetre”, amit előre terjesztünk végig a sorozaton, minden átmenetnél az új megfigyeléssel módosítva és frissítve. A folyamat

$$f_{1:t+1} = \alpha \text{ ELŐRE}(f_{1:t}, e_{t+1})$$

ahol az **ELŐRE** a (15.3) egyenlet által leírt frissítést hajtja végre.

Ha az összes állapotváltozó diszkrét, minden frissítés ideje állandó (azaz  $t$ -től független), és a tárigény is állandó. (Ezek az állandók természetesen függnek az állapottér méretétől)



és a tárgyalt időbeli modell konkrét típusától.) A frissítés idő- és tárigényének állandónak kell lennie, ha egy korlátos memóriájú ágensnek követnie kell az aktuális állapot eloszlását a megfigyelések egy korlátlan sorozata esetén.

Mutassuk be a szűrési folyamatot két lépésen keresztül az alap esernyő példában (lásd 15.2. ábra). Feltesszük, hogy a biztonsági őrünknek van valamilyen a priori hite, hogy a 0. napon esett-e, éppen azelőtt, hogy a megfigyelések sorozata elkezdődik. Tegyük fel, hogy ez  $P(R_0) = \langle 0,5, 0,5 \rangle$ . Most a két megfigyelést a következőképpen dolgozzuk fel:

- Az 1. napon az esernyő feltűnik, így  $U_1 = \text{igaz}$ . Az előrejelzés  $t = 0$ -ról  $t = 1$ -re
 
$$\begin{aligned} P(R_1) &= \sum_{r_0} P(R_1|r_0) P(r_0) \\ &= \langle 0,7, 0,3 \rangle \times 0,5 + \langle 0,3, 0,7 \rangle \times 0,5 = \langle 0,5, 0,5 \rangle \end{aligned}$$
 és a  $t = 1$ -beli bizonyítékkal való frissítés azt adja, hogy
- $P(R_1|u_1) = \alpha P(u_1|R_1) P(R_1) = \alpha \langle 0,9, 0,2 \rangle \langle 0,5, 0,5 \rangle$   
 $= \alpha \langle 0,45, 0,1 \rangle \approx \langle 0,818, 0,182 \rangle$
- A 2. napon az esernyő feltűnik, így  $U_2 = \text{igaz}$ . Az előrejelzés  $t = 1$ -ről  $t = 2$ -re
 
$$\begin{aligned} P(R_2|u_1) &= \sum_{r_1} P(R_2|r_1) P(r_1|u_1) \\ &= \langle 0,7, 0,3 \rangle \times 0,818 + \langle 0,3, 0,7 \rangle \times 0,182 \approx \langle 0,627, 0,373 \rangle \end{aligned}$$
 a  $t = 2$ -beli bizonyítékkal való frissítés pedig azt adja, hogy
 
$$\begin{aligned} P(R_2|u_1, u_2) &= \alpha P(u_2|R_2) P(R_2|u_1) = \alpha \langle 0,9, 0,2 \rangle \langle 0,627, 0,373 \rangle \\ &= \alpha \langle 0,565, 0,075 \rangle \approx \langle 0,883, 0,117 \rangle \end{aligned}$$

Érzésünk szerint az eső valószínűsége azért növekedett az 1. napról a 2. napra, mert az eső folytatódik. A 15.2. (a) feladat ennek a tendenciának a további vizsgálatát kéri.

Az előrejelzés feladatát tekinthetjük egyszerűen szűrésnek új bizonyíték hozzáadása nélkül. Valójában a szűrési folyamat már magában foglal egy egylépéses előrejelzést és könnyen származtatható a következő rekurzív számítás a  $t+k+1$  állapot előrejelzésére a  $t+k$  előrejelzéséből:

$$P(X_{t+k+1}|e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1}|x_{t+k}) P(x_{t+k}|e_{1:t}) \quad (15.4)$$

Természetesen ez a számítás csak az állapotátmenet-modellt tartalmazza és nem az érzelő modellt.

Érdekes meggondolni, hogy mi történik, ahogy egyre távolabba próbálunk előre jelezni a jövőben. A 15.2. (b) feladat megmutatja, hogy az eső előrejelzett eloszlása a  $\langle 0,5, 0,5 \rangle$  fix ponthoz konvergál, ami után a továbbiakban állandó marad. Ez az állapotátmenet-modell által meghatározott Markov-folyamat stacionárius eloszlása (stationary distribution) (lásd még 605. oldal). Renge teg ismeret gyűlt fel az ilyen eloszlások tulajdonságairól és a keverési időről (mixing time) – ami nagyjából a fix pont eléréséig eltelt idő. Gyakorlati szempontból ez bukásra ítélt minden olyan kísérletet, ami a keverési idő kis hányadánál nagyobb számú lépés múlva kísérli meg az aktuális állapot előrejelzését. Minél bizonytalannabb az állapotátmenet-modell, annál rövidebb a keverési idő, és a jövő annál homályosabb.

A szűrésen és az előrejelzésen túl, felhasználhatjuk az előrehaladó rekurziót egy bizonyítéksorozat  $P(e_{1:t})$  valószínűségének a kiszámítására is. Ez egy hasznos mennyiség, ha különböző időbeli modelleket szeretnénk összehasonlítani, amelyek ugyanazt a bizonyítéksorozatot állíthatták elő; például a 15.6. alfejezetben különböző szavakat

hasonlítunk össze, amelyek ugyanazt a hangsorát állíthatták elő. Ehhez a rekurzióhoz felhasználjuk az  $\ell_{1:t} = P(\mathbf{X}_t, \mathbf{e}_{1:t})$  valószínűségi (likelihood) üzenetet. Egyszerű feladat azt megmutatni, hogy

$$\ell_{1:t+1} = \text{ELŐRE}(\ell_{1:t}, \mathbf{e}_{t+1})$$

Kiszámítva  $\ell_{1:t+1}$ , az aktuális valószínűséget az  $\mathbf{X}_t$  feletti összegzéssel kapjuk:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \ell_{1:t}(\mathbf{x}_t) \quad (15.5)$$

## Simítás

Ahogy korábban meghatároztuk, a simítás (smoothing) múltbeli állapotok feletti eloszlás kiszámításának a folyamata a jelenig tartó bizonyítékok ismeretében; azaz  $P(\mathbf{X}_k | \mathbf{e}_{1:t})$  számítása valamely  $0 \leq k < t$  esetén (lásd 15.3. ábra). Ez legkényelmesebben két részletben végezhető el – a bizonyíték  $k$ -ig és a bizonyíték  $k + 1$ -től  $t$ -ig,

$$\begin{aligned} P(\mathbf{X}_k | \mathbf{e}_{1:t}) &= P(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \quad (\text{a Bayes-szabályt használva}) \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) \quad (\text{a feltételes függetlenséget használva}) \\ &= \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t} \end{aligned} \quad (15.6)$$

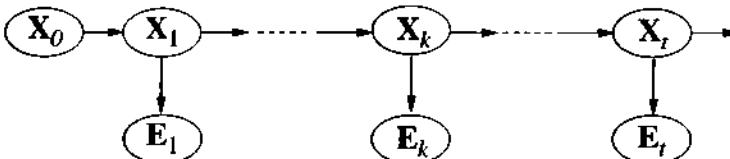
ahol definiáltuk a  $\mathbf{b}_{k+1:t} = P(\mathbf{e}_{k+1:t} | \mathbf{X}_k)$  „visszafelé” üzenetet, analóg módon az  $\mathbf{f}_{1:k}$  előre üzenetet. Az  $\mathbf{f}_{1:k}$  előre üzenetet előre szűréssel lehet kiszámolni 1-től  $k$ -ig, a (15.3) egyenlet alapján. A  $\mathbf{b}_{k+1:t}$  visszafelé üzenetről pedig kiderül, hogy egy egyszerű rekurzív folyamattal kiszámítható, ami  $t$ -től visszafelé halad:

$$\begin{aligned} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad (\text{feltételek véve } \mathbf{X}_{k+1}-et) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad (\text{a feltételes függetlenség miatt}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \end{aligned} \quad (15.7)$$

ahol az utolsó lépés az  $\mathbf{e}_{k+1}$  és az  $\mathbf{e}_{k+2:t}$  feltételes függetlenségből következik az  $\mathbf{x}_{k+1}$  melletti feltétellel. Az összegzésben a három tényezőből az első és a harmadik közvetlenül a modellből megkapható, a második pedig a „rekurzív hívás”. Az üzenetjelölést használva kapjuk, hogy

$$\mathbf{b}_{k+1:t} = \text{HÁTRA}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$$

ahol a HÁTRA a (15.7) egyenlet által leírt frissítést hajtja végre. Ahogy az előrefelé rekurzióval, az egyes frissítésekhez szükséges idő- és tárigény állandó, és így  $t$ -től független itt is.



**15.3. ábra.** A simítás egy múltbeli  $k$  időpontban az állapot a posteriori  $P(X_k|e_{1:t})$  eloszlását számítja ki egy teljes, 1-től  $t$ -ig terjedő megfigyelés sorozat ismeretében

Most láthatjuk, hogy a (15.6) egyenlet minden két tényezője kiszámítható időbeli rekurzióval, az egyik előrefelé fut 1-től  $k$ -ig, és a szűrés (15.3) egyenletét használja, a másik visszafelé fut  $t$ -től  $k+1$ -ig, és a (15.7) egyenletet használja. Figyeljünk arra, hogy a visszafelé fázist  $b_{t+1:t} = P(e_{t+1:t}|X_t) = 1$ -gyel inicializáljuk, ahol 1 egy egyesekből álló vektor. (Mivel  $e_{t+1:t}$  egy üres sorozat, a megfigyelésének valószínűsége 1.)

Alkalmazzuk most ezt az algoritmust az esernyős példára, kiszámítva az eső valószínűségének simított becslését  $t = 1$ -nél, az első és második nap megfigyeléseinek ismeretében. A (15.6) egyenlet szerint ezt az adja meg, hogy

$$P(R_1|u_1, u_2) = \alpha P(R_1|u_1) P(u_2|R_1) \quad (15.8)$$

A korábban leírt előrefelé szűrési folyamatból már tudjuk, hogy az első tényező  $\langle 0,818, 0,182 \rangle$ . A második tényezőt a (15.7) egyenletbeli visszafelé rekurziót alkalmazva számíthatjuk ki:

$$\begin{aligned} P(u_2|R_1) &= \sum_{r_2} P(u_2|r_2) P(r_2|R_1) \\ &= (0,9 \times 1 \times \langle 0,7, 0,3 \rangle) + (0,2 \times 1 \times \langle 0,3, 0,7 \rangle) = \langle 0,69, 0,41 \rangle \end{aligned}$$

Ezt visszahegyettesítve a (15.8) egyenletbe azt kapjuk, hogy a simított becslés az esőre az 1. napon

$$P(R_1|u_1, u_2) = \alpha \langle 0,818, 0,182 \rangle \times \langle 0,69, 0,41 \rangle \approx \langle 0,883, 0,117 \rangle$$

Így ebben az esetben a simított becslés nagyobb, mint a szűrt becslés (0,818). Ennek oka az, hogy az esernyő a 2. napon valószínűbbé teszi, hogy esett a 2. napon, és mivel az eső tartósan szokott esni, ez még valószínűbbé teszi, hogy esett az 1. napon.

Mind az előrefelé, mind a visszafelé haladó rekurzió lépéseként állandó időt igényel; így a simítás időkomplexitása  $e_{1:t}$  bizonyíték esetén  $O(t)$ . Ez egy konkrét  $k$  időlépésbeli simítás komplexitása. Ha a teljes szekvenciát szeretnénk simítani, nyilvánvaló módszer, hogy egyszerűen az egész simítási folyamatot lefuttatjuk minden egyes simítandó időpillanatra. Ez  $O(t^2)$  időkomplexitást eredményez. Jobb megközelítés a dinamikai programozás nagyon egyszerű alkalmazásával  $O(t)$ -re csökkenti a komplexitást. A megoldás kulcsa megjelent az esernyős példa előző elemzésében, ahol az előrefelé szűrés folyamatának eredményeit újra fel tudtuk használni. A lineáris idejű algoritmus kulcsa így az előrefelé szűrés eredményeinek a tárolása az egész sorozatnál. Aztán a visszafelé rekurziót futtatjuk  $t$ -től 1-ig, kiszámítva a szűrt becslést minden  $k$  lépésnél a kiszámolt  $b_{k+1:t}$  visszafelé üzenetből és a tárolt  $f_{1:k}$  előrefelé üzenetből. Az algoritmus, amit találón előre-hátra algoritmusnak (**forward-backward algorithm**) neveznek, a 15.4. ábrán látható.

```

function ELŐRE-HÁTRA(ev, prior) returns valószínűségeszlások egy vektora
inputs: ev, a bizonysíték (megfigyelés) értékek egy vektora az 1, ..., t lépéshoz
        prior: a kezdő állapot  $P(X_0)$  a priori eloszlása
local variables: fv, az előrefelé üzenetek vektora a 0, ..., t lépéshoz
                    b, a hátrafelé üzenet egy reprezentációja, kezdetben csupa 1
                    sv, a simított becslések vektora az 1, ..., t lépéshoz

fv[0] ← prior
for i = 1 to t do
    fv[i] ← ELŐRE(fv[i - 1], ev[i])
for i = t downto 1 do
    sv[i] ← NORMALIZÁL(fv[i] × b)
    b ← HÁTRA(b, ev[i])
return sv

```

**15.4. ábra.** Az előre-hátra algoritmus, ami állapotok egy sorozatának az a posteriori valószínűségeit számítja ki adott megfigyelési sorozat ismeretében. Az ELŐRE és a HÁTRA műveleteket rendre a (15.3) és (15.7) egyenletek definiálják.

A figyelmes olvasó észrevehette, hogy a 15.3. ábrán látható Bayes-hálóstruktúra egy **polifa** (**polytree**) a 14. fejezet szóhasználatában. Ez azt jelenti, hogy a csoportosító algoritmus egy nyilvánvaló alkalmazása szintén lineáris idejű algoritmust eredményez, ami a teljes sorozatra kiszámítja a simított becslést. Mára már világossá vált, hogy az előre-hátra algoritmus valójában a csoportosító eljárásokban használt polifa terjesztési algoritmusnak egy speciális esete (bár a kettőt egymástól függetlenül fejlesztették ki).

Az előre-hátra algoritmus alkotja a gerincét azon számítási módszereknek, amelyeket számos zajos megfigyelések sorozatával foglalkozó alkalmazásban használnak, a beszédfelismeréstől a repülőgépek radarkövetéséig. Ahogy leírtuk, két gyakorlati hátránya van. Az első, hogy a tárigénye túl nagy lehet azoknál az alkalmazásoknál, ahol az állapottér nagy, és a sorozatok hosszúak, mivel  $O(|f|t)$  méretű tárat használ, ahol |f| az előrefelé üzenet reprezentációjának a mérete. A tárigény  $O(|f|\log t)$ -re csökkenhető az időkomplexitásnak egy  $\log t$  tényezővel történő egyidejű megnövelése árán, ahogy a 15.3. feladat mutatja. Bizonyos esetekben (lásd 15.3. alfejezet) egy állandó tárigényű algoritmus használható időbüntetés nélkül.

Az alapalgoritmus második hátránya, hogy a folyamatos (online) működéshez módszert igényel. Ekkor ugyanis a korábbi időpontokhoz simított becslésekkel kell kiszámítanunk, amint folyamatosan új megfigyelések érkeznek a sorozat végéhez. A leggyakoribb követelmény az állandó időkülönbségű simítás (**fixed-lag smoothing**), ami a  $P(X_{t-d}|e_1:t)$  simított becslés kiszámítását jelenti egy rögzített  $d$ -re. Azaz a simítást a jelenlegi  $t$  időpillanat előtt  $d$  lépéssel lévő időpontra végezzük el;  $t$  növekedését a simításnak is követnie kell. Nyilvánvaló, hogy lefuttathatjuk az előre-hátra algoritmust a  $d$  lépéses „ablakban”, amint egy új bizonyíték adódik, de ez nem tűnik hatékonynak. A 15.3. alfejezetben látni fogjuk, hogy az állandó időkülönbségű simításnál egy frissítés bizonyos esetekben állandó idő alatt megtehető, függetlenül a  $d$  időkülönbségtől.

## A legvalószínűbb sorozat megtalálása

Tegyük fel, hogy a biztonsági őr esernyősorozata a munkája első öt napjában az [igaz, igaz, hamis, igaz, igaz]. Mi a legvalószínűbb időjárás-sorozat, ami ezt megmagyarázza? Az esernyő hiánya a 3. napon azt jelenti, hogy nem esett, vagy azt, hogy az igazgató elfelejtette elhozni? Ha a 3. napon nem esett, talán a 4. napon sem esik (mivel az időjárás nem változik gyorsan), de ekkor az igazgató pusztá óvatosságból hozta az esernyőt. Összességében 2<sup>5</sup> lehetséges időjárás-sorozator választhatunk. Létezik-e módszer a legvalószínűbb sorozat megkeresésére, az összes felsorolása nélkül?

Az egyik megközelítés, amit kipróbalhatunk a következő lineáris idejű algoritmus: használjuk a simító algoritmust, hogy megtaláljuk az időjárás a posteriori eloszlását minden időpillanatban; majd állítsuk elő a sorozatot minden egyes lépésnél az a posteriori szerinti legvalószínűbb időjárást felhasználva. Egy ilyen megközelítést kötelező gyanakvással kell fogadnia az olvasónak, mivel a szűrés által kiszámított a posteriori eloszlások az egyes időpontok feletti eloszlások, ezzel szemben a legvalószínűbb sorozat megtalálásához az összes időpont feletti *együttető* valószínűségeket kell figyelembe venni. Az eredmények valójában nagyon különbözök lehetnek (lásd 15.4. feladat).

Lineáris idejű algoritmus azonban *létezik* a legvalószínűbb sorozat megtalálására, de kicsit több gondolkozást igényel. Ugyanonnan a Markov-tulajdonságon alapul, mint ami hatékony algoritmusokat eredményezett a szűrésre és a simításra. A problémáról való gondolkodás legkönnyebb módja, hogy ha minden sorozatot egy *útvonalnak* tekintünk egy gráfban, aminek a csomópontjai az egyes időpillanatokban a lehetséges állapotok. Egy ilyen gráf látható az esernyős problémára a 15.5. (a) ábrán. Most gondoljuk meg a gráfon keresztül vezető legvalószínűbb út megteresésének a problémáját, ahol egy út valószínűsége (a likelihood érték) az útvonal menti átmenetek valószínűségeinek és az egyes állapotokban adott megfigyelések valószínűségeinek a szorzata. Koncentráljunk most azokra az útvonalakra, amelyek elérik az  $E\ddot{o}_5 = \text{igaz}$  állapotot. A Markov-tulajdonság miatt fennáll, hogy az  $E\ddot{o}_5 = \text{igaz}$  állapotba vezető legvalószínűbb útvonal tartalmazza azt a legvalószínűbb útvonalat, amely valamelyik 4. időpontbeli állapotba vezet, és amit egy átmenet követ az  $E\ddot{o}_5 = \text{igaz}$ -ba; a 4. időpontbeli állapot pedig, ami része lesz az  $E\ddot{o}_5 = \text{igaz}$ -ba vezető útnak, az az állapot, amelyik maximalizálja ennek az útvonalnak a valószínűségét.

Másképpen fogalmazva, van egy *rekurzív kapcsolat* az  $x_{t+1}$  állapotokba vezető legvalószínűbb útvonalak és az  $x_t$  állapotokba vezető legvalószínűbb útvonalak között. Ez a kapcsolat olyan egyenletként írható fel, amely az útvonalak valószínűségeit kapcsolja össze:

$$\max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) \max_{x_t} (P(X_{t+1} | x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t | e_{1:t})) \quad (15.9)$$

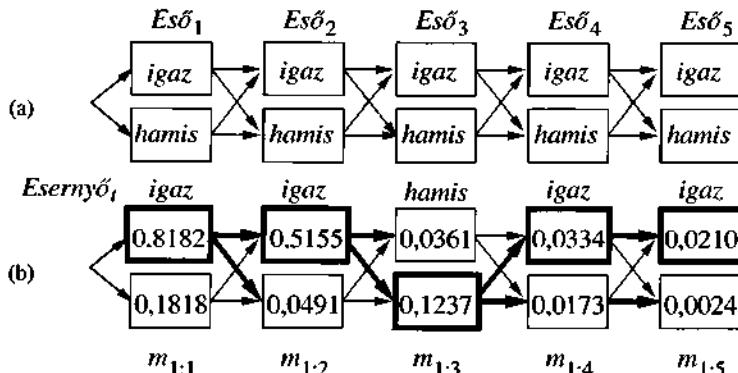
A (15.9) egyenlet megegyezik a szűrés (15.3) egyenletével, azzal a kivétellel, hogy

1. az  $f_{1:t} = P(X_t | e_{1:t})$  előre üzenet helyett a következő üzenet szerepel:

$$m_{1:t} = \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, X_t | e_{1:t})$$

ami az  $x_t$  állapotba vezető legvalószínűbb út valószínűségeit adja meg; és

2. a (15.3) egyenletbeli  $x_t$  feletti összegzés helyett a (15.9) egyenletben az  $x_t$  feletti maximalizálás szerepel.



**15.5. ábra.** (a) Az  $Eső_t$  lehetséges állapot sorozatait tekintetjük egy átvezető útnak egy gráfon, aminek csomópontjai a lehetséges állapotok az egyes időpontokban. (Az állapotokat négyzetes csomópont jelzi, hogy félreérthetetlenül megkülönböztessük őket egy Bayes-háló csomópontjaitól.) (b) A Viterbi-algoritmus működése az [*igaz*, *igaz*, *hamis*, *igaz*, *igaz*] megfigyelési sorozatra. minden  $t$  időpontra feltüntettük az  $m_{1:t}$  üzenet értékeit, ami minden egyes  $t$  időpontbeli állapothoz megadja a legjobb, benne végződő sorozat valószínűségét. minden egyes állapothoz egy vastag nyíl is vezet, ami a legjobb elődjét jelzi, a megelőző sorozat valószínűségének és az átmenet valószínűségének a szorzata szerint. A vastag nyílik visszafelé követése az  $m_{1:5}$ -beni legvalószínűbb állapotból pedig megadja a legvalószínűbb sorozatot.

Igy a legvalószínűbb állapot sorozat kiszámítása hasonló a szűréshez: előrefelé végigfut a sorozaton, minden időpontban kiszámítva az  $m$  üzeneteket a (15.9) egyenlet szerint. A számítás menetét a 15.5. (b) ábra mutatja. Végül ez kiadja az összes végső állapothoz vezető legvalószínűbb útvonal valószínűségét. Igy már könnyen kiválasztható a teljes hosszúságú legvalószínűbb sorozat (a vastagon szedettel kiemelt végső állapottól). Az aktuális sorozat azonosításához – szemben azzal, amikor csak a valószínűséget számítjuk ki – az algoritmusnak minden állapothól mutatókat kell nyilvántartania a legjobb hozzá vezető állapothoz (vastagon szedettel látható); a sorozat a mutatóknak a legjobb végső állapottól való visszafelé követésével határozható meg.

Az előzőleg leírt algoritmust **Viterbi-algoritmusnak** nevezik a megalkotója után. Hasonlóan a szűrési algoritmushoz, ennek a komplexitása is lineáris  $t$ -ben, a sorozat hosszában. Azonban eltérően a szűréstől, a tárigénye szintén lineáris  $t$ -ben. Ez azért van így, mert a Viterbi-algoritmusban mutatókkal kell nyilvántartani az egyes állapotokhoz vezető legjobb sorozatot.

## 15.3. REJTETT MARKOV-MODELEK

Az előző fejezet időbeli valószínűségi következtetésre szolgáló algoritmusokat vezetett be egy általános keretben, ami független volt az állapotátmenet- és az érzékelő modellek speciális formájától. Ebben és a következő két alfjejezetben, olyan konkrétabb modellekkel és alkalmazásokat tárgyalunk meg, amelyek bemutatják az alapalgoritmusok erejét, és bizonyos esetekben további tökéletesítéseket tesznek lehetővé.

A **rejtett Markov-modellekkel (RMM) (hidden Markov model, HMM)** kezdjük. Egy RMM egy olyan időbeli valószínűségi modell, amelyben a folyamat állapotát

*egyetlen diszkrét* valószínűségi változó írja le. [A modell karakterisztikus tulajdonsága tehát a rejtett állapot, azaz a (sztochasztikus) állapotfejlődés és annak passzív (sztochasztikus) megfigyelése. Az index időértelmezése helyett bármely szekvenciális értelmezés lehetséges, gyakori például az indexnek mint egy egydimenziós pozíciónak az értelmezése is – a *ford.*] A változó lehetséges értékei a világ lehetséges állapotai. Az előző fejezetben leírt esernyős példa ezért egy RMM, mivel csak egyetlen állapot-változója van az *Eső*. Az RMM keretei között további állapotváltozók csak úgy adhatók hozzá az időbeli modellhez, hogy az összes állapotváltozót egyetlen „megállítóba” kombináljuk, amelynek az értékei az egyes állapotváltozók értékeinek minden lehetséges kombinációja. Látni fogjuk, hogy az RMM-ek korlátosztott struktúrája lehetővé teszi az összes alapalgoritmus egy nagyon egyszerű és elegáns mátrixos átfordítását.<sup>3</sup> A 15.6. alfejezet bemutatja az RMM-ek használatát a beszédfelismerésben.

## Egyszerűsített mátrix algoritmusok

Egyetlen, diszkrét  $X_t$  állapotváltozó esetén konkrét forma adható az állapotátmenet-modell, az érzékelő modell és az előre és hátra üzenetek reprezentációira. Jelöljük az  $X_t$  állapotváltozó értékeit  $1, \dots, S$  egészekkel, ahol  $S$  a lehetséges állapotok száma. A  $P(X_t|X_{t-1})$  állapotátmenet-modell ekkor egy  $T \times S \times S$  mátrix, ahol

$$T_{ij} = P(X_t = j|X_{t-1} = i)$$

Azaz  $T_{ij}$  az átmenet valószínűsége  $i$ -ből  $j$ -be. Például az esernyős világban az állapot-átmenet-mátrix

$$T = P(X_t|X_{t-1}) = \begin{pmatrix} 0,7 & 0,3 \\ 0,3 & 0,7 \end{pmatrix}$$

Az érzékelő modellt szintén mátrixalakra hozzuk. Ebben az esetben mivel az  $E_t$  bizonyítékváltozó értéke ismert, mondjuk  $e_t$ , így a modellnek csak azt a részét használjuk, ami az  $e_t$  megjelenésének valószínűségét meghatározza. minden  $t$  időpontra konstruálunk egy  $O_t$  diagonális mátrixot, aminek az átlóbeli elemeit a  $P(e_t|X_t = i)$  értékek adják, a többi értéke pedig 0. Például az esernyős világban az 1. napon az  $O_1 = \text{igaz}$ , így a 15.2. ábra szerint azt kapjuk, hogy

$$O_1 = \begin{pmatrix} 0,9 & 0 \\ 0 & 0,2 \end{pmatrix}$$

Most oszlopvektort használva az előre és hátra üzenetek reprezentálására, a számítások egyszerű mátrix-vektor műveletekké válnak. A (15.3) előrefelé egyenlet ekkor

$$f_{1:t+1} = \alpha O_{t+1} T^T f_{1:t} \quad (15.10)$$

és a (15.7) visszafelé egyenlet pedig

$$b_{k+1:t} = T O_{k+1} b_{k+2:t} \quad (15.11)$$

<sup>3</sup> A vektor- és mátrixművelekben járatlan olvasó számára hasznos lehet az A) függeléket átnézni, mielőtt a fejezetben továbbhaladna.

Ezekből az egyenletekből látható, hogy az előre-hátra algoritmus (lásd 15.4. ábra) időkomplexitása egy  $t$  hosszúságú sorozatra történő alkalmazásnál  $O(S^2t)$ , mivel minden lépés egy  $S$  elemű vektor szorzását igényli egy  $S \times S$  mátrixszal. A tárigény pedig  $O(St)$ , mivel az előrefelé fázis  $t$  darab  $S$  méretű vektort tárol el.

Amellett hogy a mátrixos jelölés az RMM-ek esetében a szürés és a simítás algoritmusaira egy elegáns leírásmódot kínál, ez javított algoritmusokra is jelez lehetőségeket. Az első az előre-hátra algoritmus egy egyszerű változata, ami simítás elvégzését teszi lehetővé állandó tárigény mellett, a sorozat hosszától függetlenül. Az ötlet az, hogy egy konkrét  $k$  időpillanatban a simítás mind az  $\mathbf{f}_{1:k}$  előre, mind a  $\mathbf{b}_{k+1:t}$  hátra üzenetek egyidejű jelenlétéét igényli a (15.6) egyenlet szerint. Az előre-hátra algoritmus ezt úgy biztosítja, hogy az előrefelé fázisban tárolja az  $\mathbf{f}$ -eket, hogy a hátrafelé fázisban elérhetők legyenek. Egy másik módszer szerint ez úgy érhető el, hogy egyetlen menetben minden  $\mathbf{f}$ -et, minden  $\mathbf{b}$ -t ugyanabban az irányban terjesztjük. Például az  $\mathbf{f}$  „előre” üzenet terjeszthető hátrafelé, ha átrendezzük a (15.10) egyenletet, hogy a másik irányban működjön:

$$\mathbf{f}_{1:t} = \alpha'(\mathbf{T}^\top)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{t+1:t}$$

A módosított szürési algoritmus úgy működik, hogy először lefut egy szabványos előre fázis, kiszámítva  $\mathbf{f}_{1:t-d}$  (elfelejtve az összes közbenső eredményt), majd lefut egy visszafelé fázis együttes  $\mathbf{b}$ -re és  $\mathbf{f}$ -re, kiszámítva a simított becslést minden időpontban. Mivel mindegyik üzenetnek csak egy példánya szükséges, a tárigény állandó (azaz független  $t$ -től, a sorozat hosszától). Az algoritmusra egyetlen jelentős megkötés vonatkozik: az állapotátmenet-mátrixnak invertálhatónak kell lennie, és az érzékelő modellben nem lehetnek nullák – azaz minden megfigyelésnek lehetségesnek kell lenni minden állapotban.

A második terület, ahol a mátrixjelölés javítást jelez, a menet közben történő (online) simítás állandó időkülönbözettel. Az a tény, hogy a simítás elvégezhető állandó tárigénnyel azt jelzi, hogy léteznie kell egy hatékony rekurzív algoritmusnak menet közbeni simításra – azaz olyan algoritmusnak amelynek az időkomplexitása független az időkülönbözet hosszától. Tegyük fel, hogy az időkülönbözet  $d$ ; azaz a simítást  $t - d$  időpontban végezzük, ahol  $t$  a jelenlegi időpont. A (15.6) egyenlet szerint ki kell számolnunk a  $t - d$  időpontra az

$$\alpha \mathbf{f}_{1:t-d} \mathbf{b}_{t-d+1:t}$$

értékét. Majd amikor az új megfigyelés beérkezik, a  $t - d + 1$  időpontra kell kiszámítanunk az

$$\alpha \mathbf{f}_{1:t-d+1} \mathbf{b}_{t-d+2:t+1}$$

értéket. Hogyan tehető ez meg inkrementálisan? Először is, az  $\mathbf{f}_{1:t-d+1}$  kiszámítható az  $\mathbf{f}_{1:t-d}$  ből a szabványos szürés műveletét használva a (15.3) egyenlet szerint.

A visszafelé üzenet inkrementális kiszámítása trükkösebb, mivel nincs egyszerű kapcsolat a régi  $\mathbf{b}_{t-d+1:t}$  visszafelé üzenet és az új  $\mathbf{b}_{t-d+2:t+1}$  visszafelé üzenet között. Ehez, a régi  $\mathbf{b}_{t-d+1:t}$  visszafelé üzenet és a sorozat kezdő  $\mathbf{b}_{t+1:t}$  visszafelé üzenete közötti kapcsolatot fogjuk megvizsgálni. Ennek eléréséhez alkalmazzuk a (15.11) egyenletet  $d$  alkalommal, és azt kapjuk, hogy

$$\mathbf{b}_{t-d+1:t} = \left( \prod_{i=t-d+1}^t \mathbf{T} \mathbf{O}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t} \mathbf{1} \quad (15.12)$$

ahol a  $\mathbf{B}_{t-d+1:t}$  mátrix a  $\mathbf{T}$  és az  $\mathbf{O}$  mátrixok sorozatának a szorzata. A  $\mathbf{B}$ -t felfoghatjuk egy „transzformációs operátornak”, ami egy későbbi visszafelé üzenetet egy korábbiba alakít. Egy hasonló egyenlet áll fenn az új visszafelé üzenetekre a következő megfigyelés beérkezése után:

$$\mathbf{b}_{t-d+2:t+1} = \left( \prod_{i=t-d+2}^{t+1} \mathbf{T} \mathbf{O}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1} \mathbf{1} \quad (15.13)$$

Ha a (15.12) és a (15.13) egyenletben megvizsgáljuk a szorzatokat, láthatjuk, hogy egy-szerű kapcsolat van közöttük: a második szorzathoz el kell „osztani” az első szorzatot  $\mathbf{T} \mathbf{O}_{t-d+1}$ -gyel, és megszorozni az új utolsó elemmel  $\mathbf{T} \mathbf{O}_{t+1}$ -gyel. Mátrixjelöléssel ekkor a régi és az új  $\mathbf{B}$  mátrixok között egy egyszerű kapcsolat van:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1:t}^{-1} (\mathbf{T})^{-1} \mathbf{B}_{t-d+1:t} \mathbf{T} \mathbf{O}_{t+1} \quad (15.14)$$

Ez az egyenlet a  $\mathbf{B}$  mátrix egy inkrementális frissítését adja, amely viszont (a (15.3) egyenlet által) lehetővé teszi az új  $\mathbf{b}_{t-d+2:t+1}$  visszafelé üzenet kiszámítását. A teljes algoritmus, ami az  $\mathbf{f}$  és  $\mathbf{B}$  tárolását és frissítését igényli, a 15.6. ábrán látható.

```

function ÁLLANDÓ-IDŐKÜLÖNBSÉGŰ-SIMÍTÁS( $e_t$ ,  $rmm$ ,  $d$ ) returns egy eloszlás  $X_{t-d}$  felett
  inputs:  $e_t$ , a jelenlegi bizonyíték a  $t$  időpontban
            $rmm$ , egy rejtett Markov-modell  $S \times S$  méretű  $\mathbf{T}$  állapotátmenet-mátrixszal
            $d$ , az időkülönbség nagysága a simításnál
  static:  $t$ , a jelenlegi idő, kezdetben 1
             $\mathbf{f}$ , egy valószínűség-eloszlás, a  $P(X_t | e_{1:t})$  előrefelé üzenet, kezdetben PRIOR[rmm]
             $\mathbf{B}$ , a  $d$  lépésnyi hátrafelé transzformált mátrix, kezdetben az egységmátrix
             $e_{t-d:t}$ , a bizonyítékok kétirányú listája  $t - d$ -től  $t$ -ig, kezdetben üres
  local variables:  $\mathbf{O}_{t-d}$ ,  $\mathbf{O}_t$ , diagonális mátrixok az érzékelő modell információjával

  adjuk  $e_{t-d}$  az  $e_{t-d:t}$  végehez
   $\mathbf{O}_t \leftarrow$  egy  $P(e_t | X_t)$ -t tartalmazó diagonális mátrix
  if  $t > d$  then
     $\mathbf{f} \leftarrow$  ELŐRE( $\mathbf{f}$ ,  $e_t$ )
    töröld  $e_{t-d-1}$ -et az  $e_{t-d:t}$  elejéről
     $\mathbf{O}_{t-d} \leftarrow$  egy  $P(e_{t-d} | X_{t-d})$ -t tartalmazó diagonális mátrix
     $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1} \mathbf{T}^{-1} \mathbf{B} \mathbf{O}_t$ 
  else  $\mathbf{B} \leftarrow \mathbf{B} \mathbf{T} \mathbf{O}_t$ 
   $t \leftarrow t + 1$ 
  if  $t > d$  then return NORMALIZÁL( $\mathbf{f} \times \mathbf{B} \mathbf{1}$ ) else return semmi

```

**15.6. ábra.** Egy állandó  $d$  lépésnyi időkülönbözzettel simító algoritmus, folyamatos működésű (online) algoritmusként megvalósítva: egy új időpontbeli megfigyelésre kiadja az új simított becslést

## 15.4. KALMAN-SZÜRŐK

Képzeljünk el egy kismadarat, amint a dzsungel sűrűjében repül szürkületkor: a mozgásának csupán rövid, szakaszos felvillanásait pillanthatjuk meg; minden igyekezünkkel azt próbáljuk megjósolni, hogy hol van most a madár, és legközelebb hol fog

felbukkanni, hogy nehogy szem elől tévesszük. Vagy képzeljük azt, hogy a második világháborúban radarkezelők vagyunk, egy gyenge, mozgó radarjelet kérlelve, ami 10 másodpercenként jelenik meg a képernyőn. Vagy, kissé még távolabba visszalépve, képzeljük magunkat Kepler helyébe, ahogy a bolygók mozgását próbálja rekonstruálni igen pontatlan szögmérések sokaságából, amit rendszertelen és pontatlanul mért intervallumokban rögzítettek. Mindegyik esetben egy fizikai rendszer állapotát próbáljuk megbecsülni (helyet és sebességet például) időben egymást követő zajos megfigyelések ből. A problémát megfogalmazhatjuk egy időbeli valószínűségi modellben való következtetés-ként, ahol az állapotátmenet-modell a mozgás fizikáját írja le, az érzékelő modell pedig a mérési folyamatot. Ez a fejezet azokat a speciális reprezentációkat és következtetési algoritmusokat vizsgálja, amelyeket ilyen típusú problémákra fejlesztettek ki; a tárgyalt eljárást Kalman-szűrésnek (**Kalman filtering**) nevezik, a kidolgozója Rudolf E. Kalman<sup>4</sup> után.

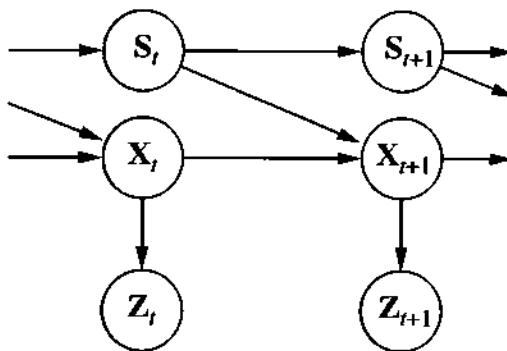
Világos, hogy több *folytonos* változóra lesz szükségünk a rendszer állapotának megadásához. Például a madár repülése megadható egy  $(X, Y, Z)$  pozíciójával és egy  $(\dot{X}, \dot{Y}, \dot{Z})$  sebességgel minden egyes időpillanatban. Szükségünk lesz még alkalmas feltételes sűrűségfüggvényekre az állapotátmenet- és érzékelő modellekhez; a 14. fejezetben hasonlóan, lineáris Gauss-eloszlásokat (**linear Gaussian**) fogunk használni. Ez azt jelenti, hogy a következő  $X_{t+1}$  állapot a jelenlegi  $X_t$  állapot lineáris függvénye, amihez még egy Gauss-zaj adódik, amely feltétel a gyakorlatban igen elfogadhatónak bizonyul. Tekintsük például a madár  $X$  koordinátáját, pillanatnyilag figyelmen kívül hagyva a többi koordinátát. Legyen a megfigyelések közötti intervallum  $\Delta$ , és tételezzünk fel állandó sebességet; ekkor a pozíció frissítésére az adódik, hogy

$$X_{t+\Delta} = X_t + \dot{X}\Delta$$

Ha Gauss-zajt adunk hozzá, akkor egy lineáris Gauss-féle állapotátmenet-modellt kapunk:

$$P(X_{t+\Delta} = x_{t+\Delta} | X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t \Delta, \sigma^2(x_{t+\Delta}))$$

Az  $X_t$  pozíció és  $\dot{X}_t$  sebességek alkotta rendszerhez tartozó Bayes-hálóstruktúra a 15.7. ábrán látható. Vegyük észre, hogy ez a lineáris Gauss-modellnek egy nagyon



**15.7. ábra.** Egy  $X_t$  hely,  $\dot{X}_t$  sebesség és  $Z_t$  helymegfigyelés alkotta lineáris dinamikus rendszerhez tartozó Bayes-hálóstruktúra

<sup>4</sup> Rudolf E. Kalman (Kálmán Rudolf) magyar származású matematikus.

speciális alakja; az általános alakot a fejezetben később írjuk le, ami alkalmazások nagyon széles körét fedi le az első bekezdés egyszerű mozgásos példáin túl. Az olvasónak hasznos lehet az A) függelékben átnézni a Gauss-eloszlások egyes matematikai tulajdonságait; a jelenlegi céljainkhoz a legfontosabb, hogy egy többváltozós Gauss-eloszlást (**multivariate Gaussian**)  $d$  változó esetén egy  $d$  elemű  $\mu$  átlag és egy  $d \times d$ -s  $\Sigma$  kovarianciamátrix ad meg.

## Gauss-eloszlások frissítése

A 14. fejezetben hivatkoztunk a lineáris Gauss-eloszlások családjának egy alaptulajdon-ságára: az eloszláscsalád zárt a standard Bayes-hálóbeli műveletekre. Most ezt az állítást pontosítjuk az időbeli valószínűsgégi modellben végzett szűrés esetére. A megkövetelt tulajdonságok megfelelnek a szűrés (15.3) egyenletben megadott kétrépéses számításának:

1. Ha a jelenlegi  $P(X_t | e_{1:t})$  eloszlás Gauss-eloszlás és a  $P(X_{t+1} | x_t)$  állapotátmenet-modell lineáris Gauss-modell, akkor az egylépéses előrejelzés eloszlása

$$P(X_{t+1} | e_{1:t}) = \int_{x_t} P(X_{t+1} | x_t) P(x_t | e_{1:t}) dx_t, \quad (15.15)$$

szintén Gauss-eloszlás.

2. Ha az előrejelzés  $P(X_{t+1} | e_{1:t})$  eloszlása Gauss-eloszlás és a  $P(e_{t+1} | X_{t+1})$  érzékelő modell lineáris Gauss-modell, akkor az új bizonyítékkal, mint feltétellel, a frissített eloszlás

$$P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t}) \quad (15.16)$$

szintén Gauss-eloszlás.

Így a Kalman-szűrés ELŐRE művelete fogad egy  $\mu$ , átlaggal és  $\Sigma$ , kovarianciamátrixszal meghatározott  $f_{1:t}$ , Gauss előre üzenetet, és előállít egy új többváltozós  $f_{1:t+1}$  Gauss előre üzenetet  $\mu_{t+1}$  átlaggal és  $\Sigma_{t+1}$  kovarianciamátrixszal. Így, ha egy  $f_{1:0} = P(X_0) = N(\mu_0, \Sigma_0)$  Gauss-eloszlással indulunk, egy lineáris Gauss-modellel való szűrés az állapotokon Gauss-eloszlást eredményez minden időpontban.

 Ez tetszetős és elegáns eredménynek tűnik, de miért is olyan fontos? Ennek a magyarázata az, hogy a most tárgyalt esethez hasonló néhány speciális esetet kivéve, a szűrés folytonos vagy hibrid (diszkrét és folytonos) hálókkal olyan állapoteloszlásokat generál, amelyek reprezentációja az idővel korlátozottan nő. Ezt az állítást általában nem könnyű bizonyítani, de a 15.5. feladat egy egyszerű példán mutatja be, hogy mi történik.

## Egy egyszerű egydimenziós példa

Az állítottuk, hogy a Kalman-szűrés ELŐRE művelete egy Gauss-eloszlást egy új Gauss-eloszlásba visz át. Ez új átlag- és kovarianciamátrix kiszámítását jelenti az előző átlagból és kovarianciamátrixból. Az általános (többváltozós) eset frissítési szabályának származtatása igen sok lineáris algebrai lépést igényel, így egyelőre a nagyon egyszerű egyváltozós esetnél maradunk, és később adjuk meg az eredményeket az általános esetre. A számítások még az egyváltozós esetre is unalmasak kissé, de úgy érezzük, hogy

érdesem látni őket, mivel a Kalman-szűrő hasznossága olyan szorosan kötődik a Gauss-eloszlások matematikai tulajdonságaihoz.

A tárgyalt időbeli modell egyetlen folytonos  $X_t$  állapotváltozós **véletlen bolyongást** (**random walk**) ír le egy zajos  $Z_t$  megfigyeléssel. Egy példa lehet erre a „vásárlói bizalom” mutatója, amit modellezhetünk úgy, mint ami havonként egy véletlen Gauss-eloszlású változáson megy át, és amelyet egy véletlen vásárlói kérdőív mér fel, ami szintén Gauss-mintavételi zajt okoz. Az a priori eloszlást gaussinak tételezzük fel  $\sigma_0^2$  szórásnégyzettel:

$$P(x_0) = \alpha e^{-\frac{1}{2} \left( \frac{(x_0 - \mu_0)^2}{\sigma_0^2} \right)}$$

(Az egyszerűség kedvéért ebben a fejezetben ugyanazt az  $\alpha$  szimbólumot fogjuk használni minden normalizációs állandó jelölésére.) Az állapotátmenet-modell egyszerűen a jelenlegi állapot egy Gauss-eloszlású módosítását jelenti állandó  $\sigma_x^2$  szórásnégyzettel:

$$P(x_{t+1} | x_t) = \alpha e^{-\frac{1}{2} \left( \frac{(x_{t+1} - x_t)^2}{\sigma_x^2} \right)}$$

Az érzékelő modell egy Gauss-eloszlású zajt tételez fel  $\sigma_z^2$  szórásnégyzettel:

$$P(z_t | x_t) = \alpha e^{-\frac{1}{2} \left( \frac{(z_t - x_t)^2}{\sigma_z^2} \right)}$$

Most, az a priori  $P(X_0)$  eloszlás ismeretében kiszámíthatjuk az egylépéses előrejelzés eloszlását felhasználva a (15.15) egyenletet:

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1 | x_0) P(x_0) dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left( \frac{(x_1 - x_0)^2}{\sigma_x^2} \right)} e^{-\frac{1}{2} \left( \frac{(x_0 - \mu_0)^2}{\sigma_0^2} \right)} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left( \frac{\sigma_0^2 (x_1 - x_0)^2 + \sigma_x^2 (x_0 - \mu_0)^2}{\sigma_0^2 \sigma_x^2} \right)} dx_0 \end{aligned}$$

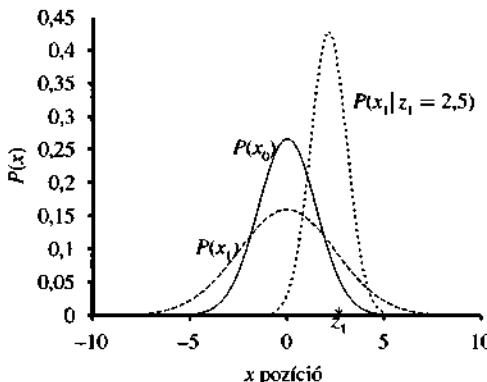
Az integrál igen bonyolultnak néz ki. A továbbhaladás lehetőségét annak észrevétele jelenti, hogy a kitevő két olyan kifejezés összege, amelyek négyzetesek  $x_0$ -ban, és így az maga is négyzetes  $x_0$ -ban. Egy egyszerű trükk, amit teljes négyzetté kiegészítésként (**completing the square**) ismerünk, lehetővé teszi bármely  $ax_0^2 + bx_0 + c$  átírását egy  $a \left( x_0 - \frac{-b}{2a} \right)^2$  négyzetes tag és egy  $c - \frac{b^2}{4a}$   $x_0$ -tól független maradék tag összegévé. A maradék tag az integrálból kivihető, így azt kapjuk, hogy

$$P(x_1) = \alpha e^{-\frac{1}{2} \left( c - \frac{b^2}{4a} \right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left( a(x_0 - \frac{-b}{2a})^2 \right)} dx_0$$

Most az integrál pontosan egy Gauss-eloszlás teljes tartomány feletti integrálja, ami egyszerűen 1-et ad. Így csupán a maradék tag maradt a négyzetesből.

A második kulcslépés annak észrevétele, hogy a maradék tagnak  $x_1$ -ben négyzetesnek kell lennie; valóban, egyszerűsítés után azt kapjuk, hogy

$$P(x_1) = \alpha e^{-\frac{1}{2} \left( \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2} \right)}$$



**15.8. ábra.** A Kalman-szűrés frissítési ciklusának a lépései egy véletlen bolyongás esetén. A véletlen bolyongás a priori eloszlása  $\mu_0 = 0.0$  és  $\sigma_0 = 1.0$ , az átmenet bizonytalansága  $\sigma_x = 2.0$ , az érzékelő bizonytalansága  $\sigma_z = 1.0$ , az első megfigyelés pedig  $z_1 = 2.5$  (az  $x$  tengelyen csillaggal jelölve). Végyük észre, ahogy a  $P(x_1)$  előrejelzés ellapul a  $P(x_0)$ -hoz képest az átmenet bizonytalansága miatt. Végyük azt is észre, hogy az a posteriori  $P(x_1|z_1)$  kissé balra helyezkedik el a  $z_1$  megfigyeléstől, mivel az átlag az előrejelzés és a megfigyelés súlyozott átlaga.

Azaz az egylépéses előrejelzés eloszlása Gauss-eloszlás, ugyanazzal a  $\mu_0$  átlaggal, a szórásnégyzet pedig egyenlő az eredeti  $\sigma_0^2$  szórásnégyzetnek és az átmenet  $\sigma_x^2$  szórásnégyzetének az összegével. Egy pillanatnyi belegondolás után ez szemléletesen is elfogadhatónak tűnik.

A frissítés lépéseinak befejezéséhez még szükséges, hogy az első időpontbeli megfigyelést,  $x_1$ -t feltételként vegyük számításba. A (15.16) egyenletből erre az adódik, hogy

$$P(x_1|z_1) = \alpha P(z_1|x_1)P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(z_1-x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}$$

Most ismét kombináljuk a kitevőket, és négyzetté egészítük ki (15.6 feladat), azt kapva, hogy

$$P(x_1|z_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1-\frac{(\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})^2}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)} \quad (15.17)$$

így egy frissítési ciklus után egy új Gauss-eloszlásunk van az állapotváltozóra.

A (15.17) egyenletben a Gauss-alakból láthatjuk, hogy az új átlag és szórás kiszámítható a régi átlagból és szórásból a következőképpen:

$$\begin{aligned} \mu_{t+1} &= \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2\mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \\ \sigma_{t+1}^2 &= \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \end{aligned} \quad (15.18)$$

A 15.8. ábrán látható egy frissítési ciklus az állapotátmenet- és az érzékelő modell konkrét értékei esetén.

Az előző egyenletpár pontosan ugyanazt a szerepet játsza, mint az *általános szűrés egyenlete* (15.3) vagy az *RMM-szűrés egyenlete* (15.10). A Gauss-eloszlások speciális tulajdonsága miatt azonban az egyenleteknek van néhány további érdekes tulajdonsága. Először is, hogy az új  $\mu_{t+1}$  átlag kiszámítása értelmezhető úgy, mint egyszerűen az új  $z_{t+1}$  megfigyelés és a régi  $\mu_t$  átlag *súlyozott átlaga*. Ha a megfigyelés megbízhatatlan, akkor  $\sigma_z^2$  nagy, és több figyelmet szentelünk a régi átlagnak; ha a régi átlag megbízhatatlan ( $\sigma_x^2$  nagy), vagy a folyamat nehezen megjósolható ( $\sigma_x^2$  nagy), akkor több figyelmet szentelünk a megfigyelésnek. Másodszor, vegyük észre, hogy a  $\sigma_{\mu_{t+1}}^2$  szórásnégyzet frissítése *független a megfigyeléstől*. Ezért előre kiszámíthatjuk, hogy mi lesz a szórásnégyzetértékek sorozata. Harmadszor, a szórásnégyzetértékek sorozata gyorsan konvergál egy adott értékhez, ami csak  $\sigma_x^2$ -től és  $\sigma_z^2$ -től függ, ezáltal lényegesen egyszerűsítve az elkövetkező számításokat (lásd 15.7. feladat).

## Az általános eset

Az előző levezetés szemléletesen bemutatta a Gauss-eloszlásoknak azt az alaptulajdonsgát, ami a Kalman-szűrés működését lehetővé teszi: azt a tényt, hogy az exponens négyzetes alakú. Ez nem csak az egy változós esetre igaz; a teljes több változós Gauss-eloszlás alakja:

$$N(\mu, \Sigma)(x) = \frac{1}{2} \pi^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} ((x - \mu)^T \Sigma^{-1} (x - \mu))$$

A kitevőben lévő tényezők összeszorzása láthatóvá teszi, hogy a kitevő is négyzetes függvénye az  $x$ -ben lévő  $x_i$  valószínűségi változóknak. Ahogy az egy változós esetben, a szűrési frissítés megőrzi az állapoteloszlás gaussi voltát.

Elsőként definiáljuk a Kalman-szűrésnél használt általános időbeli modellt. Mind az állapotátmenet-modell, mind az érzékelő modell lineáris transzformációt enged meg additív Gauss-zajjal. Így azt kapjuk, hogy

$$\begin{aligned} P(x_{t+1} | x_t) &= N(Fx_t, \Sigma_x)(x_{t+1}) \\ P(z_t | x_t) &= N(Hx_t, \Sigma_z)(z_t) \end{aligned} \quad (15.19)$$

ahol az  $F$  és a  $\Sigma_x$  mátrixok a lineáris állapotátmenet-modellt és az átmeneti zaj kovarianciáját írják le, a  $H$  és a  $\Sigma_z$  pedig az érzékelő modell megfelelő mátrixai. Ekkor az átlag és a kovariancia frissítésének az egyenletei a maguk rémisztő valóságában:

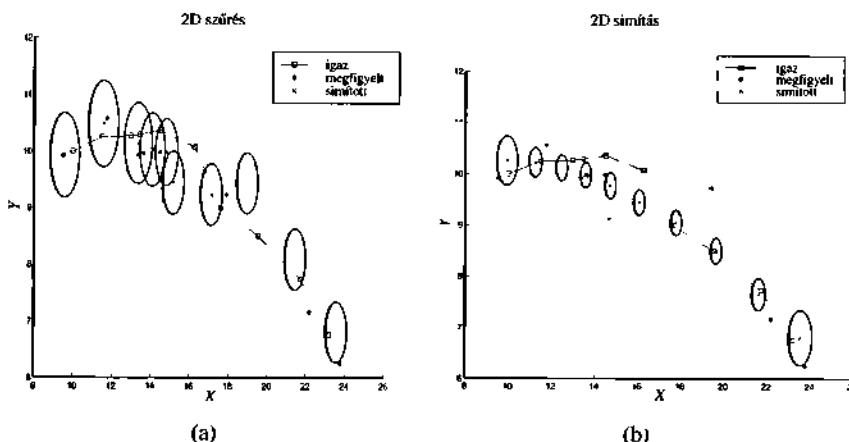
$$\begin{aligned} \mu_{t+1} &= F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \\ \Sigma_{t+1} &= (I - K_{t+1})(F\Sigma_t F^T + \Sigma_x) \end{aligned} \quad (15.20)$$

ahol a  $K_{t+1} = (F\Sigma_t F^T + \Sigma_x)H^T (H(F\Sigma_t F^T + \Sigma_x)H^T + \Sigma_z)^{-1}$  mennyiség neve a **Kalman-erősítés-mátrix (Kalman gain matrix)**. Akár hihető, akár nem, ezeknek az egyenleteknek lehetséges egy szemléletes értelmezése. Például, gondoljuk meg a  $\mu$  állapotátlag-becslés frissítését. Az  $F\mu_t$  tag a  $t+1$  időpontban előre jelzett állapot, így  $HF\mu_t$  az előre jelzett megfigyelés. Ezért a  $z_{t+1} - HF\mu_t$  tag az előre jelzett megfigyelés hibáját reprezentálja. Ez van megszorozva a  $K_{t+1}$  tényezővel az előre jelzett állapot korrigálásához; így a  $K_{t+1}$  annak mértéke, hogy mennyire kell figyelembe venni az új megfigyelést az előrejelzéshez képest. Ahogyan a (15.18) egyenletben, az a tulajdonság

most is fennáll, hogy a szórásnégyzet frissítése független a megfigyeléstől. A  $\Sigma_t$  és a  $K_t$  értékek sorozata ezért előre (offline) is kiszámolható, és a követés közben szükséges konkrét számítások igen mérsékelték.

Hogy bemutassuk az egyenletek működését egy  $X-Y$  síkon mozgó tárgy követésének a problémájára alkalmaztuk őket. Az állapotváltozók az  $\mathbf{X} = (X, Y, \dot{X}, \dot{Y})^\top$ , így az  $\mathbf{F}$ ,  $\Sigma_x$ ,  $\mathbf{H}$  és  $\Sigma_z$  4 × 4-es mátrixok. A 15.9. (a) ábrán látható az igazi pályagörbe, a zajos megfigyelések sorozata és a Kalman-szűréssel becsült pályagörbe végig a kovarianciával, amit az egységnyi szórás mérettő körfonalak jelzik. A szűrési folyamat jól teljesít az aktuális mozgás követésében, és ahogy várható, a szórásnégyzet gyorsan beáll egy rögzített értékre.

Ahogyan szűrésre, úgy simításra is származtathatók egyenletek lineáris Gauss-modell esetén. A simítás eredményei a 15.9. (b) ábrán láthatók. Vegyük észre, hogy a helyzetbecslés szórásnégyzete gyorsan lecsökken, a pályagörbe végét leszámítva (itt miért nem?), és hogy a simítással becsült pályagörbe sokkal egyenletesebb.



**15.9. ábra.** (a) A Kalman-szűrés eredménye egy  $X-Y$  síkon mozgó objektumnál, feltüntetve a valódi (balról jobbra haladó) pályagörbét, a zajos megfigyelések egy sorozatát és a Kalman-szűrés alapján becsült pályagörbét. A helybecslések szórásait az oválisok jelzik. (b) A Kalman-simítás eredménye ugyanarra a megfigyelési sorozatra.

## A Kalman-szűrés alkalmazhatósága

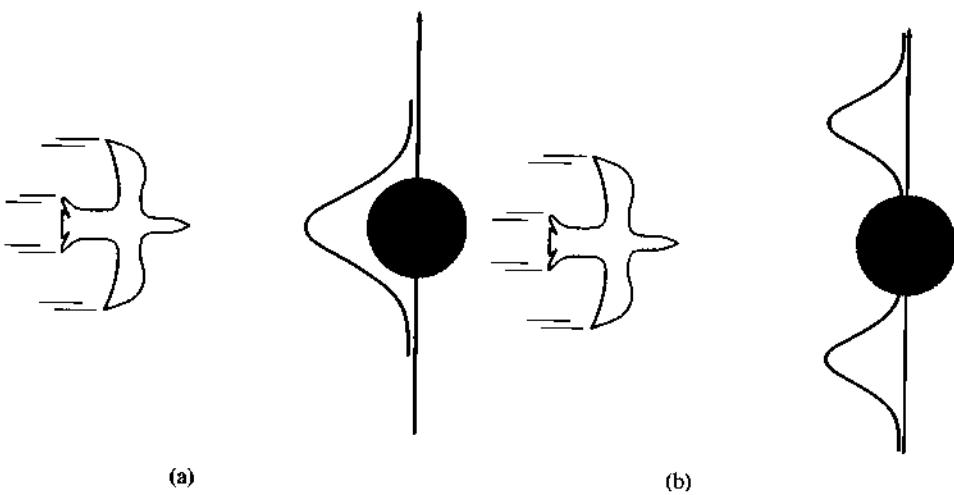
A Kalman-szűrést és kiterjesztésein alkalmazások széles körében használják. A „klaszszikus” alkalmazás légi járművek és rakéták radar követése. Kapcsolódó alkalmazások között van a tengeralattjárók és földi járművek akusztikai követése, és járművek és emberek képi követése. Egy kissé elvontabb szemlélettel a Kalman-szűrőket részecskék buborékkamrás pályagörbéjének rekonstrukciójára, valamint óceáni áramlások felszíni műholdképek alapján történő rekonstrukciójára használják. Az alkalmazások köre sokkal szélesebb, mint csupán mozgások követése: bármely rendszerre alkalmazható, ami folytonos állapotváltozókkal és zajos megfigyelésekkel jellemezhető. Ilyen rendszerek

magukban foglalnak zúzdákat, vegyi üzemeket, nukleáris reaktorokat, növényi ökoszisztémákat és nemzetgazdaságokat.

A Kalman-szűrés alkalmazhatóságának ténye egy rendszerre nem jelenti, hogy az eredmények érvényesek vagy hasznosak lesznek. A feltevések – lineáris Gauss-féle állapotátmenet-modell és érzékelő modell – igen erősek. A kiterjesztett Kalman-szűrő (KKSZ) (extended Kalman filter, EKF) megröbál úrrá lenni a modellezett rendszer nemlineáris tulajdonságain. Egy rendszer nemlineáris, ha az állapotátmenet-modell nem írható le, mint az állapotvektor mátrixszorzata, ahogyan ez a (15.19) egyenletben szerepel. A KKSZ úgy működik, hogy rendszert  $x_t$ -ben *lokálisan* lineárisként modellez az  $x_t = \mu$ , környezetében, ahol  $\mu$ , a jelenlegi állapoteloszlás átlaga. Ez jól működik „sima”, „jó magaviseletű” rendszereknél, és lehetővé teszi a követőnek, hogy egy olyan Gauss-állapoteloszlást tartson nyilván és frissítsen, ami elfogadható közelítése az igazi a posteriori eloszlásnak.

Azonban mit is jelent, hogy egy rendszer „nem sima” vagy „rossz magaviseletű”? Technikailag ez azt jelenti, hogy jelentős nemlinearitás van jelen a rendszer viselkedésében abban a régióban, ami „közel” van a jelenlegi  $\mu$ , átlaghoz (a  $\Sigma$ , kovarianciamátrix szerint). Ennek a tulajdonságnak a nem technikai megértéséhez gondoljunk arra a példára, amikor egy dzsungelben szálló madár követésével próbálkoztunk. A madár nagy sebességgel egyenesen egy fatörzs felé tart. A Kalman-szűrő, akár reguláris, akár kiterjesztett a madár pozíciójára, csak Gauss-előrejelzést adhat, és ennek a Gaussnak az átlaga a fatörzs közepére esik, ahogyan a 15.10. (a) ábrán látható. A madár egy elfogadható modellje azonban egy elkerülő műveletet jelezne előre egyik vagy másik oldalra, ahogy az a 15.10. (b) ábrán látható. Egy ilyen modell erősen nemlineáris, mivel a madár döntése nagyon eltérő a fatörzshöz vett pontos pozíciójának függvényében.

Ilyen példák kezeléséhez nyilvánvalóan egy kifejezőbb nyelvre van szükségünk a modellezett rendszer viselkedésének a reprezentálásához. A szabályozáselmélet területén, ahol olyan problémák, mint például egy repülőgép elkerülő manőverezése ugyan-



**15.10. ábra.** Egy fa felé repülő madár (felülnézetben). (a) Egy Kalman-szűrő előrejelzése a madár helyzetré, ami egyetlen Gauss-eloszlás az akadály közepére illesztve. (b) Egy valósághűbb modell számításba veszi a madár elkerülő manővereit, és azt jelzi előre, hogy az egyik vagy a másik oldalon fog elszállni.

ilyen típusú bonyodalmakat vetnek fel, a megszokott megoldás a váltó **Kalman-szűrő** (*switching Kalman filter*). Ebben a megközelítésben, több Kalman-szűrő fut párhuzamosan, mindegyik a rendszer különböző modelljét használva – például egy az egyenes repülésre, egy az éles balra fordulásra és egy az éles jobbra fordulásra. Az előrejelzéseknek egy súlyozott összegét használjuk, ahol a súly attól függ, hogy mennyire illeszkednek az egyes szűrők az aktuális adatokhoz. A következő fejezetben látni fogjuk, hogy ez egyszerűen az általános dinamikus Bayes-háló modell egy speciális esete, amit a 15.7. ábrán látható hálóból egy diszkrét „manőver” állapotváltozó hozzáadásával nyerünk. A váltó Kalman-szűrőket a 15.5. feladat tárgyalja tovább.

## 15.5. DINAMIKUS BAYES-HÁLÓK

Egy **dinamikus Bayes-háló** vagy **DBH** (*dynamic Bayesian network, DBN*) egy olyan Bayes-háló, ami egy a 15.1. alfejezetben leírt típusú időbeli valószínűségi modellt reprezentál. (Ahogy korábban is említettük az index időbeli értelmezése helyett bármely szekvenciális értelmezés lehetséges, gyakori például az indexek mint egy egydimenziós helyzeti pozíciónak az értelmezése is – *a ford.*) A DBH-kra már látunk példákat, a 15.2. ábrán az eseményös hálót és a 15.7. ábrán a Kalman-szűrő hálót. Általában egy DBH minden egyes szeletének tetszőleges számú állapotváltozója ( $X_t$ ) és bizonyíték-változója ( $E_t$ ) lehet. Az egyszerűség kedvéért fel fogjuk tenni, hogy a változók és kapcsolataik szeletről szeletre pontosan ismétlődnek, és hogy a DBH egy elsőrendű Markov-folyamatot reprezentál, így minden változónak csak a saját szeletében vagy a közvetlenül megelőző szeletben lehetnek szülei.

Nyilvánvaló, hogy minden rejtejtő Markov-modell reprezentálható mint egy DBH, egyetlen állapotváltozóval és egyetlen bizonyítékváltozóval. Az is fennáll, hogy minden diszkrét változós DBH reprezentálható mint egy RMM; ahogyan a 15.3. alfejezetben megmutattuk, a DBH összes állapotváltozója összekombinálható egyetlen állapotváltozóvá, aminek az értékei az egyes állapotváltozók értékeinek az összes lehetséges együttese. Azonban ha minden RMM egy DBH, és minden DBH átfordítható egy RMM-be, akkor mi a különbség? A különbség abban rejlik, hogy *egy komplex rendszer állapotának az öt alkotó változóra történő dekomponálásával a DBH képes kihasználni az időbeli valószínűségi modell ritkaságát*. Tegyük fel például, hogy egy DBH-nak 20 bináris állapotváltozója van, amelyek mindegyikének három szülője van az előző szeletben. Ekkor a DBH állapotátmennet-modellje  $20 \times 2^3 = 160$  valószínűségi értéket tartalmaz, míg a hozzá tartozó RMM-nek  $2^{20}$  állapota és ezért  $2^{40}$ , azaz durván egybillió átmenet-valószínűség értéke van az állapotátmennet-mátrixban. Ez legalább három ok miatt is rossz: először, hogy az RMM maga sokkal több tárat igényel, másodszor, hogy a hatalmas állapotátmennet-mátrix az RMM-következetést sokkal költségesebbé teszi, harmadszor, hogy ilyen hatalmas számu paraméter megtanulásának problémája a tiszta RMM-modelleket alkalmazhatatlanná teszi nagy problémák esetén. A DBH-k és az RMM-ek közötti kapcsolat nagyjából analóg a hagyományos Bayes-hálók és a teljes táblázatos együttes eloszlások közötti kapcsolathoz.

Már megmutattuk, hogy minden Kalman-szűrő reprezentálható egy DBH-ban folytonos változókkal és lineáris Gauss feltételes eloszlásokkal (lásd 15.7. ábra). Az előző fejezet végén tárgyaltak miatt azzal is tisztában kell lenni, hogy *nem minden* DBH reprezentálható egyetlen Kalman-szűrő modellel. Egy Kalman-szűrőnél az aktuális állapotelosz-

lás minden egy egyedüllálló többváltozós Gauss-eloszlás – azaz egyetlen „dudor” egy konkrét helyen. A DBH-k ezzel szemben tetszőleges eloszlást képesek modellezni. Számos valós alkalmazásnál ez a rugalmasság elengedhetetlen. Gondoljuk meg például a kulcsaim jelenlegi helyét. Lehetnek a zsebeimben, az éjjeli szekrényen, a konyhapulton vagy a bejárati ajtóban himbálódzva. Egyetlen Gauss-dudornak, ami ezt az összes helyet magában foglalja, jelentős valószínűséget kell rendelni ahhoz, hogy a kulcsok az előszobában a levegőben vannak. A valós világ olyan aspektusai, mint céltudatos ágensek, akadályok és zsebek „nemlineáritásokat” vezetnek be, amelyek diszkrét és folytonos változók kombinációját igénylik, hogy egy elfogadható modellhez jussunk.

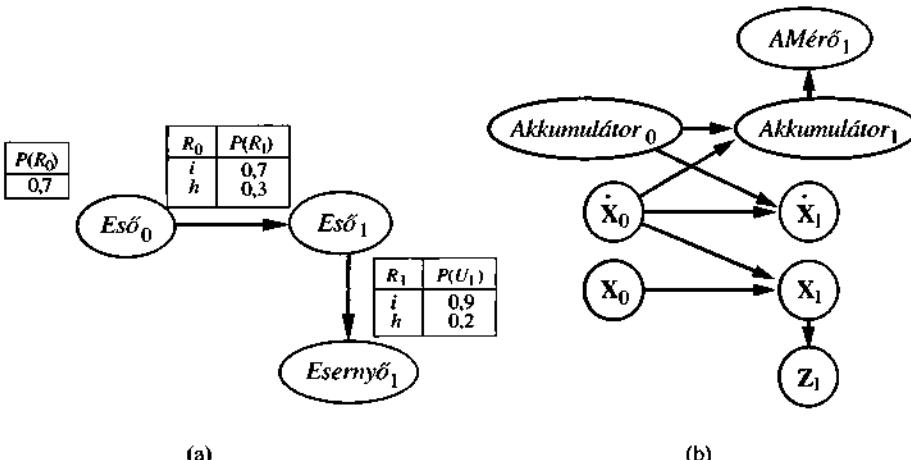
## DBH-k létrehozása

Egy DBH létrehozásához háromfajta információt kell megadni: a  $P(X_0)$  a priori eloszlást az állapotváltozók felett; a  $P(X_{t+1}|X_t)$  állapotátmenet-modellt; és a  $P(E_t|X_t)$  érzékelő modellt. Az állapotátmenet- és érzékelő modell megadásához, meg kell adni az egymást követő szeletek közötti és az állapot- és bizonyítékváltozók közötti kapcsolatok topológiáját. Mivel az állapotátmenet- és érzékelő modelleket stacionáriusnak – minden  $t$ -re azonosnak – tételezzük fel, a legkényelmesebb egyszerűen az első szeletre megadni őket. Például az esernyős világ teljes DBH-specifikációját megadja az a három csomópontos háló, ami a 15.11. (a) ábrán látható. Ebből a specifikacióból a teljes (egyik irányban végtelen) DBH megkonstruálható az első szelet megfelelő másolásával.

Gondoljunk most át egy érdekesebb példát: egy akkumulátorhajtású, az  $X-Y$  síkon mozgó robot követését, ahogyan azt a 15.1. alfejezetben bevezettük. Elsőként szükségeünk van állapotváltozóra, amelyek mind az  $X_t = (X_r, Y_r)$  pozíció-, mind az  $\dot{X}_t = (\dot{X}_r, \dot{Y}_r)$  sebességg komponenseket magukban foglalják. Feltételezünk valamilyen pozíciómérő módszert – akár egy rögzített kamerát vagy egy felszíni GPS-t (Global Positioning System) –, ami a  $Z_t$  méréseket eredményezi. A pozíció a következő időpontban a jelenlegi pozíciótól és sebességtől függ, ugyanúgy, mint a standard Kalman-szűrő modellen. A sebesség a következő időpontban a jelen sebességtől és az akkumulátor állapotától függ. Felvesszünk még egy Akkumulátor, változót az aktuális akkumulátor töltési szintjének reprezentálására, aminek szülői az előző akkumulátorszint és a sebesség, és felvesszünk még egy AMérő, változót, ami az akkumulátor töltési szintjét méri. Ez szolgáltatja a 15.11. (b) ábrán látható alapmodellt.

Érdemes részletesebben megvizsgálni az AMérő-hez tartozó érzékelőmodell tulajdonságait. Tegyük fel az egyszerűség kedvéért, hogy mind az Akkumulátor, mind az AMérő, diszkrét értékeket vehetnek fel 0-tól 5-ig – igen hasonlóan egy tipikus laptop számítógép akkumulátor mérőjéhez. Ha a mérő minden pontos akkor a  $P(AMérő_t|Akkumulátor_t)$  FVT-nek végig 1,0 valószínűségeket kell tartalmaznia az „átjáróban” és 0,0 valószínűségeket máshol. A valóságban a mérésekben a zaj minden felbukkan. Folytonos mérésekre, egy kis szórású Gauss-eloszlást lehetne ehelyett használni.<sup>5</sup> A diszkrét változóinknál a Gauss-eloszlást egy olyan eloszlással közelíthetjük, amelyben a hiba valószínűsége a megfelelő módon csökken, így nagy hiba valószínűsége igen kicsi.

<sup>5</sup> Szigorúan véve egy Gauss-eloszlás problématiskus, mivel pozitív valószínűséget rendel nagy negatív töltési szintekhez. A béta-eloszlás gyakran jobb választás az olyan változókra, amelyeknek tartománya korlátos.

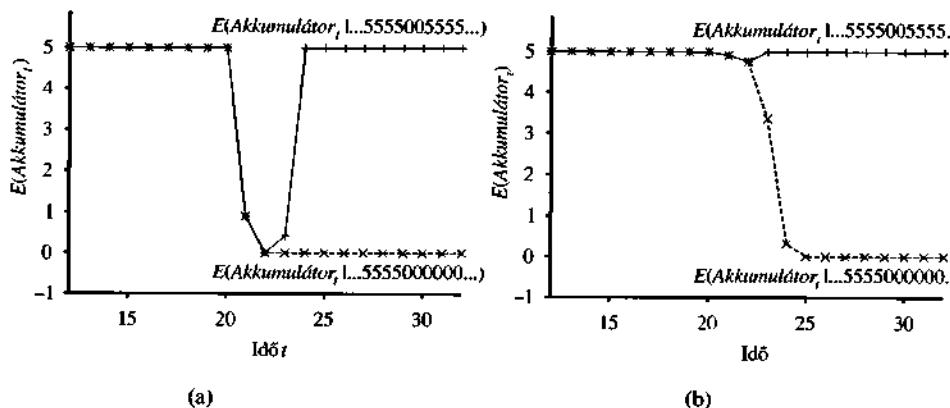


15.11. ábra. (a) Az eseményös DBH-hoz tartozó a priori eloszlás, az állapotátmenet-modell és az érzékelő modell megadása. Az összes következő szelet feltevéstünk szerint az 1. szelet másolata. (b) Egy  $X-Y$  síkon való robotmozgáshoz tartozó egyszerű DBH.

**A Gauss-hibamodell (Gaussian error model)** kifejezést fogjuk használni mind a folytonos, mind a diszkrét változatokra.

A robotikában bárki, aki egy kevés gyakorlattal rendelkezik a számítógépes folyamatvezérlésben vagy az automatikus érzékelés más eseteiben, rögtön tanúskodik arról a tényről, hogy a kis mennyiségű mérési hiba gyakran a legkevésbé fontos a problémák között. Ugyanis a valódi érzékelők elromlanak, és amikor egy érzékelő elromlik, akkor nem küld szükségszerűen egy jelet azzal, hogy „Ó, egyébiránt az adat, amit küldök, egy halom zagyvaság.” Ehelyett egyszerűen küldi a zagyvaságot. A meghibásodás legegyszerűbb formáját **átmeneti hibának (transient failure)** nevezik, amikor az érzékelő alkalmasként elhatározza, hogy valami zagyvaságot küld. Például az akkumulátor színtérzékelőjének lehet olyan szokása, hogy nullát küld, amikor valaki meglöki a robotot, még akkor is, ha az akkumulátor teljesen fel van töltve.

Lássuk, mi történik, amikor egy átmeneti hiba bekövetkezik egy Gauss-hibamodellnél, ami nem tudja kezelni az ilyen hibákat. Tételezzük fel például, hogy a robot csendesen üldögél, és 20 egymást követő mérés az akkumulátoron 5-öt ad. Ezután az akkumulátor-mérő időleges kiesése miatt a következő mérés eredménye  $AMérő_{21} = 0$ . Mire kell következtetnünk egy egyszerű Gauss-hibamodellből az  $Akkumulátor_{21}$ -gyel kapcsolatban. A Bayes-szabály szerint a válasz függ mind a  $P(AMérő_{21} = 0 | Akkumulátor_{21})$  érzékelő modelltől, mind a  $P(Akkumulátor_{21} | AMérő_{1:20})$  előrejelzéstől. Ha egy nagy érzékelési hiba valószínűsége szignifikánsan kisebb, mint az  $Akkumulátor_{21} = 0$ -ba való átmenet valószínűsége, még ha ez utóbbi nagyon valószínűtlen is, akkor az a posteriori eloszlás nagy valószínűséget fog rendelni a lemerült akkumulátorhoz. Egy második nulla értékű mérési eredmény  $t = 22$ -nél ezt a konklúziót majdnem biztosá teszi. Ha az átmeneti hiba aztán eltűnik, és a jelzett érték visszaáll 5-re  $t = 23$ -tól, akkor az akkumulátorszint becslése gyorsan vissza fog állni 5-re, szinte varázsütésre. Ezt az eseménysort mutatja be a 15.12. (a) ábra felső görbeje, ami az  $Akkumulátor_t$ , vátható értékét mutatja idő szerint egy diszkrét Gauss-hibamodellt használva.



**15.12. ábra.** (a) Felső görbe: az  $Akkumulátor_t$  várható értékének a pályagörbje egy olyan megfigyelési sorozatnál, ami csak 5-ösöket tartalmaz, kivéve a 0-kat a  $t = 21$  és a  $t = 22$  időpontokban, egy egyszerű Gauss-hibamodell felhasználásával. Alsó görbe: a pályagörbe, amikor a megfigyelések 0-val folytatódnak  $t = 21$ -től. (b) Ugyanez a kísérlet az átmenetihiba-modellel elvégezve. Vegyük észre, hogy az átmeneti hibát jól kezeli, de a végleges hiba túlzott pesszimizmust eredményez.

A helyreállás ellenére előfordul ( $t = 22$ -nél), hogy a robot meg van győződve, hogy az akkumulátora lemerült; feltehetően ekkor ki kell küldenie egy segítség jelet, és le kell kapcsolnia. Sajnos a túlegyszerűsített érzékelő modellje tévűtra vezette. Hogyan lehet ezt kijavítani? Gondolunk egy ismerős példára az emberek minden napos vezetési gyakorlattából: éles kanyarokban vagy meredek hegyeken néha felvillan az „üzemanyagtartály üres” figyelmeztető lámpa. A segélykérő telefon keresése helyett inkább az szokott az ember eszébe jutni, hogy az üzemanyagmérő gyakran igen nagy hibákat ad, amikor az üzemanyag lötyög a tartályban. A történet tanulsága a következő: **az érzékelő hibájának helyes kezeléséhez a rendszer érzékelő modelljének tartalmaznia kell a meghibásodás lehetőségét.**

Egy érzékelő legegyszerűbb hibamodellje egy bizonyos valószínűséggel megengedi, hogy az érzékelő teljesen helytelen értéket adjon vissza, függetlenül a világ valódi állapotától. Például ha az akkumulátor mérő 0-t visszaadva hibázik, azt mondhatjuk, hogy

$$P(AMérő_t = 0 | Akkumulátor_t = 5) = 0.03$$

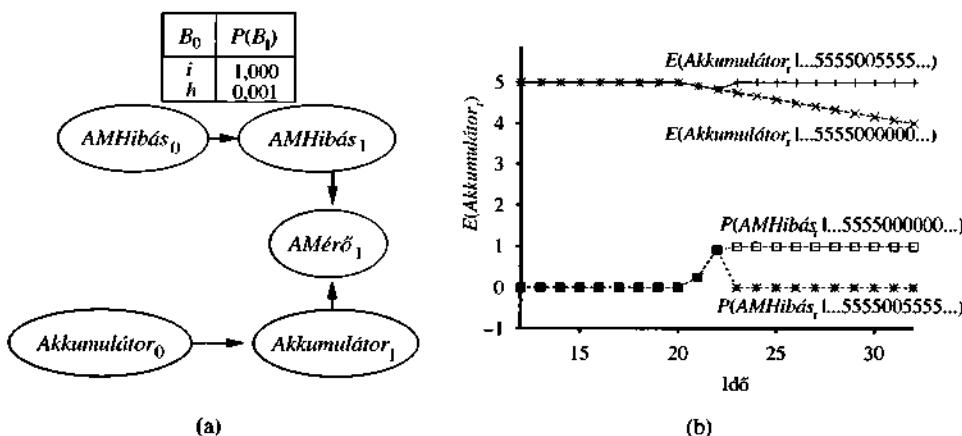
ami feltételezhetően sokkal nagyobb, mint az a valószínűség, amit az egyszerű Gauss-hibamodell rendelne hozzá. Hívjuk ezt az **átmenetihiba-modellnek (transient failure model)**. Hogyan segít ez, amikor a 0 jelzéssel kerülünk szembe? Feltételezve, hogy az *előre jelzett* valószínűség, hogy az akkumulátor lemerült, az eddigi jelzések alapján sokkal kisebb, mint 0.03, akkor az  $AMérő_{21} = 0$  megfigyelés legjobb magyarázata az, hogy az érzékelő időlegesen meghibásodott. Szemléletesen úgy képzelhetjük el a bizonyosságot az akkumulátor feltöltöttségéről, mint aminek van egy bizonyos mennyiségű „tehetetlensége”, ami segít úrrá lenni a mérőjelzés időleges zavarain. A 15.12. (b) ábra felső görbüjén látható, hogy az átmenetihiba-modell képes átmeneti hibákat kezelní a bizonyosság katasztrófális változásai nélkül.

Ennyit hát az időleges zavarokról. Mi a helyzet az érzékelő állandó meghibásodásával? Sajnos, az ilyenfajta meghibásodások túlságosan is gyakoriak. Ha az érzékelő húsz

5-ös jelzést ad, amit húsz 0-s jelzés követ, akkor az előző bekezdésben leírt átmenetihiba-modell azt fogja eredményezni, hogy a robot fokozatosan azt kezdi hinni, hogy az akkumulátora lemerült, amikor valójában az is lehet, hogy az érzékelő hibásodott meg. A 15.12. (b) ábrán látható alsó görbe mutatja a bizonyosság „pályagörbét” erre az esetre. A  $t = 25$  időpontra – 5 darab 0 jelzésre – a robot meg van győződve, hogy az akkumulátora üres. Nyilvánvaló, hogy jobban szeretnénk, ha a robot azt hinné, hogy az akkumulátor mérője romlott el – ha valóban ez a valószínűbb esemény.

Nem meglepő, hogy egy tartós hiba kezeléséhez szükségünk lesz egy **tartóhiba-modellre (persistent failure model)**, ami leírja, hogyan működik az érzékelő normális körülmények között és meghibásodás után. Ennek eléréséhez ki kell egészítünk a rendszer rejtett állapotát egy további változóval, mondjuk egy  $AMHibás$ -sal, ami megadja az akkumulátor mérő állapotát. A hiba tartós megmaradását egy nyíllal kell modellezni, ami az  $AMHibás_0$ -t az  $AMHibás_1$ -hez kapcsolja. Ennek a **megmaradási nyílnak (persistence arc)** olyan FVT-je van, ami minden időlépésben megengedi a meghibásodást egy kis valószínűséggel, mondjuk 0,001-gyel, de leírja, hogy ha egy érzékelő elromlott, akkor az rossz is marad. Amikor az érzékelő rendben van, akkor az  $AMérő$  érzékelő modellje meggyezik az átmenetihiba-modellel; amikor az érzékelő rossz, az  $AMérő$  minden 0, függetlenül az akkumulátor aktuális töltöttségétől.

Az akkumulátor érzékelőjének tartóhiba-modellje a 15.13. (a) ábrán látható. A teljesítménye két adatsoron (az időleges zavarjelnél és a tartós meghibásodáson) a 15.13. (b) ábrán látható. Számos dolgot érdemes megfigyelni ezeken a görbékben. Először is, hogy az időleges zavarjelnél az érzékelő hibájának a valószínűsége jelentősen megemelkedik a második 0 jelzés után, de azonnal visszaesik, ahogy egy 5-öst megfigyelt. Másodszor, hogy tartós hibánál az érzékelő hibájának a valószínűsége gyorsan megemelkedik 1 közelébe, és ott marad. Végül, hogy amikortól az érzékelő hibásnak hiszi, a robot csak azt tételezheti fel, hogy az akkumulátor „normális” sebességgel veszti el töltését, ahogy az  $E(Akkumulátor_t|...)$  fokozatosan csökkenő szintje ezt mutatja.



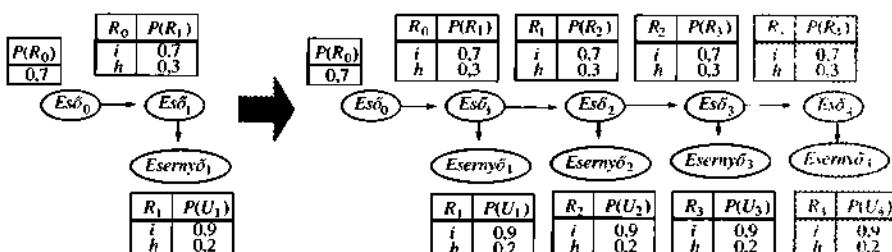
Eddig csupán felületesen érintettük a komplex folyamatok reprezentálásának problémáját. Az állapotátmenet-modellek változatossága óriási, magában foglalva egymástól olyan távol eső témaikat, mint például az emberi endokrin rendszer (hormonális működés) és az autópályákon közlekedő járművek modellezését. Az érzékelő modellezés egymában is hatalmas terület, de még az olyan nehezen megfogható jelenségek, mint például az érzékelő elállítódása, hirtelen dekalibrálása és a külső körülményeknek (mint az időjárásnak) a hatásai az érzékelő jelzésein is kezelhetők explicit reprezentációkkal a dinamikus Bayes-hálóknál.

## Egzakt következtetés DBH-kban

Miután felvázoltunk néhány ötletet komplex folyamatok DBH-kkal történő reprezentálásáról, most a következtetés kérdését vizsgáljuk meg. Bizonyos értelemben ezt a kérdést már megválaszoltuk: a dinamikus Bayes-hálók Bayes-hálók, és nekünk már vannak algoritmusaink Bayes-hálóban történő következtetésre. Adott megfigyeléssorozat esetén a teljes Bayes-hálós reprezentáció felépíthető a szeletek megismétlésevel, ameddig a háló elég nagy nem lesz a megfigyelések befogadásához, ahogy a 15.14. ábrán látható. Ezt a technikát **kibontásnak (unrolling)** nevezik. (Technikailag a DBH ekvivalens egy egy irányban végtelen hálóval, amit végtelen számú kibontással kapunk. Az utolsó megfigyelés után hozzáadott szeleteknek nincs hatása a következtetésre a megfigyelési periódusban, ezért elhagyhatók.) Ha a DBH-t már kibontottuk, alkalmazhatók a következető algoritmusok: változó eliminálás, egyesítési fa módszere stb. (lásd 14. fejezet).

Sajnos a kibontás naiv alkalmazása nem volna valami hatékony. Ha szűrést vagy simitást szeretnénk elvégezni megfigyelések  $e_{1:t}$  hosszú sorozatán, akkor a kibontott háló tárigénye  $O(t)$  lenne, és korlát nélkül nőne, amint még több megfigyelést adnánk hozzá. Ráadásul ha egyszerűen újra lefuttatjuk a következető algoritmust minden időpontról, amikor megfigyelés érkezik, a következetési ideje frissítéseként szintén  $O(t)$  szerint fog növekedni.

A 15.2. alfejezetre visszatekintve látjuk, hogy állandó idő- és tárigény elérhető a szűrési frissítésnél, ha a számítások rekurzív módon végezhetők. Lényegében a szűrési frissítés (15.3) egyenlete az előző időponthoz tartozó állapotváltozók kiösszegzését (*summing out*) végzi el, hogy megkapja az új időponthoz tartozó eloszlást. Változók ki-



**15.14. ábra.** Egy dinamikus Bayes-háló kibontása: a szeletek ismétlődnek befogadva a megfigyelési sorozatot (a halvány csomópontok). További szeleteknek már nincs hatása a megfigyelési perióduson belüli következtetésre. Ha a DBH-t kibontottuk, bármely következetési algoritmus használható – változó eliminálás, egyesítési fa módszerek és így tovább –, ami a 14. fejezetben szerepel.

összegzése pedig pontosan az, amit a változóeliminálás (**variable elimination**) algoritmus elvégez (lásd 14.10. ábra), és megmutatható, hogy a változó eliminálás időrendi változókon futva pontosan a szűrés rekurzív frissítésének (15.3) egyenlet szerinti működését utánozza. A módosított algoritmus egyszerre legfeljebb két szeletet tart a memóriában: kezdve a 0. szeettel, hozzáadjuk az 1. szeletet, kiösszegezzük a 0. szeletet, aztán hozzáadjuk a 2. szeletet, aztán kiösszegezzük az 1. szeletet és így tovább. Ezzel a módszerrel elérhető az állandó idő- és tárigény a szűrési frissítésnél. (Ugyanez a teljesítmény elérhető az egyesítési fa algoritmusának megfelelő módosításaival.) A 15.10. feladat kéri ennek a tények az ellenőrzését az esernyős hálóra.

Eddig tartottak a jó hírek, most következnek a rosszak: megmutatható, hogy egy frissítés idő- és tárkomplexitásának az „állandója” majdnem minden esetben exponenciális az állapotváltozók számában. Az történik ugyanis, hogy amint a változó eliminálás halad, a tényezők megnőnek, magukban foglalva az összes állapotváltozót (vagy pontosabban azokat az állapotváltozókat, amelyeknek van szüleiük az előző időszeletben). A maximális tényezőmérét  $O(d^{n+1})$ , a frissítés költsége pedig  $O(d^{n+2})$ .

 Természetesen ez sokkal kevesebb, mint az RMM frissítésének költsége, ami  $O(d^{2n})$ , de nagyszámú változónál még mindig kivitelezhetetlenül nagy. Ezt a súlyos tényt nehéz elfogadni. Ugyanis a következőt jelenti: *annak ellenére, hogy a DBH-k felhasználhatók olyan nagyon komplex időbeli folyamatok reprezentálására, amelyek számos, ritkán összekapcsolt változóval rendelkeznek, nem vagyunk képesek hatékonyan és pontosan következetni ezekről a folyamatokról*. A DBH-modell, ami az összes változó feletti a priori együttes eloszlást reprezentálja, faktorizálható az öt alkotó FVT-kre, de az a posteriori együttes eloszlás a megfigyelési szekvencián vett feltétellel – azaz az előre üzenet – általában már *nem* faktorizálható. Eddig senki sem talált megoldást erre a problémára, annak ellenére, hogy megoldást a tudomány és mérnöki tervezés fontos területei tudnák nagyon jól hasznosítani. Így közelítő módszerekre kell szorítkoznunk.

## Közeliítő következtetés DBH-kban

A 14. fejezet két következtető módszert írt le: a valószínűségi súlyozást (lásd 14.14. ábra) és a Markov lánc Monte Carlo módszert (MCMC, lásd 14.15. ábra). A kettő közül az első igazítható legkönnyebben a DBH-keretbe. Látni fogjuk azonban, hogy egy gyakorlati módszer kialakulásához a standard valószínűségi súlyozó algoritmuson számos javításra van szükség.

Emlékezzünk vissza, hogy a valószínűségi súlyozás a működése során a nem bizonyítékváltozókat mintavételezi azok topológiai sorrendjében, minden mintát azzal a valószínűséggel súlyozva, amennyire egyezik a megfigyelő bizonyítékváltozókkal. Ahogyan az egzakt közelítő algoritmusok esetében is, a valószínűségi súlyozást közvetlenül alkalmazhatnánk egy kibontott DBH-ra, de ugyanazok a problémák jelentkeznének, nevezetesen, hogy a megfigyeléssorozat növekedésével a frissítésekénti idő- és tárigény növekedne. A probléma az, hogy az alapalgoritmus az egyes mintákat egymás után sorban végigfuttatja a háló teljes hosszában. Ehelyett egyszerűen az összes  $N$  mintát együtt futathatnánk a DBH-n keresztül, szeletenként egyszerre haladva. (Azaz a mintavételeket az újragenerálás költségének elkerülése miatt nyilvántartanánk és fokozatosan frissitenénk – *a ford.*) A módosított algoritmus illeszkedik a szűrési algoritmus általá-

nos mintájához, az  $N$  mintával mint előrefelé üzenettel. Az első kulcsgondolat az, hogy ekkor a mintákat magukat használjuk az állapoteloszlás egy közelítő reprezentációjaként. Ez eleget tesz a frissítésenkénti „állandó” időigénynek, bár az állandó értéke a minták számától függ, aminek elég nagynak kell lenni a valódi a posteriori eloszlás elfogadható közelítéséhez. Szintén nincs szükség a DBH kibontására, mivel csak az aktuális és a következő szeletet kell memoriában tartani.

A valószínűségi súlyozás tárgyalásakor a 14. fejezetben rámutattunk, hogy az algoritmus pontossága leromlik, ha a bizonyítékváltozók „lentebbek” a mintavételezett változóknál, mivel ekkor a minták generálására nincsen hatással a bizonyíték. Egy DBH tipikus struktúráját megnézve – mondjuk az esernyős DBH-t a 15.14. ábrán – láthatjuk, hogy a korai állapotváltozók mintavételezésénél valóban nem hasznosulnak a későbbi bizonyítékok. Valójában, még tüzetesebben megnézve azt láthatjuk, hogy az állapotváltozók egyikének sincs egyetlen bizonyítékváltozó sem az ősei között! Így, bár az egyes minták súlya függeni fog a bizonyítékoktól, a generált minták aktuális halmaza teljesen független lesz a bizonyítékoktól. Például még ha a fönök minden nap el is hozza az esernyőjét, a mintavételei folyamat ekkor is hallucinálhat véget nem érő kánikulát. A gyakorlatban ez azt jelenti, hogy a minták azon aránya, ami elfogadhatóan közel marad az esernények valódi sorához, exponenciálisan csökken  $t$ -vel, a megfigyelés sorozat hosszával; máshogy fogalmazva egy adott pontossági szint fenntartásához, a minták számát exponenciálisan növelni kell  $t$  függvényében. Mivel egy valós időben működő szűrő algoritmus csak rögzített számnú mintát tud használni, a gyakorlatban az történik, hogy igen kis számnú frissítési lépés után a hiba határtalanul megnő.

Világos, hogy jobb megoldásra van szükségünk. A második kulcsgondolat az, hogy a minták halmazát az állapottér nagy valószínűségű részeire koncentráljuk. Ez meglehető a megfigyelések szerint nagyon alacsony súlyú minták eldobásával, miközben megsokszorozzuk a nagy súlyúakat. Ezen a módon a minták populációja elfogadhatóan közel marad a valósághoz. Ha úgy gondolunk a mintákra, mint egy eszközre az a posteriori eloszlás modellezéséhez, akkor van értelme több mintát használni az állapottér azon részeiben, ahol az a posteriori valószínűség nagyobb.

Az algoritmusok egy családját, a részecskeszűrést (**particle filtering**) pontosan ennek elvégzésére terveztek. A részecskeszűrés a következőképpen működik: először egy  $N$  mintájú populációt alkotunk a 0. időpontbeli  $P(X_0)$  a priori eloszlás mintavételezéséből. Ezután minden időpontban egy frissítési ciklus ismétlődik:

- minden mintát előreterjesztünk mintavételezve az  $x_{t+1}$  következő állapot értéket feltéve, hogy a minta aktuális értéke  $x_t$  adott, és felhasználva a  $P(X_{t+1}|x_t)$  állapotátmenetmodellt.
- minden mintát súlyozunk azzal a  $P(e_{t+1}|x_{t+1})$  valószínűsséggel, amit az új bizonyítékhöz rendel.
- a populációt újra mintavételezzük, hogy egy  $N$  mintájú új populációt generálunk. minden új mintát a jelenlegi populációból választunk ki; egy konkrét minta kiválasztásának a valószínűsége arányos a súlyával. Az új minták súlyozatlanok.

Az algoritmus részletesen a 15.15. ábrán látható, és működését az esernyős DBH esetén a 15.16. ábra szemlélteti.

Megmutatjuk, hogy ez az algoritmus konzisztens – így helyes valószínűségeket ad, ahogy  $N$  a végtelenbe tart – meggondolva, hogy mi történik egy frissítési ciklus alatt.

**function** RÉSZECESKE-SZÜRÉS (e, N, dbh) **returns** egy mintahalmaz a következő időpontra

**inputs:** e, az új beérkező bizonyíték

N: a fenntartani kívánt mintaszám

dbh: egy DBH egy  $P(X_0)$  a priori eloszlással,  $P(X_1|X_0)$  állapotátmenet-modellel és

$P(E_i|X_1)$  érzékelő modellel

**static:** S, a minták egy N méretű vektora, kezdetben  $P(X_0)$ -ból generált

**local variables:** W, a súlyok egy N méretű vektora

**for**  $i = 1$  to  $N$  **do**

$S[i] \leftarrow$  sorsolt minta  $P(X_1|X_0 = S[i])$ -ből

$W[i] \leftarrow P(e_i|X_1 = S[i])$

$S \leftarrow$  SÚLYOZOTT-MINTA-VISSZATÉTELLEL(N, S, W)

**return** S

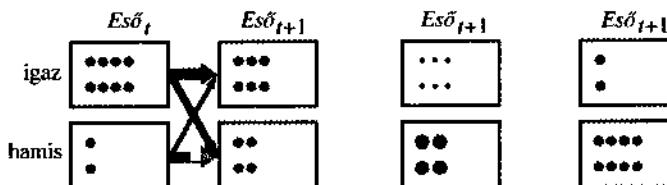
**15.15. ábra.** A részecskeszűrés algoritmus megvalósítása egy rekurzív frissítési művelettel és állapottal (ami a minták halmaza). A mintavételező lépések mindegyike az aktuális szelét változónak a mintavételezést jelenti egy topológiai sorrendben, hasonlóan, mint a PRIOR-MINTA-ban. A SÚLYOZOTT-MINTA-VISSZATÉTELLEL művelet megvalósítható  $O(N)$  várható futási idővel.

Feltesszük, hogy a mintapopuláció a  $t$  időpontbeli  $\mathbf{f}_{1:t}$  előre üzenet egy helyes reprezentációjaként indul. Az  $N(\mathbf{x}_t|\mathbf{e}_{1:t})$  jelölést használva azon minták számára, amelyek az  $\mathbf{e}_{1:t}$  megfigyelések feldolgozása után az  $\mathbf{x}_t$  állapotban vannak, azt kapjuk, hogy

$$N(\mathbf{x}_t|\mathbf{e}_{1:t})/N = P(\mathbf{x}_t|\mathbf{e}_{1:t}) \quad (15.21)$$

nagy N-ekre. Most minden mintát előreterjesztünk mintavételezve az állapotváltozókat a  $t+1$  időponban az adott  $t$  időponthoz tartozó mintabeli értékek mellett. Az egyes  $\mathbf{x}_t$  állapotokból az  $\mathbf{x}_{t+1}$  állapotot elérő minták száma az átmenet-valószínűség és az  $\mathbf{x}_t$  populáció számának a szorzata; így az  $\mathbf{x}_{t+1}$  állapotot elérő minták teljes száma

$$N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t) N(\mathbf{x}_t|\mathbf{e}_{1:t})$$



(a) Továbbterjesztés

(b) Súlyozás

(c) Mintavétel

**15.16. ábra.** A részecskeszűrés frissítési ciklusa az esernyő DBH-nál  $N = 10$  esetén. bemutatva minden egyes állapot mintapopulációját. (a) A  $t$  időponban 8 minta jelez Eső-t és 2  $\neg$ Eső-t. Mindegyiket továbbterjesztjük a következő állapotátmenet-modell szerinti mintavételezésével. A  $t+1$  időponban 6 minta jelez Eső-t és 4  $\neg$ Eső-t. (b) A  $t+1$  időponban  $\neg$ Esernyő a megfigyelés értéke. Mindegyik mintát súlyozzuk a mintának a megfigyelésre vett feltételes valószínűségével, amit a körök nagysága jelez. (c) Egy új 10-es mintahalmaz generálódott az aktuális halmazból történő súlyozott sorsolással, 2 olyan mintát eredményezve, ami Eső-t jelez, és 8 olyat, ami  $\neg$ Eső-t.

Most minden egyes mintát súlyozunk a  $t + 1$  időpontbeli bizonyíték valószínűségével. Egy  $x_{t+1}$  állapotban lévő minta  $P(e_{t+1}|x_{t+1})$  súlyt kap. Az  $x_{t+1}$  állapotban lévő minták teljes súlya az  $e_{t+1}$  megfigyelése után ezért

$$W(x_{t+1}|e_{1:t+1}) = P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t})$$

Most a mintavételi lépés jön. Mivel minden egyes minta a súlyával arányos valószínűséggel ismétlődik, az újramintavételezés után az  $x_{t+1}$  állapotban lévő minták száma arányos az  $x_{t+1}$  állapotban lévő minták teljes súlyával az újramintavételezés előtt:

$$\begin{aligned} N(x_{t+1}|e_{1:t+1})/N &= \alpha W(x_{t+1}|e_{1:t+1}) \\ &= \alpha P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t}) \\ &= \alpha P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t)N(x_t|e_{1:t}) \\ &= \alpha NP(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t}) \end{aligned} \quad (15.21 \text{ miatt})$$

$$\begin{aligned} &= \alpha' P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t}) \\ &= P(x_{t+1}|e_{1:t+1}) \end{aligned} \quad (15.3 \text{ miatt})$$

Azaz a mintapopuláció egy frissítési ciklus után helyesen reprezentálja a  $t + 1$  időpontbeli előrefelé üzenetet.

A részecskezsűrés így *konziszens*, de *hatékony-e*? A gyakorlat azt mutatja, hogy a válasz igen: a részecskezsűrés, úgy tűnik, az igazi a posteriori eloszlás egy jó közelítést tartja fenn állandó számú mintát használva. Azonban jelenleg nincs elméleti garancia; a részecskezsűrés jelenleg aktívan kutatott terület. Számos javítást és változatot javasoltak, és az alkalmazások köre is egyre bővül. Mivel ez egy mintavételező algoritmus, a részecskezsűrés könnyen felhasználható hibrid és folytonos DBH-kban, lehetővé téve az alkalmazását akár olyan területeken is, mint komplex mintázatok követése videofelvételben (Isard és Blake, 1996) és tőzsdei előrejelzés (de Freitas és társai, 2000).

## 15.6. BESZÉDFELISMERÉS

Ebben az alfejezetben az időbeli valószínűségi modellek egyik legfontosabb alkalmazását, a **beszédfelismerést** (*speech recognition*) tekintjük át. A feladat egy beszélő által elmondott szósorozat azonosítása az adott akusztikus jelből. A beszéd az emberek közötti kommunikációnak a domináns modalitása, és a megbízható számítógépes beszédfelismerés felmérhetetlenül fontos lenne. Még ennél is hasznosabb volna a **beszédmegértés** (*speech understanding*) – az elhangzott beszéd *jelentésének* a meghatározása. Ennek tárgyalására a 22. fejezetig kell várunk.

A beszéd jelenti az első találkozásunkat a valódi érzékelők szolgáltatta adatok nyers, tisztítatlan világával. Ezek az adatok zajosak, a szó szoros értelmében: a *zaj* lehet háttérzaj és lehet a digitalizálási folyamat okozta melléktermék; eltérések lehetnek a szavak kiejtési módjában még ugyanannál a beszélőnél is; különböző szavak hangzása ugyanaz lehet és így tovább. Ezen okok miatt a beszédfelismerésre mint valószínűségi következetettségi problémára kezdtek el tekinteni.

A legáltalánosabb szinten a valószínűségi következtetési problémát a következőképpen definiálhatjuk. Legyen a *Szavak* egy valószínűségi változó az összes lehetséges szószekvencia felett, ami elhangozhat, és legyen a *jel* a megfigyelt akusztikusjel-szekvencia. Ekkor az elhangzottak legvalószínűbb értelmezése a *Szavak* azon értéke, ami maximalizálja a  $P(\text{szavak}|jel)$  értéket. Amint az gyakori eset, a Bayes-szabály alkalmazása hasznos:

$$P(\text{szavak}|jel) = \alpha P(jel|\text{szavak}) P(\text{szavak})$$

A  $P(jel|\text{szavak})$  alkotják az ún. **akusztikai modellt (acoustic model)**. Ez írja le a szavak hangzását – a „ceiling (mennyezet)” egy lágy „c”-vel kezdődik és ugyanúgy hangszik, mint a „sealing (pecsételés/tömítés/fókavadászat)”. (Az azonos hangzású szavakat **homofon (homophone)** szavaknak nevezzük.) A  $P(\text{szavak})$  értékei képezik az ún. **nyelvi modellt (language model)**. Ez adja meg az a priori valószínűségét minden egyes kijelentésnek – például hogy a „magas mennyezet (high ceiling)” sokkal valószínűbb szókapesolat, mint a „magas pecsételés (high sealing)”.

A beszédfelismerő rendszerekben használt nyelvi modellek gyakran igen egyszerűek. A fejezetben később leírt **bigram (bigram) modell** minden lehetséges esetre megadja annak a valószínűségét, hogy egy szó egy másik szót követ. Az akusztikai modell sokkal bonyolultabb. Az alapjait egy fontos felfedezés alkotja a **fonológia (phonology)** (a nyelvi hangok tanulmányozása) területéről, nevezetesen, hogy az emberiség minden nyelve egy 40–50 hangból álló korlátos választékkal él, amelyeket **beszédhangoknak (phones)** hívunk. Egy beszédhang nagyjából az a hang, amely egy egyedi mással- vagy magánhangzónak felel meg. A helyzetet komplikálja valamelyest, hogy az olyan betükombinációk, mint „th” vagy „ng” egyedi beszédhangot eredményeznek, viszont egyes betűk többséfele beszédhanghoz vezetnek, a környezetükktől függően (például az „a” betű a „rat” és a „rate” szóban). Az angol nyelv összes beszédhangja, példákkal együtt, a 15.17. ábrán látható. A **fonéma (phoneme)** a legkisebb hangi egység, ami önálló jelentéssel bír egy adott nyelv használi számára. Például az angolban a „t” beszédhang a „stick” szóban ugyanaz a fonéma, mint a „t” beszédhang a „tick” szóban, de a thai nyelvben ezek megkülönböztethetők, mint két különböző fonéma.

A beszédhangok létezése lehetővé teszi az akusztikai modell két részre bontását. Az első rész a **kiejtést (pronunciation)** kezeli, és minden egyes szóra megad egy valószínűség-eloszlást az összes lehetséges beszédhangsor felett. Például a „ceiling” szó kiejtése [s iy l ih ng], vagy néha [s iy l ix ng], vagy esetleg [s iy l en]. A beszédhangok közvetlenül nem megfigyelhetők, így nagyjából a beszéd egy olyan rejtett Markov-modellel reprezentálható, aminek  $X_t$  állapotváltozója a  $t$  időpontban kimondott beszédhang.

Az akusztikai modell második része a beszédhangok akusztikus jelként való megvalósulási módjával foglalkozik: azaz a rejtett Markov-modell  $E_t$ , bizonyítékváltozója adja meg az akusztikus jel  $t$  időpontban megfigyelt jellemzőit, és az akusztikus modell adja meg az  $P(E_t|X_t)$  feltételes valószínűséget, ahol  $X_t$  az aktuális beszédhang. A modellnek lehetőséget kell adni eltérésekre a hangmagasságban, a sebességen és a hangerőben, és **jelfeldolgozási (signal processing)** technikákon alapul, hogy olyan jelreprezentációt biztosítson, ami megfelelően robusztus módon kezeli ezeket a fajta eltéréseket.

Magánhangzók		B–N mássalhangzók		P–Z mássalhangzók	
Beszédhang	Példa	Beszédhang	Példa	Beszédhang	Példa
[iy]	beat	[b]	<u>bet</u>	[p]	<u>pet</u>
[ih]	bit	[ch]	<u>Chet</u>	[t]	<u>rat</u>
[ey]	bet	[d]	<u>debt</u>	[s]	<u>set</u>
[ae]	bat	[f]	<u>fat</u>	[sh]	<u>shoe</u>
[ah]	but	[g]	<u>get</u>	[t]	<u>ten</u>
[ao]	bought	[hh]	<u>hat</u>	[th]	<u>thick</u>
[ow]	boat	[hv]	<u>high</u>	[dh]	<u>that</u>
[uh]	book	[jh]	<u>jet</u>	[dx]	<u>butter</u>
[ey]	bait	[k]	<u>kick</u>	[v]	<u>yet</u>
[er]	Bert	[l]	<u>let</u>	[w]	<u>wet</u>
[az]	buy	[el]	<u>bottle</u>	[wh]	<u>which</u>
[oy]	boy	[m]	<u>met</u>	[y]	<u>yet</u>
[axr]	diner	[em]	<u>bottom</u>	[z]	<u>zoo</u>
[aw]	down	[n]	<u>net</u>	[zh]	<u>measure</u>
[ax]	about	[en]	<u>button</u>		
[ix]	roses	[ng]	<u>sing</u>		
[aa]	cot	[eng]	<u>washing</u>	{-}	(Színen)

15.17. ábra. A DARPA fonetikus ábécé, avagy ARPAbet, amely az amerikai angolban használt beszéhangokat sorolja fel. Számos alternatív felfrás létezik, köztük az International Phonetic Alphabet (IPA), mely az összes ismert nyelv beszéhangját tartalmazza.

A fejezet hátralevő része a modelleket és az algoritmusokat írja le lentről felfelé haladva, az akusztikus jelekkel és a beszéhangokkal kezdve, majd az egyedi szavakkal és végül teljes sorozatokkal folytatva. Befejezésül áttekintjük, hogy ezek a modellek hogyan taníthatók, és mennyire működik jól a kapott rendszer.

## Beszédhangok

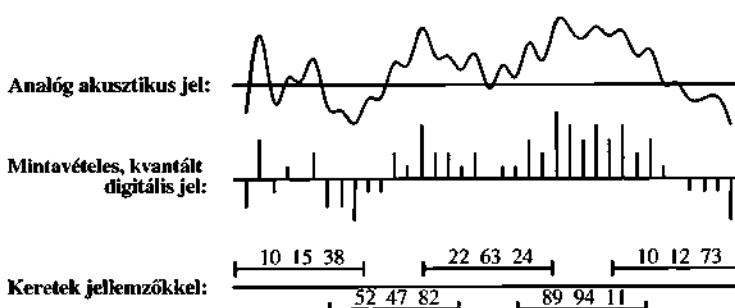
A hanghullámok olyan periodikus nyomásváltozások, amelyek a levegőben terjednek. A hang egy mikrofonnal mérhető, aminek membránját elmozdítja a nyomásváltozás, és egy folytonosan változó áramot generál. Az áramerősséget – ami a hanghullám amplitúdójához tartozik – egy analóg-digitális konverter méri diszkrét időintervallumonként a mintavételi frekvencia (sampling rate) szerint. Beszéd esetén a tipikus mintavételi frekvencia 8 és 16 kHz (azaz 8000-től 16 000-ig másodpercenkként) között van. (Jó minőségű zenei felvételeket mintavételeznek 44 kHz vagy még magasabb frekvenciával.) A mérés pontosságát az egyes mintavételi pontokban a kvantálási tényező (quantization factor) határozza meg; a beszédfelismerők tipikusan 8–12 bitet használnak. Ez azt jelenti, hogy a legkevésbé igényes rendszer, amely 8 kHz-es frekvenciával mintavételez, és 8 bites kvantálást használ, közel fél megabájtot igényel egy egyperces beszéd tárolásához. Ilyen nagy mennyiséggű jelinformáció esetén kivitelezhetetlen létrehozni és használni a  $P(jel|beszédhang)$  eloszlásokat, így az akusztikus jelnek tömörebb leírásait kell kifejlesztenünk.

Elsőként azt vesszük figyelembe, hogy bár a beszédben a hangfrekvencia akár több kHz-es is lehet, a jel tartalmának a megváltozásai sokkal ritkábban fordulnak elő, talán

nem több mint 100 Hz-cel. Ezért a beszédrendszer a jel tulajdonságait hosszabb idő-intervallumonként, úgynevezett keretenként (**frames**) összegzik. Egy 10 ms-os (80 minta 8 kHz-es mintavétel mellett) keret elég rövidnek tűnik ahhoz, hogy csak néhány rövid idejű jelenség mosódjon el az összegzési folyamat miatt. Az egyes kereteken belüli folyamatot egy **jellemző** (**feature**) vektorral ábrázoljuk. Például jellemezni akarjuk az energia szintjét számos frekvenciasáv mindegyikében. Más fontos jellemzők a keret globális energiaszintje és a megelőző kerettől számított (energia)különbség. A jellemzők kiemelése a beszédjelből hasonlít egy kicsit arra, ahogy a zenekar játékát figyelve megjegyezzük: „Most a vadászkürtök hangosan és a hegedűk halkan szólnak.” A 15.18. ábrán láthatjuk az átalakítás lépéseiit a nyers hangtól a keretek sorozatáig. Figyeljük meg, hogy a keretek átlapolódnak; ez megvédi minden attól, hogy pont a keret határán fellépő fontos akusztikus események információját elveszítsük.

A példánkban csupán három jeggyel rendelkező keretet mutattunk. Valós rendszerek jellemzők tucatjaival vagy akár százaival dolgoznak. Például  $n$  darab jellemző esetén, amikor mindegyik 256 lehetséges értéket vehet fel, egy keret egy ponttal adható meg egy  $n$ -dimenziós térben, és  $256^n$  lehetséges keret létezik.  $n > 2$  esetén kivitelezhetetlen volna a  $P(\text{jellemzők}|\text{beszédhang})$  eloszlásnak egy explicit táblázatként történő ábrázolása, így további tömörítésre van szükségünk. Két lehetséges megközelítés létezik:

- A **vektorkvantálás (VK)** (**vector quantization, VQ**) módszere felosztja az  $n$  dimenziós teret mondjuk 256 partícióra, C1-től C256-ig címkézve. Ekkor az egyes kereteket inkább egy egyedi címkével, mint egy  $n$  számból álló vektorral lehet jellemzni. Így a  $P(VQ|\text{beszédhang})$  eloszlás táblázatos formája 256 valószínűséget tartalmaz minden egyes beszédhangra. Nagy rendszerekben a vektorkvantálás már nem népszerű.
- A jellemzők terének diszkretizálása helyett parametrikus folytonos eloszlásokat használhatunk a  $P(\text{jellemzők}|\text{beszédhang})$  leírásához. Például használhatunk minden beszéhanghoz egy Gauss-eloszlást, beszéhangonként különböző átlaggal és kovarianciamátrixszal. Ez jól működik, ha az egyes beszéhangok akusztikus megvalósulása a jellemzők terében egyetlen területen csoportosul. A gyakorlatban egy hanghoz több területen való csoportosulás is tartozik, és **Gauss-eloszlások keverékét (mixture of Gaussians)** kell használni. Egy keverék  $k$  egyedi eloszlás súlyozott összege, így a  $P(\text{jellemzők}|\text{beszédhang})$  eloszláshoz  $k$  súly,  $k$   $n$  elemű



**15.18. ábra.** Az akusztikus jel átalakítása keretek sorozatává; mindegyik keretet három akusztikus jellemzővel írunk le

áttagvektor és  $k n^2$  méretű kovarianciamátrix tartozik, ami  $O(kn^2)$  paraméter minden beszédhangra.

Természetesen bizonyos információ elvész a feldolgozás során, ami a teljes beszédjeltől egy VK címkéig vagy a keverékparaméterek egy halmazáig tart. A jelfeldolgozás művészete abban rejlik, hogy úgy válasszuk meg a jellemzőket és a tartományokat (vagy a Gauss-eloszlásokat), hogy a hasznos információ vesztesége minimális legyen. Egy adott beszédhang számos módon kiejthető: hangosan vagy halkan, gyorsan vagy lassan, magasan vagy mély hangon, csendben vagy háttérzajban és beszélők sok milliója által, mindegyikük különböző kiejtésével és eltérő hangképző szerveivel.<sup>6</sup>

Még két további finomítást kell elvégeznünk az eddig leírt egyszerű modellen. Az előző a beszédhangok időbeli szerkezetével foglalkozik. Normális beszédben a legtöbb beszédhangnak a terjedelme 50–100 ezredmásodperc vagy 5–10 keret. A  $P(\text{jellemzők}|\text{beszédhang})$  valószínűségi modell ugyanaz ezen keretek mindegyikére, pedig a legtöbb beszédhangnak jelentős belső struktúrája van. Például a [t] egyike a zárhangoknak (*stop consonants*), amelyekben a levegő áramlása megszűnik egy rövid időre egy határozott kibocsátás előtt. Az akusztikus jelet megvizsgálva azt látjuk, hogy egy [t]-nek a kezdete hangtalan, közepén egy kis kirobbanással és (általában) egy sziszegéssel a végén. A beszédhangoknak ez a belső struktúrája egy háromállapotú modellel ragadható meg; minden beszédhangnak van egy kezdeti, közép- és végállapota, és minden állapotnak megvan a saját eloszlása a jegyek felett.

A második finomítás azzal a környezettel foglalkozik, amelyben a beszédhang elhangzik. Egy adott beszédhang hangzását megváltoztathatják a környező beszédhangok.<sup>7</sup> Emlékezzünk arra, hogy a beszédhangok az ajkak, a nyelv és az álkapocs mozgása és a levegőnek a hangképző szerveken történő kényszerített átáramlása révén keletkezik. Ezen komplex mozgások koordinálásánál, amik másodpercenként öt vagy még több beszédhangot produkálnak, az agy már akkor elindít a második beszédhanghoz tartozó műveletet, mielőtt az első befejeződött volna, így megváltoztatva az egyik vagy minden két beszédhangot. Például a „sweet (édes)” kiejtésénél az ajkak kerekítétek az [s] előállítása közben számítva a következő [w]-re. Ezeket a koartikulációs hatásokat (*coarticulation effects*) részlegesen megragadja a hármashangzó (*triphone*) modell, amelyben az egyes beszédhangok akusztikai modellje függhet az előző és következő beszédhangtól. Így a [w] a „sweet”-ben úgy írható, hogy [w(s, iy)], azaz [w] egy [s] bal- és egy [iy] jobb-környezetben.

A háromállapotú és hármashangzó modellek együttes hatása megnöveli az időbeli folyamat lehetséges állapotainak a számát az eredeti beszédhang készletbeli  $n$  beszédhangról  $3n^3$ -ra (az ARPAbet-ben  $n \approx 50$ ). A tapasztalatok azt mutatják, hogy a megnövelt pontosság bőven ellensúlyozza a következtetés és a tanulás megnövekedett költségeit.

<sup>6</sup> A duális probléma – a beszélőazonosítás (*speaker identification*) pontosan ennek fordítottja, a közös elemeket hagyja el, és az egyéni különbségeket tartja meg, majd megpróbálja a különbségeket az egyéni beszédmellekhez illeszteni.

<sup>7</sup> Ebben az értelemben a beszéd „beszédhangmodelljét” inkább hasznos közelítésnek kell tekinteni, és nem megváltoztathatatlan törvénynek.

## Szavak

Minden egyes szóra gondolhatunk úgy, mint ami meghatároz egy külön  $P(X_{1,i}|\text{szó})$  valószínűségi eloszlást, ahol  $X_i$  megadja az  $i$ -edik keretben a beszédhang állapotát. Tipikusan ezt az eloszlást két részre osztjuk. A **kiejtési modell (pronunciation model)** egy eloszlást ad meg a beszéhangsorok felett (figyelmen kívül hagyva az ütemet és a kereteket), majd a **beszédhangmodell (phone model)** leírja, ahogyan a beszéhangok leképződnek a keretek szekvenciájára.

Gondolunk arra a szóra, hogy „tomato (paradicsom)”. Gershwin szerint ezt úgy ejtik ki, hogy [t oh m ey t oh] (Gershwin, 1937), én viszont úgy ejtem, hogy [t oh m aa t oh]. A 15.19. ábra felső részén látható egy olyan állapotátmenet-modell, ami gondoskodik erről a változatról. A modellen keresztül csak két út létezik, egy, ami a [t oh m ey t oh] beszédhang szekvenciához tartozik, és a másik a [t oh m aa t oh] szekvenciához. Egy út valószínűsége az útvonalat alkotó nyilakon szereplő valószínűségek szorzata:

$$P([\text{townmeytow}]|, \text{„tomato"}) = P([\text{townmaatow}]|, \text{„tomato"}) = 0,5$$

A fonetikai változékonysság második forrása a **koartikuláció (coarticulation)**. Például a [t] beszéhangnál a nyelv a szájpadlásnál helyezkedik el, az [ow]-nál viszont a száj alján. Gyors beszédnél a nyelv sokszor közbülső pozícióba kerül, melynek eredménye inkább a [t ah], és nem a [t oh]. A 15.19. ábra alsó része a „tomato” kiejtésének egy bonyolultabb modelljét mutatja, amely ezt a koartikulációs hatást figyelembe veszi. Ebben a modellben négy különböző út van, és azt kapjuk, hogy:

$$P([\text{townmeytow}]|, \text{„tomato"}) = P([\text{townmaatow}]|, \text{„tomato"}) = 0,1$$

$$P([\text{tahmeytow}]|, \text{„tomato"}) = P([\text{tahmaatow}]|, \text{„tomato"}) = 0,4$$

Hasonló modelleket lehetne konstruálni minden felismerni kívánt szóhoz.

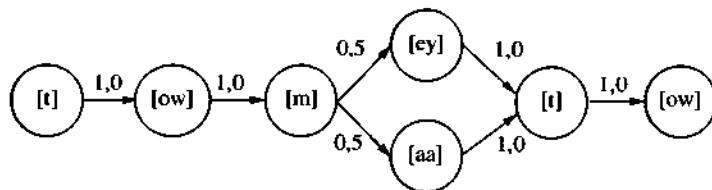
A háromállapotú beszéhangmodell állapotátmenet-diagramja a 15.20. ábrán látható. A modell egy konkrét beszéhanghoz tartozik, az [m]-hez, de az összes beszéhangnak hasonló topológiájú modellje van. Az ábrán láthatók az egyes beszédhang állapotokhoz kapcsolódó akusztikai modellek is, feltételezve, hogy a jelet egy VK címke reprezentálja. Például a modell szerint  $P(E_t = C_1 | X_t = [\text{m}]_{\text{Kezdet}}) = 0,5$ . Vegyük észre a hurkokat az ábrán; például az  $[\text{m}]_{\text{Közép}}$  állapot 0,9 valószínűséggel fennmarad, ami azt jelenti, hogy az  $[\text{m}]_{\text{Közép}}$  állapot várható időtartama 10 keret. A modellünkben az egyes beszéhangok hossza független a többi beszéhang hosszától; egy kifinomultabb modell képes lenne a gyors és lassú beszédet megkülönböztetni.

Hasonló modelleket hozhatunk létre minden egyes beszéhangra, akár a hármashangzó környezet hatását is figyelembe véve. minden szómodell, amikor a beszéhangmodellekkel kombináljuk, egy RMM teljes specifikációját adja. A modell megadja a beszédhang állapotok közötti, keretről keretre történő átmeneteknek a valószínűségeit csakúgy, mint az akusztikus jegyek valószínűségeit minden egyes beszéhang állapothoz.

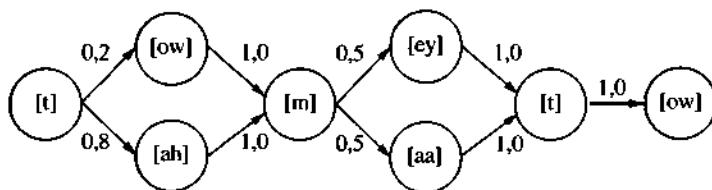
Ha **egyedülálló szavakat (isolated words)** szeretnénk felismerni – azaz egyértelmű határokkal rendelkező és mindenmű környezeti összefüggés nélkül kiejtett szavakat –, akkor azt a szót kell megkeresnünk, amelyik maximalizálja azt, hogy

$$P(\text{szó}|e_{1:t}) = \alpha P(e_{1:t}|\text{szó})P(\text{szó})$$

(a) Nyelvjárást különbséget tekintetbe vevő modell:

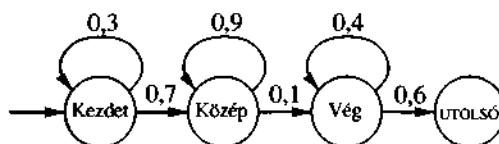


(b) Koartikulációs és nyelvjárást különbségeket tekintetbe vevő modell:



**15.19. ábra.** A „tomato” szó két kiejtési modellje. Mindegyik modellt egy átmenetdiagramként ábrázoljuk, amiben az állapotokat körrel jelöljük, a megengedett átmeneteket pedig nyílakkal, rajtuk a kapcsolódó valószínűségekkel. (a) Egy nyelvjárást különbségeket is tekintetbe vevő modell. A 0,5-ös értékek a két szerző preferált kiejtésein alapuló becslések. (b) Egy olyan modell, ami az első magánhangzón egy koartikulációs hatást is figyelembe vesz, megengedve az [ow] vagy az [ah] beszédhangokat.

Beszédhang RMM [m]-re:



A kimenetelek valószínűségi a beszédhang RMM-ben:

Kezdet:	Közép:	Vég:
$C_1: 0,5$	$C_3: 0,2$	$C_4: 0,1$
$C_2: 0,2$	$C_4: 0,7$	$C_6: 0,5$
$C_3: 0,3$	$C_5: 0,1$	$C_7: 0,4$

**15.20. ábra.** Az [m] háromállapotú beszédhang egy RMM-je. Mindegyik állapotnak számos lehetséges kimenetele lehet, különálló valószínűségekkel. A  $C_1, \dots, C_7$  VK-címek önkényesen lettek megválasztva.

A  $P(\text{szó})$  a priori valószínűség valódi szöveges adatból kapható meg. A  $P(e_{1:t} | \text{szó})$  pedig az akusztikus jegyek sorozatának a valószínűsége a szómodell szerint. Ilyen valószínűségek kiszámítását a 15.2. alfejezet tárgyalta; nevezetesen a (15.15) egyenlet egy egyszerű rekurzív számítást definiál, lineáris költséggel  $t$ -ben és a Markov-lánc

állapotainak számában. A legvalószínűbb szó megtalálásánál ezt a számítást minden lehetséges szóra elvégezhetjük, megsorozzuk az a priori valószínűsséggel, és e szerint választjuk ki a legjobb szót.

## Mondatok

Az emberi kommunikációban való részvételhez egy gépnek a **folytonos beszédet** (**continuous speech**) kell felismernie, és nem pusztán egyedülálló szavakat. Azt gondolhatnánk, hogy a folytonos beszéd nem több, mint szavak sorozata, amelyek mindegyikére alkalmazhatjuk az algoritmust az előző fejezetből. Ez a megközelítés két ok miatt is bukásra van ítéltve. Először is, már láttuk (a 638. oldalon), hogy a legvalószínűbb szavak sorozata nem a legvalószínűbb együttes szósorozat. Például a *Take the Money and Run (Fogd a pénzt és fuss)* c. filmben a banki alkalmazott félreérte Woody Allen irkafirka bankrablási üzenetét, és azt „I have a gun (Nálam van a slukker)”-nak olvassa (az „I have a gun, vagyis „Nálam van a stukker” helyett – *a ford.*). Egy jobb nyelvi modell az „I have a gun”-t javasolná, mint aminek sokkal nagyobb a valószínűsége, még ha a legutolsó szó inkább „gub”-nak és nem „gun”-nak olvasható. A második probléma, amivel folytonos beszéd esetén szembesülnünk kell, a **szegmentálás (segmentation)** – annak az eldöntése, hogy hol van egy szónak a vége, és hol kezdődik a következő. mindenki, aki megkísérelt egy idegen nyelvet megtanulni, méltányolni fogja a probléma nehézségét: először úgy tűnik, hogy a beszédben a szavak egybefolynak, majd fokozatosan tanulunk meg egyedi szavakat kiemelni a hangok egyvelegéből. Ebben az esetben az első benyomás korrekt: a spektrográfiai elemzés azt mutatja, hogy a folyamatos beszédben a szavak tényleg szünetek nélkül követik egymást. A szavak határait annak ellenére tanuljuk meg megkülönböztetni, hogy a szavakat nem választják szét csendszakaszok.

Kezdjük a nyelvi modellel, aminek a beszédfelismerésben az a feladata, hogy valószínűséget rendeljen minden egyes szósorozathoz. Legyen  $w_1 \dots w_n$  az  $n$  szóból álló szósorozat, és legyen  $w_i$  a szósorozat  $i$ -edik szava, akkor a szósorozat-valószínűséget az alábbi módon íthatjuk:<sup>8</sup>

$$\begin{aligned} P(w_1 \dots w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1\dots w_{n-1}) = \\ &= \prod_{i=1}^n P(w_i|w_1\dots w_{i-1}) \end{aligned}$$

Ezen kifejezések többsége igen bonyolult és nehezen becsülhető vagy számítható. Szerencsére a képletet egyszerűbb kifejezéssel is közelíthetjük, miközben a nyelvi modell zömét mégis megragadjuk. Az egyik egyszerű, köz kedvelt és hatékony eljárás a **bigram (bigram)** modell. Ez a modell a  $P(w_i|w_1\dots w_{i-1})$ -et a  $P(w_i|w_{i-1})$ -gyel közelíti. Más szóval, feltételezi, hogy a szósorozatokra teljesül az elsőrendű Markov-feltétel.

A bigram modell nagy előnye, hogy a modell egyszerűen tanítható az egyes szópárok megszámolásával szósorozatok egy reprezentatív korpuszában és a valószínűségek ezen számok alapján történő megbecslésével. Így például, ha az „a” a mintakorpuszban

<sup>8</sup> Valóban jobb lenne, ha minden valószínűséget egy adott helyzetre vonatkozó feltétel mellett lehetne figyelembe venni. Kevés beszédfelismerő rendszer él ezzel a lehetőséggel, mert nehéz formálisan meghatározni, hogy mi számít adott helyzetnek.

10 000-szer fordul elő, és a „gun” azt 37-szer követi, akkor  $\hat{P}(\text{gun}_i | a_{i-1}) = 37/10\,000$ , ahol a  $\hat{P}$  a becsült valószínűség. Az ilyen tanítás befejeztével elvárnánk, hogy az „I have” és az „a gun” valószínűsége relatíve magas legyen, az „I has” és az „an gun” valószínűsége viszont alacsony. A 15.21. ábra a jelen könyv (eredeti angol nyelvű kiadásának) szavaiból számított néhány bigram gyakoriságát mutatja.

Szó	Unigram gyakoriság	Előző szó							
		of	in	is	on	to	from	model	agent
the	33508	3833	2479	832	944	1365	597	28	24
on	2573	1	0	33	2	1	0	0	6
of	15474	0	0	29	1	0	0	88	7
to	11527	0	4	450	21	4	16	9	82
is	10566	3	6	1	4	2	1	47	127
model	752	8	1	0	1	14	0	6	4
agent	2100	10	3	3	2	3	0	0	36
idea	241	0	0	0	0	0	0	0	0

**15.21. ábra.** Az unigram és bigram számlálóknak egy részleges táblázata a jelen könyv (eredeti angol nyelvű kiadásának) szavai alapján. A „the” a leggyakoribb egyedülálló szó 33 508 előfordulással (513 893 összes szóból). A leggyakoribb bigram az „of the”, 3833 előfordulással. Néhány szám nagyobb, mint várható (például 4-szer fordul elő az „on is”), mivel a bigram számlálás figyelem kívül hagyja az elválasztást: egy mondat végződhet „on”-ra és a következő kezdődhet „is”-zel.

A modellt ki lehet bővíteni trigrammá (trigram), amely a  $P(w_i | w_{i-1} w_{i-2})$  mennyiségeket használja. Ez egy hatékonyabb nyelvi modell, képes megadni, hogy az „ate a banana (banánt ettem)” valószínűbb, mint az „ate a bandana (kendőt ettem)”. A trigram modellben és kisebb mértékben a bigram és unigram modellekben is problémát jelentenek a nulla értékű számlálók: nem szeretnénk egy szókombinációt lehetetlennek minősíteni pusztán azért, mert nem fordulnak elő a tanító korpuszban. A simítás (smoothing) folyamata ilyen kombinációkhöz egy kis, pozitív valószínűséget rendel. Ezt a 952. oldalon tárgyaljuk.

A bigram és trigram modellek nem annyira kifinomultak, mint a 22. és 23. fejezetben szereplő nyelvi modellek némielyike, de jobban tekintetbe veszik a helyi összefüggések hatásait, és bizonyos helyi szintaxist sikeresen kifejeznek. Például az a tény, hogy az „I has” és a „man have” szópárok alacsony pontszámokat kapnak, az alany-állítmány egyeztetést fejezi ki. A probléma az, hogy ezeket a kapcsolatokat csak helyileg lehet észlelni: „the man have” alacsony pontszámot kap, de a „the man with yellow hat have” nincs büntetve.

Most gondoljuk meg a nyelvi modell és a szómodellek egyesítését, hogy minden szó-sorozatot megfelelően tudjunk kezelni. Az egyszerűség kedvéért egy bigram nyelvi modellt fogunk feltételezni. Egy ilyen modellel az összes szómodell (ami viszont kiejtési és beszédhangmodellekből áll) egyetlen nagy RMM-modellbe állítható össze. Az egyszavas RMM-modell egy állapota egy keret, felcímkézve az aktuális beszédhanggal és beszédhangállappal (például [m]Kezdet); egy folytonos beszédű RMM-ben egy állapot a szóval is fel van címkézve, mint például [m]<sup>tomato</sup>Kezdet. Ha minden szónak átlagban  $p$  háromállapotú beszédhang szerepel a kiejtési modelljében, akkor  $W$  szó esetén a folyto-

nos beszédű RMM-nek  $3pW$  állapota van. Átmenetek következhetnek egy adott beszédhangon belül a beszédhang állapotai között, egy adott szó beszédhangjai között, illetve egy szó utolsó állapota és egy másik szó kezdő állapota között. A szavak közti átmenetek a bigram modell szerinti valószínűséggel következnek be.

Ha előállítottuk az egyesített RMM-modellt, akkor a felhasználásával elemezhetjük a folytonos beszédjelet. Különösen a (15.9) egyenletben megadott Viterbi-algoritmusnak vehetjük hasznát a legvalószínűbb állapot sorozat megtalálásában. Ebből az állapot sorozatból már kinyerhető a szósorozat egyszerűen az állapotok szócímkéinek kigyűjtésével. Így a Viterbi-algoritmus a szószegmentálás problémáját úgy oldja meg, hogy dinamikus programozást felhasználva (valójában) egyszerre figyelembe veszi az összes lehetséges szósorozatot és szóhatárt.

Vegyük észre, hogy nem azt mondunk, hogy „kinyerhető a legvalószínűbb szósorozat”. A legvalószínűbb szósorozat nem szükségképpen az, ami a legvalószínűbb állapot sorozatot tartalmazza. Ez amiatt van így, mert egy szósorozat valószínűsége az összes olyan állapot szekvencia valószínűségének összege, amely konzisztens ezzel a szósorozattal. Két szósorozat összehasonlítva, mondjuk „a back” és „aback” szósorozatokat, előfordulhat, hogy tíz alternatív állapot sorozat van az „a back”-hez, ezek mindegyike 0,03 valószínűséggel, de csak egyetlen állapot sorozat az „aback”-hez, viszont 0,20 valószínűséggel. A Viterbi-algoritmus az „aback”-t választja, pedig az „a back” valójában valószínűbb.

A gyakorlatban ez a nehézség nem életveszélyes, de elég komoly ahhoz, hogy más módszereket is kipróbáljanak. A legáltalánosabb választás az **A\* dekódoló (A\* decoder)**, ami leleményesen használja fel az A\* keresést (lásd 4. fejezet) a legvalószínűbb szósorozat megtalálásában. Az ötlet az, hogy minden szósorozatot egy olyan gráfpontra vezető útnak fogunk fel, aminek a csomópontjai a szavakkal vannak felcímkezve. Egy csomópont gyermekei az összes olyan szó, amelyek utána következhetnek; így az összes  $n$  hosszúságú vagy rövidebb mondathoz tartozó gráfnak  $n$  szintje van, mindegyik  $W$  szélességgű, ahol  $W$  a lehetséges szavak száma. A bigram modellnél a  $w_1$  és  $w_2$  címekű csomópontok közti nyílhoz rendelt  $g(w_1, w_2)$  költséget a  $-\log P(w_2|w_1)$  definíálja. Ekkor egy sorozat teljes útköltsége

$$\text{Költség}(w_1 \dots w_n) = \sum_{i=1}^n -\log P(w_i|w_{i-1}) = -\log \prod_{i=1}^n P(w_i|w_{i-1})$$

Az útköltség ezen definíciója mellett, a legrövidebb út megtalálása pontosan megegyezik a legvalószínűbb szósorozat megtalálásával. A keresés hatékonyságához egy jó  $h(w_i)$  heurisztika is szükséges a szósorozat befejezéséhez szükséges költség megbecslésére. Nyilvánvalóan ennek főként a beszédjelnek ahhoz a részéhez kell kapcsolódnia, amit a jelenlegi úton fekvő szavak még nem fedtek le. Eddig még nem született különösebben érdekes heurisztika erre a problémára.

## Egy beszédfelismerő építése

Egy beszédfelismerő rendszer minősége az összes részének a minőségtől függ – a nyelvi modelltől, a szókiejtési modelltől, a beszédhangmodelltől és a jelfeldolgozó algoritmusoktól, amelyek a spektrális jegyeket kinyerik az akusztikus jelből. Megtárgyalunk, hogyan hozhatunk létre egy nyelvi modellt, és a jelfeldolgozás részletei más könyvekben is meg-

találhatók. Így már csak a kiejtési és a beszédhangmodellek maradnak. A kiejtési modellek struktúrája – mint a paradicsom modelljei a 15.19. ábrán – általában kézi fejlesztésű. Nagy fonetikai szótárak most már elérhetők angol és más nyelvekhez, bár a pontosságuk nagymértékben változik. A háromállapotú beszédhangmodell struktúrája minden beszédhangra ugyanaz, mint a 15.20. ábrán látható. Így már csak a valószínűségek maradnak. Honnan szerezhetők be ezek, különösen hogy a modellek paraméterek százreit vagy akár millióit is igényelhetik?

Az egyetlen elfogadható módszer a modellek tanulása valódi beszédadatból, amiből bizonyosan nincsen hiány. A következő kérdés a tanulás mikéntje. A teljes választ a 20. fejezetben adjuk meg, de a fő elképzeléseket itt is bemutathatjuk. Gondoljuk át a bigram nyelvi modellt; ennek megtanulását már elmagyaráztuk, ami a szópárok gyakoriságának a vizsgálatán alapult valódi szövegben. Megtehető ugyanez mondjuk a beszéhangok átmenet-valószínűségeire is a kiejtési modellben? A válasz igen, de csak akkor, ha valaki veszi a fáradságot, és végigjegyzeteli az összes előforduló szót a helyes beszéhangsorozattal. Ez egy bonyolult és számos hibalehetőséget magában rejtő feladat, de néhány standard adathalmazra elvégezték, amelyek több órás beszédet tartalmaznak. Ha ismerjük a beszéhang-székvinciát, akkor a kiejtési modell átmenet-valószínűségeit megbecsülhetjük a beszéhangpárokból. Hasonlóan, ha minden keretre ismert a beszéhang állapot – egy még gyötrelmesebb kézi felcímkézés eredményeképpen – akkor megbecsülhetjük a beszéhangmodell átmenet-valószínűségeit is. Ha minden keretben ismerjük a beszéhangállapotot és az akusztikus jegyeket, akkor pedig megbecsülhetjük az akusztikai modellt, akár közvetlenül a gyakoriságokból (a VK modellek esetében) vagy statisztikai illesztési módszerekkel (a Gauss-modellek keveréke esetében; lásd 20. fejezet).

A kézi címkézésű adatnak a költsége és a ritkasága, illetve az a tény, hogy az elérhető kézi címkézésű adathalmazok nem feltétlenül reprezentálják a beszélők és az akusztikus feltételek azon típusait, amelyek egy új beszédfelismerési feladatban fellépnek, ezt a megközelítést bukásra ítélik. *Szerencsére a várhatóérték-maximalizálás vagy EM-algoritmus (expectation-maximization, EM) anélküli is megtanulja az RMM átmenet-valószínűségeit és érzékelő modelljeit, hogy címkézett adatot használna.* Kézi címkézésű adatokból származó becsléseket felhasználhatunk a modell kezdeti beállítására; ezután az EM következik, és betanítja a modellt az éppen aktuális feladatara. Az ötlet egyszerű: egy adott RMM és egy megfigyeléssorozat esetén, felhasználhatjuk a 15.2. és a 15.3. alfejezetben részletezett simító algoritmusait az egyes állapot-valószínűségek kiszámítására minden időpontban, illetve egy egyszerű kiterjesztéssel az állapot-állapot párok valószínűségét is az egymást követő időpontokban. Ezeket a valószínűségeket felfoghatjuk bizonytalan címkéknek. A bizonytalan címkékből megbecsülhetünk új átmenet- és érzékelési valószínűségeket, majd az EM eljárás megismétlődik. A módszer minden iterációban bizonyítottan növeli a modell adathoz való illeszkedését, és általában a kezdetben beállított kézi címkézésű becslések értékeinél, a paraméterétek egy sokkal jobb halmozához konvergál.

A legkorszerűbb beszédfelismerő rendszerek óriási adathalmazokat és hatalmas számítási erőforrásokat használnak fel a modelljeik tanításához. Egyedülálló szófelismerésnél, jó akusztikus feltételekkel (háttérzaj és visszhang nélkül), egy néhány ezres szótárnál és egy beszélő esetén, a pontosság 99% feletti lehet. Folytonos beszédfelismerésnél és több különböző beszélő esetén, a 60–80%-os pontosság az általános, még jó akusztikus körümények esetén is. Háttérzajnál és telefonos átvitelnél a pontosság tovább romlik. Bár az üzembe helyezett rendszerek évtizedek óta javulnak, számos ötlet még kidolgozásra vár.

## 15.7. ÖSSZEFOGLALÁS

Ez a fejezet az ábrázolás és a következtetés általános problémáját tárgyalta időbeli valószínűségi folyamatok esetében. A legfőbb pontok a következők:

- A világ változó állapotát valószínűségi változók halmazának a felhasználásával kezeljük, amelyek az állapotot ábrázolják minden időpontban.
- Kidolgozhatók olyan reprezentációk, amelyek eleget tesznek a Markov-tulajdonság-nak (**Markov property**), nevezetesen, hogy a jövő független a múlttól a jelen ismertetében. Ez együttesen a folyamat stacionaritásának (**stationary**) feltevéssel – azaz azazal, hogy a dinamika az idővel nem változik – nagyban egyszerűsíti a reprezentációt.
- Egy időbeli valószínűségi modellt felfoghatunk egy állapotátmenet-modell (**transition model**) és egy érzékelő modell (**sensor model**) együttesének, ahol az első az állapot alakulását, a második pedig a megfigyelés folyamatát írja le.
- Időbeli modellekben az alapvető következtetési feladatok a szűrés (**filtering**), az előrejelzés (**prediction**), a simítás (**smoothing**) és a legvalószínűbb magyarázat kiszámítása (**most likely explanation**). Ezek mindegyike elvégzhető egyszerű, rekurzív algoritmusokkal, melyek futási ideje a szekvencia hosszában lineáris.
- Időbeli modellek három családját tanulmányoztuk részletesebben: a rejtett Markov-modellek (**hidden Markov model**), a Kalman-szűrőket (**Kalman filters**), és a dinamikus Bayes-hálókat (**dynamic Bayesian networks**) (ami magában foglalja a másik kettőt, mint speciális esetet).
- A beszédfelismerés (**speech recognition**) és a követés (**tracking**) az időbeli valószínűségi modellek két fontos alkalmazási területe.
- Hacsak nincsenek speciális feltételezések, mint a Kalman-szűrőknél, egzakt következtetés több állapotváltozó esetén nem kivitelezhető. A gyakorlatban a részecske-szűrő (**particle filtering**) algoritmus tűnik egy hatékony közelítő algoritmusnak.

## Irodalmi és történeti megjegyzések

Számos alapötlet dinamikus rendszerek állapotának a becslésére a matematikus C. F. Gausstól származik, aki egy determinisztikus legkisebb-négyzeteken alapuló algoritmust alkotott meg az égitestek pályájának a megbecsülésére csillagászati megfigyelések-ból (Gauss, 1809). Az orosz matematikus A. A. Markov fejlesztette ki a később róla elnevezett Markov-feltételezést a sztochasztikus folyamatokat vizsgálva; az elsőrendű Markov-láncot maga Markov alkalmazta először a *Jevgenij Anyegin* szövegének betűszekvencia-elemzésénél (Markov, 1913). Jelentős titkosított munka folyt a szűrés problémáján a második világháború alatt folytonos idejű folyamatok esetében (Wiener, 1942) és diszkrét idejű folyamatoknál is (Kolmogorov, 1941). Bár ez a munka fontos technikai fejlődést eredményezett a következő 20 évben, a felhasznált frekvenciatartománybeli reprezentáció több számítást is elég nehézkessé tett. A sztochasztikus folyamatok közvetlen állapottérbeli modellezése egyszerűbbnek bizonyult, ahogy azt Swerling és Kalman munkái megmutatták (Swerling, 1959; Kalman, 1960). Az utóbbi munka vezette be a most Kalman-szűrésnek nevezett technikát előrefelé következtetés elvégzésére lineáris rendszerekben Gauss-zaj mellett. Rauch és társai fontos eredménye-

ket értek el a simításhoz kapcsolódóan, a megkapóan elnevezett Rauch–Tung–Striebel-simítás még ma is szabványtechnika (Rauch és társai, 1965). Számos korai eredményt gyűjtötték össze a (Gelb, 1974) kiadványban. Bar-Shalom és Fortmann egy modernebb tárgyalást kínál bayesi vonásokkal, miközben számos hivatkozást is ad a hatalmas mennyiséges szakirodalomra (Bar-Shalom és Fortmann, 1988). Chatfield az idősor analízis „klasszikus” megközelítését fedi le (Chatfield, 1989).

A Kalman-szűrés számos alkalmazásában nem csak a bizonytalan érzékeléssel és a dinamikával kell foglalkozni, hanem bizonytalan azonosítással is; azaz ha több objektumot kell megfigyelni, a rendszernek meg kell határozni, hogy melyik megfigyelést melyik objektum generálta, mielőtt az állapotbecsléseket felfrissíthetné. Ez az **adattársítás (data association)** problémája (Bar-Shalom és Fortmann, 1988; Bar-Shalom, 1992). Azon esetben, amikor  $n$  megfigyelés és  $n$  követés van (ami egy igen szelíd eset), a megfigyelések  $n!$  lehetséges módon rendelhetők a követésekhez; a helyes valószínűségi kezelés minden figyelembe venné, ami megmutathatóan NP-teljes (Cox, 1993; Cox és Hingorani, 1994). Az MCMC-alapú polinom idejű közelítő módszerek úgy tűnik, jól működnek a gyakorlatban (Pasula és társai, 1999). Érdekes, hogy az adattársítás problémája egy elsőrendű nyelvben végzett valószínűségi következtetés egy esete; a legtöbb valószínűségi következtetési problémától eltérően, amelyek tisztán ítéletlogikai szinten vannak, az adattársítás objektumokat és azonossági relációt foglal magában. Ezért közvetlenül kapcsolódik a 14. fejezetben említett elsőrendű valószínűségi nyelvekhez. Új kutatások megmutatták, hogy általában az azonosság feletti érvélés és konkrétan az adattársítás elvégezhető az elsőrendű valószínűségi keretben (Pasula és Russell, 2001).

A rejtejt Markov-modellt és a kapcsolódó következtető és tanuló algoritmusokat, beleértve az előre-hátra algoritmust, Baum és Petrie fejlesztették ki (Baum és Petrie, 1966). Ettől függetlenül hasonló ötletek megjelentek a Kalman-szűréssel foglalkozó kutatásnál is (Rauch és társai, 1965). Az előre-hátra algoritmus volt az egyik fő előzménye az EM algoritmus általános megfogalmazásának (Dempster és társai, 1977); lásd még 20. fejezet. Az állandó idejű simítás Bindernek és munkatársainak a munkájában jelenik meg, csakúgy mint az „oszd meg és uralkodj” algoritmus, amit a 15.3. feladatban mutatunk be (Binder és társai, 1997b).

A dinamikus Bayes-hálókra tekinthetünk úgy, mint a Markov-folyamatok egy ritka kódolására, és az MI-n belül elsőként a következő munkákban használták fel: (Dean és Kanazawa, 1989b; Nicholson, 1992; Kjaerulff, 1992). A legutóbbi munkák egyike a HUGIN Bayes-hálós rendszer általános kiterjesztése, ami dinamikus Bayes-hálók generálásához és szerkesztéséhez biztosítja a szükséges eszközöket. A dinamikus Bayes-hálók népszerűvé váltak különböző komplex mozgási folyamatok modellezésére a számítógépes látás területén (Huang és társai, 1994; Intille és Bobick, 1999). A kapcsolatot az RMM-k és a DBH-k között, illetve az előre-hátra algoritmus és a Bayes-hálós terjesztés között Smyth és társai tisztázták (Smyth és társai, 1997). Egy további egyeségesítés a Kalman-szűrőkkel (és más statisztikai modellekkel) Roweis és Ghahramani munkájában jelenik meg (Roweis és Ghahramani, 1999).

A 15.5. fejezetben leírt részecske-szűrő algoritmusnak különösen érdekes története van. Az első szűrésre szolgáló mintavételi algoritmus a szabályozáselmélet területén született (Handschin és Mayne, 1969), és az újramintavételezés ötlete, ami a részecske-szűrés központi eleme egy szabályozástechnikával foglalkozó orosz folyóiratban jelent meg (Zaritskii és társai, 1975). Később többször újra felfedezték: a statisztika területén

mint szekvenciális fontossági mintavételezésű újramintavételezés (SUFM) (*sequential importance-sampling resampling, SIR*) (Rubin, 1988; Liu és Chen, 1998), a szabályozáselmélet területén mint részecskeszűrés (Gordon és társai, 1993; Gordon, 1994), a mesterséges intelligencia területén mint a legjobb túlélése (*survival of the fittest*) (Kanazawa és társai, 1995) és a számítógépes látásban mint sűrítés (*condensation*) (Isard és Blake, 1996). Kanazawa munkája egy javítást is tartalmazott, az úgynevezett **bizonyítekmegfordítást** (*evidence reversal*), ami által a  $t + 1$  időpontban az állapot mintavételezése a  $t$  időpontbeli állapot és a  $t + 1$  időpontbeli bizonyíték feltételek mellett történik (Kanazawa és társai, 1995). Ez lehetővé teszi, hogy a bizonyíték közvetlenül befolyásolja a minták generálását és bizonyítottan csökkenti a közelítési hibát (Doucet, 1997).

A közelítő szűrésre szolgáló módszerek között találjuk a lecsengő MCMC (*decayed MCMC*) algoritmust (Marthi és társai, 2002) és a faktorizált közelítés módszerét Boyen és társaitól (Boyen és társai, 1999). Mindkét módszer rendelkezik azzal a fontos tulajdonsággal, hogy a közelítési hiba nem nő az idővel. Variációs technikákat (lásd 14. fejezet) szintén fejlesztettek ki időbeli modellekre. Ghahramani és Jordan egy közelítő algoritmust ismertet faktoriális RMM-kre, így olyan DBH-ra, amiben két vagy több függetlenül fejlődő Markov-lánc egy közös megfigyelési folyammal van összekapcsolva (Ghahramani és Jordan, 1997). Jordan és társai számos más alkalmazást is tárgyalnak (Jordan és társai, 1998). A keverési idők tulajdonságait Pak, valamint Luby és Vigoda tárgyalja (Pak, 2001; Luby és Vigoda, 1999).

A beszédfelismerés őstörténete az 1920-as években kezdődött Radio Rexsel, a hangvezérlésű játék kutyával. Rex ugrált az 500 Hz körüli hangfrekvenciákra válaszul, ami az [eh] magánhangzóhoz tartozik a „Rex!”-ben. Kissé komolyabb munka a második világháború után kezdődött. Az AT&T Bell Labsnál egy rendszert építettek egyedülálló számjegyek felismerésére akusztikus jellemzők egyszerű mintaillesztésével (Davis és társai, 1952). A beszédhang átmenet-valószínűségeket először egy, a londoni University College-ban épített rendszerben használtak (Fry, 1959; Denes, 1959). 1971-ben az Egyesült Államok Védelmi Minisztériumának Kutatási Ügynöksége (Defense Advanced Research Project Agency, DARPA) négy ötéves kutatási tervet kezdett el finanszírozni, hogy nagy teljesítményű beszédfelismerő rendszereket fejlesszenek ki. A győztes, és egyben az egyetlen rendszer, ami az 1000 szavas szótáron a kitűzött 90%-os pontosságot elérte, a HARPY rendszer volt a CMU-ról (Lowerre, 1976; Lowerre és Reddy, 1980).<sup>9</sup> A HARPY végső változatát egy DRAGON nevű rendszerből származtatták, amit egy CMUs diák, James Baker épített (Baker, 1975). A DRAGON volt az első rendszer, ami RMM-eket használt beszédre. Majdnem egy időben Jelinek az IBM-nél kifejlesztett egy másik RMM-alapú rendszert (Jelinek, 1976). Ettől az időpontról kezdve, a valószínűségi módszerek általában is, de az RMM-ek különösen egyre inkább dominálták a beszédfelismerés kutatását és fejlesztéseit. Az utóbbi éveket a gyarapodó fejlődés, a nagyobb adathalmazok és modellek, és a szigorúbb verseny realisztikusabb beszédhelyzeteken jellemzi.

<sup>9</sup> A versenyben másodikként végző rendszernek, a HEARSAY-II-nek az MI más ágaira volt nagy hatása, mert a táblaarchitektúrát (*blackboard architecture*) használta (Erman és társai, 1980). Ez egy szabályalapú szakértői rendszer volt, több többé-kevésbé független moduláris tudásforrással (*knowledge source*) rendelkezett, amelyek egy közös **tábla** (*blackboard*) révén kommunikáltak egymással, olvashattak a tábláról és írhattak rá. A táblaarchitektúrát használó rendszerek a modern felhasználói felületek architektúráinak az alapjai.

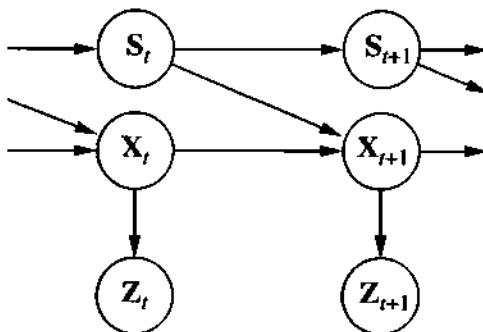
Egyes kutatók megvizsgálták a DBH-k felhasználásának lehetőségét az RMM-ek helyett, azzal a céllal, hogy a DBH-k nagyobb kifejezőerejét kihasználva a beszédképző szervek komplex rejtett állapotából többet tudjanak megragadni (Zweig és Russell, 1998; Richardson és társai, 2000).

A beszédfelismeréshez jó bevezető irodalom (Rabiner és Juang, 1993; Jelinek, 1997; Gold és Morgan, 2000; Huang és társai, 2001). A témahoz tartozó fontos cikkek gyűjteménye (néhány áttekintő cikket is beleértve) (Waibel és Lee, 1990). Jelen fejezet anyagát Kay, Gawron és Norvig áttekintő munkájára (Kay és társai, 1994), valamint Jurafsky és Martin egyik könyvére alapoztuk (Jurafsky és Martin, 2000). A beszédfelismerésről kutatási eredményeket közölnek a *Computer Speech and Language*, a *Speech Communication* és az *IEEE Transactions on Acoustics, Speech, and Signal Processing* folyóiratok, valamint a *DARPA Workshops on Speech and Natural Language Processing* kiadványok, továbbá a Eurospeech, az ICSLP és az ASRU konferenciák kiadványai.

## Feladatok

- 15.1.** Mutassa meg, hogy bármely másodrendű Markov-folyamatot át lehet írni először rendű Markov-folyamatot az állapotváltozóknak egy megnövelt halmazával használva. Megtehető-e ez minden *takarékosan*, azaz az állapotátmenet-modellt megadó paraméterek számának a növelése nélküli?
- 15.2.** Ebben a feladatban azt vizsgáljuk, hogy mi történik a valószínűségekkel az esernyő világban, ha az időszorok hossza a végterenbe tart.
  - (a) Tételezzük fel, hogy a napok olyan végtelen sorát figyeljük meg, amikor az esernyő minden feltűnik. Mutassa meg, hogy a napok műlásával az eső valószínűsége az aktuális napon monoton növekszik egy határértékhez. Számítsa ki ezt a határértéket.
  - (b) Most gondolja át az *előrejelzés-t* az egyre távolabbi jövőben, csupán a két első esernyő megfigyelés ismeretében. Először számolja ki a  $P(R_{2+k}|U_1, U_2)$  valószínűséget  $k = 1 \dots 20$  esetén, és ábrázolja az eredményeket. A valószínűségek láthatóan konvergálnia kell egy határértékhez. Számítsa ki a határérték pontos értékét.
- 15.3.** Ez a feladat a 15.4. ábrán leírt előre-hátra algoritmus egy tárta karékos változatát fejleszti ki. A  $P(X_k|e_{1:t})$  valószínűséget szeretnénk kiszámítani  $k = 1, \dots, t$ -re. Ezt egy „oszd meg és uralkodj” séma szerint végezzük el.
  - (a) Tételezzük fel az egyszerűség kedvéért, hogy  $t$  páratlan, és legyen a felezőpont  $h = (t+1)/2$ . Mutassa meg, hogy  $P(X_k|e_{1:t})$  kiszámítható  $k = 1, \dots, h$  esetén csupán az  $f_{1:0}$  kezdő előrefelé üzenetet, a  $b_{h+1:t}$  visszafelé üzenetet és az  $e_{1:h}$  bizonyítékokat ismerve.
  - (b) Mutassa meg, hogy a sorozat második felére is fennáll egy hasonló eredmény.
  - (c) Az (a) és (b) eredmények ismeretében egy rekurzív „oszd meg és uralkodj” algoritmust lehet létrehozni, először előrefelé haladva a sorozaton és aztán visszafelé a végétől, csak a szükséges üzeneteket tárolva középen és a

- végeken. Aztán az algoritmus lefut minden két félre. Írja le az algoritmust részletesen.
- (d) Számítsa ki az algoritmus tár- és időkomplexitását a sorozat hosszának,  $t$ -nek a függvényében. Hogyan változik ez, ha felosztjuk a bemenetet két részre?
- 15.4.** A 638. oldalon vázoltunk egy hibás eljárást a legvalószínűbb állapotszekvencia megtalálására egy adott megfigyelési szekvencia esetén. Az eljárás azon alapul, hogy megkeresi a legvalószínűbb állapotokat minden egyes időpontra, és visszaad egy ezen állapotokból álló sorozatot. Mutassa meg, hogy bizonyos időbeli valószínűségi modellekre és megfigyelési sorozatokra, ez az eljárás egy lehetetlen állapotisorozatot ad vissza (azaz amely sorozatnak az a posteriori valószínűsége nulla).
- 15.5.** Gyakran szeretnénk megfigyelni egy olyan folytonos állapotú rendszert, aminek a viselkedése megjósolhatatlanul vált  $k$  különböző „mód” között. Például egy repülőgép egy rakéta elkerülése közben különböző manővereket hajthat végre, melyeket a rakéta megróbálhat követni. Egy ilyen váltó **Kalman-szűrő** Bayes-háló reprezentációja a 15.22. ábrán látható.
- (a) Tételezzük fel, hogy az  $S_t$  diszkrét állapotnak  $k$  lehetséges értéke van, illetve, hogy a  $P(X_0)$  a priori folytonos állapotbecslés egy többváltozós Gauss-eloszlás. Mutassa meg, hogy a  $P(X_1)$  előrejelzés **Gauss-eloszlások keveréke** (*mixture of Gaussians*) – azaz Gauss-eloszlások súlyozott összege, ahol a súlyok 1-re összegződnek.
- (b) Mutassa meg, hogy ha a jelenlegi folytonos állapot becslése,  $P(X_t|e_{1:t})$ ,  $m$  Gauss-eloszlás keveréke, akkor általános esetben a  $P(X_{t+1}|e_{1:t+1})$  frissített állapotbecslés  $km$  Gauss-eloszlás keveréke lesz.
- (c) Az időbeli folyamatnak milyen jellegét reprezentálják a súlyok a Gauss-keverékben?
- Az (a) és (b) eredmények együtt azt mutatják, hogy az a posteriori eloszlás reprezentációja korlátozás nélkül nő, még a váltó Kalman-szűrők esetében is, amelyek a legegyszerűbb hibrid dinamikus modellek.
- 15.6.** Pótolja a hiányzó lépést a (15.17) egyenlet vezetésében, az első frissítési lépést az egydimenziós Kalman-szűrőnél.
- 15.7.** Vizsgáljuk meg a szórásnégyzet-frissítés alakulását a (15.18) egyenletben.
- (a) Ábrázolja a  $\sigma_i^2$  értékét a  $t$  függvényében, adott  $\sigma_x^2$  és  $\sigma_z^2$  értékek mellett.
- (b) Mutassa meg, hogy a frissítésnek létezik egy  $\sigma^2$  határértéke úgy, hogy  $t \rightarrow \infty$  esetén  $\sigma_i^2 \rightarrow \sigma^2$ , és számítsa ki ezt a  $\sigma^2$  értéket.
- (c) Adjon kvalitatív magyarázatot arra, hogy mi történik, ha  $\sigma_x^2 \rightarrow 0$  és  $\sigma_z^2 \rightarrow 0$ .
- 15.8.** Mutassa meg, hogyan reprezentálna egy RMM-öt egy rekurzív relációs valószínűségi modellként, amint azt a 14.6. fejezetben javasoltuk.
- 15.9.** Ebben a feladatban részletesebben elemezzük az akkumulátorérzékelő tartóhiba-modellt (lásd 15.13. (a) ábra).



**15.22. ábra.** Egy váltó Kalman-szűrő Bayes-háló reprezentációja. Az  $S_t$  váltó változó egy diszkrét állapotváltozó, aminek az értéke meghatározza az  $X_t$  folytonos állapotváltozók átmeneti modelljét. minden  $i$  diszkrét állapota a  $P(X_{t+1}|X_t, S_t = i)$  állapotátmenet-modell egy lineáris Gauss-modell, pontosan úgy, mint egy szabályos Kalman-szűrőben. A diszkrét állapotok közti  $P(S_{t+1}|S_t)$  állapotátmenet-modellt egy mátrixnak tekinthetjük, ahogyan egy rejtett Markov-modellben.

- (a) A 15.13. (b) ábra véget ér  $t = 32$ -nél. Írja le kvalitatívan, mi történik  $t \rightarrow \infty$  esetén, ha az érzékelő továbbra is 0-t mutat.
- (b) Tegyük fel, hogy a külső hőmérséklet befolyásolja az akkumulátor érzékelőjét, mégahozzá olyan módon, hogy a hőmérséklet emelkedésével az átmeneti hibák valószínűbbekké váltnak. Mutassa meg, hogyan egészíténé ki a 15.13. (a) ábra DBH-struktúráját, és magyarázza el az esetlegesen szükséges változtatásokat az FVT-kben.
- (c) Az új hálóstruktúra esetén felhasználhatja-e a robot az akkumulátor méréseket az aktuális hőmérséklet kikövetkeztetésére?
- 15.10.** Gondoljuk meg a változó eliminálás algoritmusának az alkalmazását az esernyős DBH három szeletre történő kibontása esetén a  $P(R_3|U_1, U_2, U_3)$  kérdés megválaszolására. Mutassa meg, hogy az algoritmus komplexitása – a legnagyobb tényező mérete – ugyanaz, függetlenül attól, hogy az első változókat előrefelé vagy visszafelé sorrendben elimináljuk.
- 15.11.** A 15.19. ábra „tomato” modellje megengedi a koartikulációt az első magánhangzón két lehetséges beszédhangot kínálva fel. Egy alternatív megközelítés a háromállapotú modell használata, amelyben az [ow(t,m)] beszédhang automatikusan magában foglalja a magánhangzó változását. Rajzoljon egy teljes háromállapotú modellt a „tomato”-ra, ideértve a tájszólásos változatokat.
- 15.12.** Számolja ki a legvalószínűbb utat a 15.20. ábra RMM-én keresztül a  $[C_1, C_2, C_3, C_4, C_5, C_6, C_7]$  kibocsátási sorozat esetén. Adja meg ennek valószínűségét is.

# 16. EGYSZERŰ DÖNTÉSEK MEGHOZATALA

Ebben a fejezetben láthatjuk, hogyan kell egy ágensnek döntéseket úgy meghoznia, hogy elérje, amit akar – az esetek nagy részében legalábbis.

Ebben a fejezetben visszatérünk a hasznosságelmélet gondolatához, amit a 13. fejezetben vezettünk be. Megmutatjuk a hasznosságelméletnek a valószínűség-számítással való összekapcsolását egy döntéselméleti ágens létrehozásához, amely a meggyőződéseinek és célkitűzéseinek megfelelő racionális döntéseket képes hozni. Az ilyen ágensek képesek döntéseket hozni olyan esetekben is, amikor a bizonytalanság és az ellentétes célok egy logikai ágens számára nem tennék lehetővé a döntést. Egy célorientált ágens valójában az állapotokat két csoportba, egy jó (cél) és egy rossz (nem cél) csoportba sorolja be, míg egy döntéselméleti ágens az állapotok jóságát egy folytonos mértékkel fejezi ki.

A 16.1. alfejezet a döntéselmélet alapvető elveit vezeti be: a várható hasznosság maximálását. A 16.2. alfejezet megmutatja, hogy bármely racionális ágens viselkedése leírható egy hasznosságfüggvény feltételezésével és egy azon alapuló maximálással. A 16.3. alfejezet részletesebben megvizsgálja a hasznosságfüggvény tulajdonságait, különösen ezek kapcsolatát olyan egyedi mennyiségekhez, mint a pénz. A 16.4. alfejezet ismerteti, hogyan lehet a több mennyiségtől függő hasznosságfüggvényeket kezelni. A 16.5. alfejezet leírja a döntéshozó rendszerek megvalósítását. Nevezetesen, bevezetjük a döntési hálózatok (**decision networks**) formalizmust – más néven **hatásdiagramot**, (**influence diagram**) –, ami kiterjeszti a valószínűségi hálózatokat, hogy a „cselekvéseket” és a „hasznosságokat” is tudják kezelní. A fejezet többi része azokat a kérdéseket vizsgálja, amelyek a döntéselméletnek a szakértői rendszerekkel kapcsolatos alkalmazásakor lépnek fel.

## 16.1. MEGGYŐZŐDÉSEK ÉS KÍVÁNSÁGOK ÖSSZEKAPCSOLÁSA BIZONYTALANSÁG ESETÉN

Arnauld francia filozófus az 1662-ben írt *Port-Royal Logic* c. művében azt írta, hogy

Annak megítéléséhez, hogy valaki elérje a jót, és elkerülje a gonoszt, nemesak a jót és a gonoszt kell önmagában megfontolni, hanem annak valószínűségét is, hogy ezek megtörténnek-e vagy sem; és azt az arányt kell megnézni, amely ezek együtteséhez tartozik.

A kortárs tudományos szövegek inkább hasznosságáról beszélnek, mint jóról és gonosról, de az elvek ugyanazok. Az ágens preferenciáit a világ állapotai között egy **hasznosságfüggvény (utility function)** adja meg, ami az egyes állapotok kívánatosságának kifejezésére minden állapothoz egyetlen számot rendel. A hasznosságokat a cselekedetek

következményeinek a valószínűségével kombinálva kapjuk az egyes cselekedetekhez tartozó várható hasznosságot.

Egy  $S$  állapotnak a döntést meghozó ágens szempontja szerinti hasznosságára az  $U(S)$  jelölést fogjuk használni. Mostani vizsgálódásunknál az állapotokat a világ teljes pillanatfelvételének fogjuk tekinteni, hasonlóan a 10. fejezetben szereplő szituációhoz (*situations*). Bár ez egyszerűsíti a kezdeti fejezetet, a hasznosság definiálása minden egyes állapotra külön-külön elég nehézkessé válhat. A 16.4. alfejezetben látni fogjuk, hogy az állapotok hogyan bonthatók fel bizonyos körülmények között a hasznosság hozzárendelése érdekében.

Egy nemdeterminisztikus  $A$  cselekvésnek az  $Eredmény_i(A)$  állapotok a lehetséges következményei, ahol az  $i$  index a különböző következményeken fut végig. Az  $A$  végre-hajtása előtt az ágens egy  $P(Eredmény_i(A)|Tesz(A), E)$  valószínűséget rendel minden egyes következményhez, ahol az  $E$  az ágens által a vilagról elérhető tényeket jelöli, és a  $Tesz(A)$  egy állítás, hogy az  $A$  cselekvés végrehajtódik a jelenlegi állapotban. Ekkor a következő formulával kiszámíthatjuk a cselekvés  $EU(A|E)$  várható hasznosságát (**expected utility**) adott tények esetén:

$$EU(A|E) = \sum_i P(Eredmény_i(A)|Tesz(A), E) U(Eredmény_i(A)) \quad (16.1)$$

**A maximális várható hasznosság (MVH) (maximum expected utility, MEU)** elve azt mondja ki, hogy egy racionális ágensnek azt a cselekvést kell választania, ami maximalizálja az ágens várható hasznosságát. Ha cselekvések egy legjobb sorozatát szeretnénk kiválasztani ennek az egyenletnek a felhasználásával, akkor az összes lehetséges cselekvéssorozatot számba kellene venni, és a legjobbat kiválasztani, ami hosszú sorozatok esetén nyilvánvalóan nem lehetséges. Ezért ez a fejezet egyszerű döntésekre (általában egyetlen cselekvésre vonatkozó döntésekre) koncentrál, és a következő fejezet mutat be új technikákat cselekvéssorozatok hatékony kezelésére.

Bizonyos értelemben az MVH-elv felfogható a teljes MI meghatározásának. Hiszen egy intelligens ágensnek minden össze annyit kell tennie, hogy kiszámítja a különféle mennyiségeket, maximalizálja a hasznosságot a cselekvései felett, és kész. Ám ez nem jelenti, hogy ezzel a definícióval az MI problémaköre meg lenne oldva!

Ámbár az MVH-elv bármely döntési helyzet esetén meghatározza a helyes cselekvést, a szükséges számítások lehetnek kivitelezhetetlenek, és néha maga a probléma megfogalmazása is bonyolult. A világ kezdeti állapotának ismerete érzékelést, tanulást, tudásreprezentációt és következtetést igényel. A  $P(Eredmény_i(A)|Tesz(A), E)$  kiszámítása a világ teljes okozati modelljét igényli, és – ahogyan azt a 14. fejezetben láttuk – NP-teljes számítást a valószínűségi hálózatokban. Az egyes állapotok  $U(Eredmény_i(A))$  hasznosságának a kiszámítása gyakran keresést vagy tervezést igényel, mivel az ágens nem tudja, hogy egy állapot mennyire jó addig, ameddig nem tudja, hogy hova is kerülhet ebből az állapotból. Így a döntéselmélet nem csodaszer, ami megoldja az MI-problémát. Másrészről azonban, ez egy olyan keretet ad, amelyben áttekinthető egy MI-rendszer összes részének a beilleszkedése.

Az MVH-elv egyértelműen kapcsolódik a 2. fejezetben felvetett teljesítménymértékek elköpzeléshez. Az alapötlet nagyon egyszerű. Tekintsük azokat a lehetséges környezeteket, amelyek érzékeléssel és memóriával rendelkező ágenseket tesznek lehetővé, és gondoljuk át azt, hogy milyen lehetséges ágenseket tudnánk tervezni. *Ha egy ágens maximalizálja*



a hasznosságfüggvényét, és az helyesen tükrözi a teljesítmény mértékét, amivel a viselkedését megítélik, akkor ez a lehető legmagasabb teljesítménypontszámot éri el, ha a hasznosságot azon lehetséges környezetek felett átlagoljuk, amelyekbe az ágens kerülhet. Ez az alapvető igazolása az MVH-elvnek. Bár az állítás tautológiának tűnhet, valójában nagyon fontos átmenetet jelent a racionálitás globális, külső kritériumaitól – a múltbeli környezetek felett vett teljesítménytől – egy lokális, belső kritériumig, ami a következő állapotra vonatkozó hasznosságfüggvény maximalizálását jelenti.

Ebben a fejezetben csak az egyszeri vagy egylépéses döntésekkel (*one-shot decisions*) foglalkozunk, bár a 2. fejezet a teljesítménymértékeket a múltbeli környezetek felett definiálja, amelyek általában számos döntést tartalmaznak. A következő fejezetben, ami a szekvenciális döntésekkel (*sequential decisions*) foglalkozik, megmutatjuk, hogy ez a két nézet hogyan békíthető össze.

## 16.2. A HASZNOSÁGELMÉLET ALAPJAI

Ránézésre az MVH-elv ésszerű módszernek látszik döntések meghozatalára, de az nem világos, hogy vajon ez-e az *egyetlen* lehetőség. Végül is, a hasznosság *átlagának* maximálása miért is lenne olyan különleges? Miért nem maximalizáljuk a lehetséges hasznosságok köbeinek összegét, vagy miért nem minimalizáljuk a lehetséges veszteségek maximumát? Hasonlóan, miért nem cselekedhet az ágens racionálisan, ha csupán az állapotok közötti preferenciáit rögzítené, és nem rendelne hozzájuk numerikus értéket? Végül, egyáltalán miért kell léteznie egy megkövetelt tulajdonságokkal rendelkező hasznosságfüggvények? Talán egy racionális ágensnek olyan bonyolult a preferenciarendszere, hogy azt nem lehet ilyen egyszerűen az egyes állapotokhoz rendelt valós számokkal kifejezni.

### Megkötések a racionális preferenciákra

Ezek a kérdések megválaszolhatók, ha leírjuk a preferenciára vonatkozó azon megkötéseket, amelyekkel egy racionális ágensnek rendelkeznie kell, majd ezután megmutatjuk, hogy az MVH-elv ezekből a megkötésekkel levezethető. Az ágens preferenciáit a következő jelöléssel írjuk le:

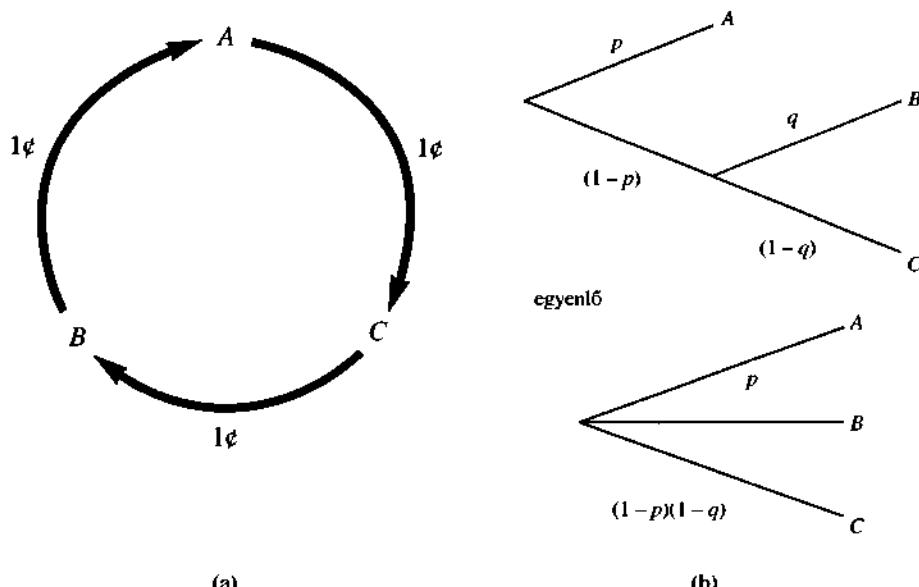
$A > B$       A preferált  $B$ -hez képest

$A \sim B$       az ágens egyformán preferálja  $A$ -t és  $B$ -t

$A \succsim B$       az ágens  $A$ -t preferálja  $B$ -hez képest, vagy egyformán preferálja  $A$ -t és  $B$ -t

Természetesen tisztázni kell, hogy mit is jelölhet  $A$  és  $B$ . Ha az ágens cselekvései determinisztikusak, akkor  $A$  és  $B$  tipikusan a cselekvések konkrét, teljesen specifikált eredményállapotai. Az általánosabb, nemdeterminisztikus esetben  $A$  és  $B$  szerencsejáttékok (*lotteries*). Egy szerencsejáték alapvetően egy valószínűségi eloszlás az aktuális következmények halmaza felett (melyek a szerencsejáték „díjai”). Az  $L$  szerencsejátékot a  $C_1, \dots, C_n$  következményekkel, amelyek  $p_1, \dots, p_n$  valószínűségekkel következhetnek be, a következőképpen jelöljük:

$$L = [p_1, C_1; p_2, C_2; \dots; p_n, C_n]$$



**16.1. ábra.** (a) A cserék egy teljes ciklusa, bermutatva, hogy a nemtranzitív  $A \succ B \succ C \succ A$  preferenciák irrationális viselkedést eredményeznek. (b) A felbonthatósági axióma.

(Az egyetlen következménnyel rendelkező szerencsejáték  $A$ -val vagy  $[1, A]$ -val jelölhető.) Általában egy szerencsejáték bármely következménye lehet egy atomi állapot vagy egy másik szerencsejáték. Az elsődleges kérdés a hasznosságelmélet számára annak megértése, ahogyan az összetett szerencsejátékok közötti preferenciák viszonyulnak ezen szerencsejátékokat meghatározó állapotok közötti preferenciákhöz.

Ennek eléréshéz ésszerű megkötéseket írunk elő a preferenciarelációkra, hasonlóan ahhoz, ahogyan racionális kényszereket vártunk el a meggyőződés mértékeivel kapcsolatban a valószínűség axiómáinak a származtatásánál a 13. fejezetben. Egy ésszerű megkötés, hogy a preferenciák **tranzitívak** (*transitive*) legyenek: azaz ha  $A \succ B$  és  $B \succ C$ , akkor elvárhatjuk, hogy  $A \succ C$ . A tranzitivitás indoklásaként megmutatjuk, hogy egy ágens, amelynek a preferenciái nem tranzitívak, irrationálisan viselkedne. Tételezzük például fel, hogy egy ágens preferenciái nem tranzitívak:  $A \succ B \succ C \succ A$ , ahol  $A$ ,  $B$  és  $C$  áruk, amik szabadon cserélhetők. Ha az ágensnél jelenleg  $A$  van, akkor felkínálhatjuk  $C$ -t cserébe  $A$ -ért és egy kis pénzáért. Ha az ágens jobban szereti  $C$ -t, akkor hajlandó lesz pénzt is áldozni  $C$  megszerzésére. Ezután felkínálhatjuk  $B$ -t  $C$ -ért, még több pénzt begyűjtve, és végül elcseréljük  $B$ -t  $A$ -ért. Ezzel visszaérünk a kezdeti állapotba, azt leszámítva, hogy az ágensnek kevesebb a pénze (16.1. (a) ábra). Ezt a ciklust addig folytathatjuk, ameddig az ágens összes pénze el nem fogy. Indokoltnak látszik azt állítani, hogy az ágens ebben a helyzetben nem cselekedett racionálisan.

A következő hat megkötést a hasznosságelmélet axiómáinak szokás tekinteni. Ezek a preferenciákra és a szerencsejátékokra vonatkozó legnyilvánvalóbb szemantikai megkötéseket határozzák meg.

- **Sorrendezhetőség (orderability).** Bármely két állapot esetén az ágensnek vagy preferálnia kell az egyiket a másikkal szemben, vagy egyformán preferálnak kell minősítenie mindenktől. Azaz az ágens nem kerülhet el a döntést. Ahogyan az 557–558. oldalon tárgyalunk, egy fogadás visszautasítása az idő folyásának a felfüggesztéséhez hasonló.

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

- **Tranzitivitás (transitivity).** Bármely három állapot esetén, ha az ágens preferálja A-t B-vel szemben, és B-t C-vel szemben, akkor az ágensnek preferálnia kell A-t is C-vel szemben.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

- **Folytonosság (continuity).** Ha valamely B állapot A és C között helyezkedik el a preferenciák szempontjából, akkor létezik egy  $p$  valószínűség, amely mellett a racionális ágens számára közömbössé válik, hogy a biztos B-t kapja, vagy egy olyan szerencsejátékot, amiben  $p$  valószínűséggel A,  $1-p$  valószínűséggel pedig C nyerhető.

$$A \succ B \succ C \Rightarrow \exists p [p, A; 1-p, C] \sim B$$

- **Helyettesíthetőség (substitutability).** Ha egy ágens az A és B szerencsejátékot egyformán preferálja, akkor az ágens két összetettebb szerencsejátékot is egyformán preferál, amelyek csak abban különböznek, hogy az egyikben A B-re van cserélve. Ez a szerencsejátékokbeli valószínűségektől és más kimenetektől függetlenül igaz.

$$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$$

- **Monotonitás (monotonicity).** Tételezzük fel, hogy két szerencsejátéknak ugyanaz a két kimenetele van, A és B. Ha az ágens A-t preferálja B-vel szemben, akkor az ágensnek azt a szerencsejátékot kell preferálnia, ami nagyobb valószínűséggel eredményezi A-t (és fordítva is).

$$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \succ [q, A; 1-q, B])$$

- **Felbonthatóság (Decomposability):** Összetett szerencsejátékok egyszerűbbekre bonthatók a valószínűség-számítás szabályai szerint. Ezt „a hazárdjáték unalmas” szabálynak nevezik el, mivel ez azt mondja ki, hogy két egymást követő szerencsejátékot össze lehet olvasztani egyetlen ekvivalens szerencsejátékba, ahogyan azt a 16.1. ábra mutatja.<sup>1</sup>

$$[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$$

## És aztán jött a hasznosság

Vegyük észre, hogy a hasznosságelmélet axiómái nem mondannak semmit a hasznosságról. Kizárolag a preferenciákról szólnak. Azaz a racionális ágens alapvető tulajdonságainak a preferenciákat tételezzük fel. A hasznosságfüggvény létezése a hasznosság-axiómákból következik:

<sup>1</sup> A hazárdjáték elvezetét ekkor úgy modellezhetjük, hogy belefoglalhatjuk a hazárdjáték játszás eseményeit az állapotlefrásokba; például a „10 dollár és hazárdjáték játszás” preferált lehet a „10 dollár hazárdjáték játszás nélkül”-tel szemben.

### 1. A hasznosság elv (utility principle)

Ha az ágens preferenciái eleget tesznek a hasznosság axiómáinak, akkor létezik egy, az állapotokon értelmezett  $U$  valós értékű függvény, mellyel  $U(A) > U(B)$  akkor és csak akkor, ha  $A$  preferált  $B$ -vel szemben, és  $U(A) = U(B)$  akkor és csak akkor, ha az ágens számára  $A$  és  $B$  egyformán preferált.

$$U(A) > U(B) \Leftrightarrow A \succ B$$

$$U(A) = U(B) \Leftrightarrow A \sim B$$

### 2. A maximális várható hasznosság elve (maximum Expected Utility principle)

- Egy szerencsejáték hasznossága az egyes kimenetek valószínűségeivel szorzott kimenetelhasznosságok összege.

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

Más szóval, amint a lehetséges kimeneteli állapotoknak a valószínűségei és hasznosságai specifikáltak, az ezeket tartalmazó összetett szerencsejáték hasznossága teljesen meghatározott. Mivel egy nemdeterminisztikus cselekmény kimenetele egy szerencsejáték, ez egy MVH-alapú döntési szabályt jelent a (16.1) egyenlet szerint.

Fontos megjegyezni, hogy a hasznosságfüggvény léte, ami leírja az ágens preferenciáit, nem jelenti azt szükségszerűen, hogy az ágens *explicit módon* maximalizál egy hasznosságfüggvényt a mérlegelései során. Amint azt a 2. fejezetben megmutattuk, racionális viselkedés számos módon előállítható, némelyik sokkal hatékonyabb, mint a hasznosság maximalizálásának explicit elvégzése. Az ágens preferenciáinak megfigyelésével azonban lehetséges válik a hasznosságfüggvény megkonstruálása, ami az ágens elérni szándékozott céljait reprezentálja.

## 16.3. HASZNOSSÁGFÜGGVÉNYEK

A hasznosság egy függvény, ami valós számokat rendel az állapotokhoz. Vajon ez minden, amit a hasznosságfüggvéniről el lehet mondani? Szigorúan nézve, igen, ennyi. A korábban felsorolt megkötéseknek eleget téve az ágensnek tetszőleges preferenciái lehetnek. Például az ágens preferálhatja, hogy a bankbetétein elhelyezett dollárok száma prímszám legyen; ekkor, ha 16 dollárja volna, 3-at elajándékozna. Lehet, hogy egy 1973-as ütött-kopott Ford Pintót jobban szeret, mint egy csillagó új Mercedest. A preferenciák kapcsolatban állhatnak egymással: például lehet, hogy csak akkor preferál prímszámú dollárt, ha Pintója van, de amikor Mercedese van, akkor a több dollárt jobban szereti, mint a kevesebbet.

Ha az összes hasznosságfüggvény ilyen tetszőleges volna, mint ez, akkor viszont a hasznosságelméletnek nem lenne sok haszna, mivel meg kellene figyelnünk az ágens preferenciáit minden lehetséges körülmény esetén, mielőtt a viselkedésével kapcsolatban bármilyen előrejelzésre is képesek lennének. Szerencsére az igazi ágensek preferenciái sokkal rendszerezettebbek. Ennek megfelelően szisztematikus módszerek léteznek a hasznosságfüggvények megtervezésére, amelyeket aztán egy mesterséges ágensbe építve az ágens a helyes, általunk elvárt viselkedést fogja produkálni.

## A pénz hasznossága

A hasznosságelmélet a közigazdaságtanból származik, a közigazdaságtan pedig egy nyilvánvaló jelölet kínál a hasznosság mérésére: a pénzt (vagy pontosabban az ágens teljes nettó vagyonát). A pénz majdnem univerzális felcserélhetősége bármely áura és szolgáltatásra azt mutatja, hogy a pénz jelentős szerepet játszik az emberi hasznosságfüggvényekben. (Valójában, a legtöbb ember a közigazdaságtant a pénz tanulmányozásához köti, pedig a közigazdaság szó gyökere a gazdálkodásra vonatkozik, míg jelenleg a hangsúly a választások kezelésén van.)

Ha csak azokat a cselekvéseket vesszük figyelembe, amelyek az ágens pénzmennyiségett befolyásolják, akkor általában az lesz a tapasztalat, hogy az ágens a több pénzt előnyben részesíti a kevesebbel szemben, ha egyéb dolgok egyenlők. Azt mondjuk, hogy az ágens **monoton preferenciát** (**monotonic preference**) mutat egy adott összegű pénz esetén. Azonban ez nem elegendő annak garantálására, hogy a pénzt egy hasznosságfüggvénynek tekinthessük, mivel ez semmit sem mond azon szerencsejátékok közötti preferenciákról, amelyek kimenetele pénz.

Tételezzük fel, hogy ön győzedelmeskedett a versenytársak felett egy televíziós játékból. A házigazda most választásra kéri fel: elviheti az 1 000 000 dolláros díjat, vagy felteheti egy pénzfeldobásos hazárdjátékot. Ha fej, nem kap semmit, ha írás, akkor kap 3 000 000 dollárt. Ha hasonló a többi emberhez, akkor vonakodna játszani, és zsebre vágna a milliót. Ez irracionális volna?

Feltéve, hogy hisz az érme szabályos voltában, a játék várható pénzügyi értéke (**VPÉ**) (**expected monetary value**)  $\frac{1}{2}(0 \text{ dollár}) + \frac{1}{2}(3\,000\,000 \text{ dollár}) = 1\,500\,000$  dollár, és az eredeti díj, a VPÉ-je természetesen 1 000 000 dollár, ami kisebb. De ez nem jelenti szüksgszerűen, hogy a hazárdjáték elfogadása a jobb döntés. Tételezzük fel, hogy  $S_n$  jelöli az  $n$  dollárt birtokló állapotot, és a jelenlegi vagyon  $k$  dollár. Ekkor a két cselekedetnek, a hazárdjáték elfogadásának vagy visszautasításának a várható hasznossága:

$$EU(\text{Elfogad}) = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+3\,000\,000})$$

$$EU(\text{Elutasít}) = U(S_{k+1\,000\,000})$$

Annak eldöntéséhez, hogy mit is tegyünk, az egyes kimeneteli állapotokhoz hasznosságot kell rendelnünk. A hasznosság nem közvetlenül arányos a pénzügyi értékkel, mivel a hasznosság – az okozott pozitív változás az életstílusban – az első millió dollár esetén nagyon nagy (vagy legalábbis azt mondják), ezzel szemben a további milliók hasznossága sokkal kisebb. Tételezzük fel, hogy 5-ös hasznosságot rendel a jelenlegi ( $S_k$ ) pénzügyi helyzethez, 10-et az  $S_{k+3\,000\,000}$  állapothoz, 8-at az  $S_{k+1\,500\,000}$  állapothoz. Ekkor a racionális döntés az elutasítás lenne, mivel az elfogadás várható hasznossága 7,5 (kisebb, mint az elutasításhoz tartozó 8). Most tételezzé fel, hogy már van a számláján 500 000 000 dollár (és a játékban való részvétel csak a móka kedvéért történik). Ebben az esetben a hazárdjáték valószínűleg elfogadható, feltéve, hogy a több pénz preferálja a kevesebbel szemben, mivel az 503-adik millió haszna valószínűleg ugyanakkora, mint az 501-edik millióé.

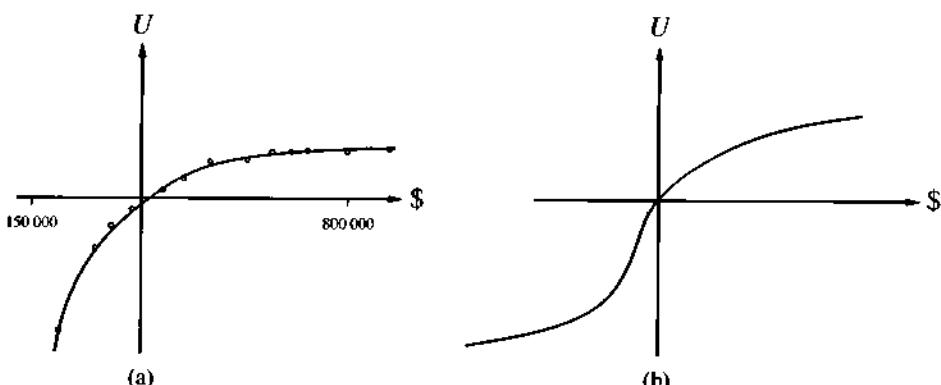
A valóságos hasznossági függvényekről szóló úttörő jelentőségű tanulmányban Grayson azt találta, hogy a pénz hasznossága majdnem teljesen arányos a mennyiségnak *logaritmusával* (Grayson, 1960). (Ezt először Bernoulli vetette fel [Bernoulli,

1783]; lásd 16.3. feladat.) Egy ilyen görbe, egy bizonyos Beard úré, a 16.2. (a) ábrán látható. A Beard úrtól kapott preferenciák konzisztensek a következő hasznossági függvényel:

$$U(S_{k+n}) = -263,31 + 22,09 \log(n + 150\,000)$$

az  $n = -150\,000$  dollár és  $n = 800\,000$  dollár közötti tartományban.

Azt nem tételezhetjük fel, hogy ez egy univerzális hasznosságfüggvénye a vagyoni értéknek, de valószínű, hogy a legtöbb embernek olyan hasznosságfüggvénye van, ami konkáv a pozitív tartományban. Adósságba keveredni általában vészesnek tartott dolog, de a preferenciák a különböző adósságszintek között a pozitív tartománybeli konkávitás megfordulását is mutathatják. Például ha valakinek már van 10 000 000 dollár adósága, akkor lehet, hogy igencsak részt vesz egy olyan pénzfeldobásos hazárdjátékban, ahol 10 000 000 dollár a nyeremény, ha fej, és 20 000 000 dollár a veszteség, ha frás.<sup>2</sup> Ez egy S alakú görbét eredményez, amit a 16.2. (b) ábra mutat.



**16.2. ábra.** A pénz hasznossága. (a) Empirikus adat Beard úrtól egy korlátos tartományban. (b) Egy típus Görbe a teljes tartományban.

Ha csak a görbe pozitív részét vizsgáljuk, ahol a meredekség csökken, akkor itt bár-mely  $L$  szerencsejáték esetén a várható hasznosság kisebb, mint annak a biztos eseménynek a hasznossága, amikor az átadott pénzügyi érték a várható pénzügyi értékkel egyezik meg.

$$U(L) < U(S_{VPÉ(L)})$$

Azaz az ilyen alakú görbével rendelkező ágens **kockázatkerülő** (**risk-averse**): előnyben részesít egy biztos eseményt egy szerencsejátékkal szemben, még akkor is, ha a biztos összeg kisebb, mint a szerencsejáték várható pénzügyi értéke. Másfelől, a 16.2. (b) ábrán a nagy adóságokhoz tartozó „elkeseredett” régióban a viselkedés **kockázatkereső** (**risk-seeking**). Azt az értéket, amit az ágens a szerencsejáték helyett elfogad, a játék determinisztikus **ekvivalensének** (**certainty equivalent**) nevezzük. A tanulmányok

<sup>2</sup> Az ilyen viselkedést elkeseredetneknevezhetjük, mindenazonáltal teljesen racionális, ha valaki kilátástalan helyzetben van.

azt mutatják, hogy a legtöbb ember elfogad körülbelül 400 dollárt egy olyan szerencsejátékért cserébe, ami  $\frac{1}{2}$  valószínűsggel eredményez 1000 dollárt és 0 dollárt – azaz ennek a szerencsejátéknak a determinisztikus ekvivalense 400 dollár. A szerencsejáték várható pénzügyi értéke és a determinisztikus ekvivalense közötti értéket **biztosítási prémiumnak (insurance premium)** hívjuk. A kockázatkerülés az alapja a biztosítási üzletágnak, mivel ez azt jelenti, hogy a biztosítási prémium pozitív. Az emberek inkább fizetnek egy alacsonyabb biztosítási díjat, mintsem hogy kockára tegyék a házuk teljes árát egy lehetséges tűzeset miatt. Viszont a biztosítási társaságok szemszögéből a ház teljes ára igen kicsi a cégek összes vagyonához képest. Emiatt a biztosító hasznossági görbéje megközelítően lineáris ebben a kicsiny régióban, és a biztosítás a társaságnak alig kerül valamibe.

Vegyük észre, hogy a *kis* vagyoni változások az aktuális vagyoni helyzethez képest közel lineárisak a görbe bármely szakaszán. A lineáris görbüvel rendelkező ágenst **kockázatsemlegesnek (risk-neutral)** nevezzük. Ezért kis összegű hazárdjátekok esetén kockázatsemlegességet várhatunk el. Egy bizonyos értelemben ez igazolja azt az egyszerűsített eljárást, ami kis összegű szerencsejátékokkal segítette valószínűségek megbecsülését, és igazolta a valószínűség-számítás axiómáit a 13. fejezetben.

## Hasznosságskálák és a hasznosság megbecslése

A hasznosságxiómák, ha az ágens viselkedési preferenciái rögzítettek, nem határoznak meg egy kitüntetett hasznosságfüggvényt. Például az  $U(S)$  hasznosságfüggvényt a következő alakra transzformálhatjuk:

$$U'(S) = k_1 + k_2 U(S)$$

ahol  $k_1$  egy állandó és  $k_2$  egy tetszőleges pozitív állandó. Nyilvánvaló, hogy ez a lineáris transzformáció nem befolyásolja az ágens viselkedését.

*Determinisztikus* környezetekben, ahol állapotok vannak, és nincsenek szerencsejátékok, a viselkedés nem fog megváltozni semmilyen *monoton* transzformációra. Például vehetjük a köbét az összes hasznosságnak anélkül, hogy ez a cselekmények preferenciarendezését befolyásolná. Egy determinisztikus környezetben lévő ágens esetén azt mondjuk, hogy az ágensnek **értékfüggvénye (value function)** vagy **sorrendezett hasznosságfüggvénye (ordinal utility function)** van; a függvény valójában csak az állapotok sorrendezését biztosítja, és nem ad értelmezhető numerikus értékeket. Ezt a különbségtételt látta a 6. fejezetben a játékok esetén: determinisztikus játékokban, mint a sakk, a kiértékelő függvények értékfüggvények, míg nemdeterminisztikus játékokban, mint az ostábla, ezek valódi hasznosságfüggvények.

### Emberi ítélezethozatal és a hibázás lehetősége

A döntéselmélet egy **normatív elmélet (normative theory)**: azt írja elő, hogy az ágensnek hogyan kell cselekednie. A közigazdaságtan elméletének alkalmazását nagyban javítaná, ha ez egyben egy **leíró (descriptive)** elmélet is volna az aktuális emberi döntéshozatalról. Azonban kísérleti bizonyítékok vannak arra, hogy az emberek szisztematikusan megsértik a hasznosságelmélet axiómáit. Egy

példát a pszichológus Tversky és Kahneman adnak a közgazdász Allais példája alapján (Tversky és Kahneman, 1982; Allais, 1953). A kísérleti személyeknek kétszer kell választaniuk, az *A* és a *B* szerencsejáték, majd a *C* és a *D* szerencsejáték között.

*A*: 4000 dollár 80% eséllyel  
*B*: 3000 dollár 100% eséllyel

*C*: 4000 dollár 20% eséllyel  
*D*: 3000 dollár 25% eséllyel

A többség *B*-t választja *A*-val szemben, és *C*-t *D*-vel szemben. De ha elfogadjuk, hogy  $U(0 \text{ dollár}) = 0$ , akkor az első választásra utal, hogy  $0,8U(4000 \text{ dollár}) < U(3000 \text{ dollár})$ , míg a második választás pontosan a fordítottjára. Másképpen fogalmazva, nincs olyan hasznosságfüggvény, ami ezekkel a választásokkal konzisztnens lenne. Az egyik lehetséges következtetés, hogy az emberek egyszerűen irrationálisak a hasznosságelmélet axiómái alapján. Egy alternatív nézet, hogy az elemzés nem veszi figyelembe a megbánást (regret) – azt az érzést, amiről az emberek tudják, hogy érezni fognak, ha egy biztos nyeremény (*B*) helyett egy 80%-os valószínűségű nagyobb nyereményt választanak, majd vesznek. Más szavakkal, ha *A*-t választja, 20% eséllye van, hogy nem lesz pénze, és úgy érzi magát, mint egy komplett idióta.

Kahneman és Tversky továbbment, és kifejlesztett egy leíró elméletet, ami megmagyarázza, hogy az emberek miért kerülik a kockázatot nagy valószínűségű eseményeknél, viszont miért hajlandók nagyobb kockázatot vállalni valószínűtlen jutalmakkal. A kapcsolat az MI és ezen tapasztalat között az, hogy az ágensünk döntései csak annyira jók, amennyire jók a preferenciák, amin alapulnak. Ha az emberi informátoraink ellentmondásos preferenciákhoz ragaszkodnak, akkor az ágensünk semmit sem tud tenni, hogy konzisztnens legyen veltük.

Szerencsére, az emberek által hozott preferenciaitelek gyakran könnyen felülbírálhatók későbbi megfontolások alapján. Keeney és Raiffa egy korai munkájukban a Harvard Business Schoolon a pénz hasznosságának a megbecslése kapcsán a következő írt (Keeney és Raiffa, 1976, 210. o.):

Kísérleti vizsgálatok nagy része mutatta ki, hogy súlyos hiányosságok vannak a becslési protokollban. A kísérleti személyek kicsiben hajlamosak túlságosan kockázatkerülővé válni és ezért... az illesztett hasznosságfüggvények elfogadhatatlanul nagy kockázati prémiumot mutatnak a nagy tartományú szerencsejátékokra. ...A kísérleti személyek azonban rendezni tudják az inkonziszenciájukat, és úgy érzik, hogy valami fontosat tanultak arról, hogyan akarnak viselkedni. Eredményképpen néhányan lemondották a gépjármű baleseti biztosításukat és több időszakos életbiztosítást kötötték meg.

Az emberi (ir)racionálitás még ma is intenzív vizsgálatok tárgya.

A hasznosságok megbecsülésére egy megoldás, hogy választunk egy skálát a „lehető legnagyobb díj”  $U(S) = u_T$  és a „lehető legnagyobb katasztrófa”  $U(S) = u_{\perp}$  alapján. Normált hasznosságok (normalized utilities) esetén a skála  $u_{\perp} = 0$  és  $u_T = 1$ . A közbeeső kimenetelek hasznosságának a megbecslése úgy történik, hogy az ágenst megkérík, jelezze a preferenciáját az *S* kimeneteli állapot és egy  $[p, u_T; (1-p), u_{\perp}]$  standard szerencsejáték (standard lottery) között. A *p* valószínűséget addig módosítják, amíg az ágens nem preferálja egyformán *S*-et és a standard szerencsejátékot. Normalizált hasznosságokat feltételezve *S* hasznossága *p*.

Orvosi, szállítási és környezetvédelmi problémáknál, egyebek között, emberek élete forog kockán. Ezekben az esetekben  $u_{\perp}$  a közvetlen halálesethez (esetleg halálesetekhez) hozzárendelt érték. Ámbár senkinek sem könnyű értékeket rendelni emberi életekhez, az tény, hogy szünet nélkül kompromisszumokat kötünk. A repülőgépeket csak adott időközönként vetik alá ellenőrzéseknek, amit az utazások és megtett kilométerek határoznak meg, nem pedig minden utazás után. A költségcsökkenések miatt a gépkocsik karosszériája viszonylag vékony fémből készül, a balesetek túlélési arányának csökkenése ellenére. Az ólmozott üzemanyag még mindig igen elterjedt, ámbár jól ismert, hogy az



egészségre káros. Így paradox módon az az ellenkezés, hogy „az élet pénzben nem kifejezhető”, azt eredményezi, hogy az élet *atulértekelt*. Ross Shachter egy kormányhivatali kapcsolatos eléményről számolt be, amely hivatal megrendelte egy tanulmányt az azbeszt iskolákban történő eltávolításáról. A tanulmány egy adott pénzösszeget rendelt az iskolás-korú gyerekek életéhez, és úgy érvelt, hogy ezen feltevés mellett a racionális döntés az azbeszt eltávolítása. A kormányhivatal, morálisan felháborodva teljes mértékben elutasította a jelentést. Ezzel egyben az azbeszt eltávolítása ellen is döntöttek.

Történtek próbálkozások arra nézve, hogy kiderítsék, mennyi értéket tulajdonítanak az emberek az életüknek. Az orvosi és biztonsági elemzésekben két elterjedt „pénznevet” használnak, a mikrohalált (**micromort**) (az egymilliomod esélyét az elhalálozásnak) és a QALY-t, vagy szabványos minőségű életévet (ami ekvivalens egy évvel, jó vagyoni helyzetben, betegségektől mentesen). Számos, személyek igen széles körét vizsgáló tanulmány azt mutatta, hogy a mikrohalál 20 dollárt ér (1980-ban). Azt már látuk, hogy a hasznosságfüggvény nemlineáris, ezért ez nem jelenti azt, hogy a döntéshozó megölné magát 20 millió dollárért. Újra hangsúlyozandó, hogy a hasznossággörbe lokális linearitása azt jelenti, hogy a mikrohalál és a QALY értékei kis relatív kockázatok és jutalmak esetén hasznosak.

## 16.4. TÖBBVÁLTOZÓS HASZNOSSTÁGFÜGGVÉNYEK

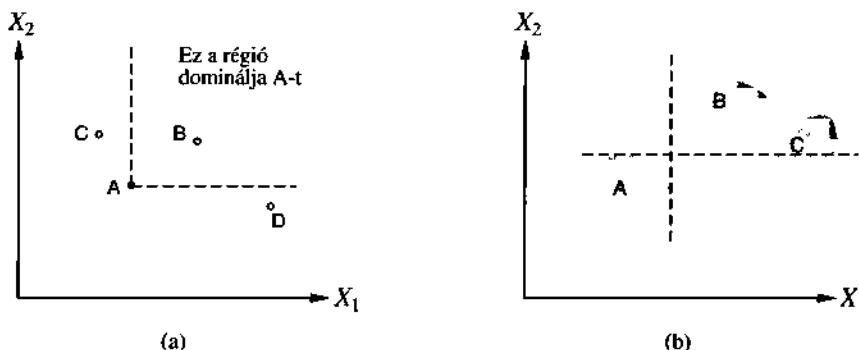
A közérdekű ügyeknél történő döntéshozatal minden dollármillióhoz, minden emberhez és halálához is kapcsolódik. Például annak előttében, hogy rákkeltő anyagok milyen koncentrációi engedhetők meg a környezetben, a döntéshozóknak mérlegelnüük kell a halál megelőzését és azokat a közigazdasági nehézségeket, amelyek bizonyos termékek és folyamatok megszüntetése miatt jelentkezhetnek. Egy új repülőtér elhelyezése esetén meg kell fontolni az építkezés okozta felfordulást; a telkek értékét; a népesebb központuktól való távolságot; a reptér zaját; a biztonsági kérdéseket a helyi földrajzi és az időjárási viszonyok tekintetében és így tovább. Az ehhez hasonló problémákat, ahol a kimenetet két vagy több attribútum jellemzi a többattribútumú hasznosságelmélet (**multiattribute utility theory**) kezeli.

Jelölje  $X = X_1, X_2, \dots, X_n$  az attribútumokat, az attribútumértékek vektorát pedig  $x = \langle x_1, x_2, \dots, x_n \rangle$ . Általában minden attribútumról feltételezzük, hogy diszkrét vagy folytonos értékekkel rendelkezik. Az egyszerűség kedvéért feltételezzük, hogy az attribútumokat úgy határoztuk meg, hogy a nagyobb attribútumértékek nagyobb hasznosságértékekhez tartoznak, ha minden más változatlan. Például a repülőtér probléma esetén minél nagyobb a ZajMentesség értéke, annál jobb a megoldás. Bizonyos esetekben szükséges lehet az értéktartomány felosztása, ahol a hasznosságértékek az egyes tartományokban monoton módon változnak.

Elsőként azokat az eseteket vizsgáljuk meg, amikor döntéseket anélkül hozhatunk, hogy az attribútumértékeket egyetlen hasznosságértékké egyesítenénk. Majd azokat az eseteket vizsgáljuk, ahol az egyes attribútumkombinációk hasznosságát nagyon tömören lehet megadni.

## Dominancia

Tételezzük fel, hogy az  $S_1$  reptér kevesebbe kerül, kisebb zajjal jár és biztonságosabb is, mint  $S_2$ . Ekkor az mondjuk, hogy  $S_1$  szigorúan dominálja (**strict dominance**)  $S_2$ -t. Általában, ha egy lehetőség minden attribútumának kisebb az értéke, mint egy másik lehetőségé, akkor nem szükséges tovább vizsgálni. A szigorú dominancia gyakran nagyon hasznos, mert leszűkíti a választási lehetőségeket a valódi jelöltekre, bár ritkán eredményez egyetlen választási lehetőséget. A 16.3. (a) ábra egy sematikus helyzetet mutat egy kétattribútumos esetben.

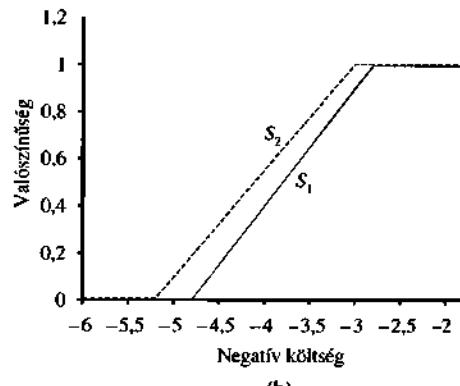
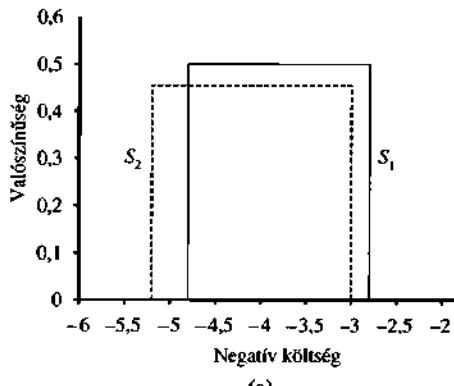


**16.3. ábra** Szigorú dominancia. (a) Determinisztikus: A-t szigorúan dominálja B, de sem C, sem D nem.  
 (b) Bizonytalan: A-t szigorúan dominálja B, de C nem.

Mindez remekül használható determinisztikus esetekben, ahol a tulajdonságértékek biztosan ismertek. De mi történik abban az általános esetben, ahol a cselekvések kimenetele bizonytalan? A szigorú dominancia közvetlen analógiája könnyen megalkotható, ha a bizonytalanság ellenére,  $S_1$  minden lehetséges kimenetele szigorúan dominálja  $S_2$  minden lehetséges kimenetelét. (Ezt mutatja a 16.3. (b) ábra sematikus ábrázolása.) Természetesen ez valószínűleg sokkal ritkábban fordul elő, mint a determinisztikus esetben.

Szerencsére létezik egy használhatóbb általánosítás, az úgynevezett **sztochasztikus dominancia** (**stochastic dominance**), ami valós problémáknál is gyakran előfordul. A sztochasztikus dominanciát könnyebb megérteni egyetlen változó esetében. Tételezzük fel, hogy a reptér költsége az  $S_1$  helyen egyenletes eloszlású 2,8 és 4,8 milliárd dollár között, és az  $S_2$  helyszínen a költség egyenletes eloszlású 3,0 és 5,2 milliárd dollár között. A 16.4. (a) ábra ezen költségek eloszlásait mutatja, ahol a költségek negatív értékként szerepelnek. Ekkor csupán azt az információt ismerve, hogy a hasznosság csökken a költségekkel, azt mondhatjuk, hogy  $S_1$  sztochasztikusan dominálja  $S_2$ -t – azaz  $S_2$  elhagyható. Fontos felismerni, hogy ez nem következik a várható költségek összehasonlításából. Például ha tudnánk, hogy  $S_1$  költsége pontosan 3,8 milliárd dollár, akkor a pénz hasznosságára vonatkozó további információk nélkül nem tudnánk döntést hozni.<sup>3</sup>

<sup>3</sup> Furcsának tűnhet, hogy az  $S_1$  költségéről lévő több információ az ágens döntésképességét csökkenti. A paradoxonra magyarázatot kapunk, ha figyelembe vesszük, hogy az egzakt költséginformáció hiányában meghozott döntés kisebb valószínűséggel a legnagyobb hasznossággal.



**16.4. ábra.** Sztochasztikus dominancia. (a)  $S_1$  sztochasztikusan dominálja  $S_2$ -t a költségek vonatkozásában. (b)  $S_1$  és  $S_2$  negatív költségeinek eloszlásfüggvényei.

Az attribútumok eloszlásai közötti pontos kapcsolat, amely a sztochasztikus dominancia megállapításához szükséges, legkönyebb az eloszlásfüggvény megvizsgálásával látható, ahogy azt a 16.3. (b) ábra is illusztrálja. Az eloszlásfüggvény annak a valószínűséget méri, hogy a költség kisebb-e egy adott összegnél, vagy egyenlő vele – azaz az eloszlásfüggvény az eredeti eloszlást integrálja. Ha  $S_1$  eloszlásfüggvénye minden jobbra esik  $S_2$  eloszlásfüggvényétől, akkor sztochasztikus szemszögből  $S_1$  olcsóbb, mint  $S_2$ . Formálisan, ha az  $A_1$  és az  $A_2$  cselekvések a  $p_1(x)$  és a  $p_2(x)$  valószínűség-eloszlásokra vezetnek az  $X$  attribútumon értelmezve, akkor  $A_1$  sztochasztikusan dominálja  $A_2$ -t az  $X$ -en, ha

$$\forall x \int_{-\infty}^x p_1(x') dx' \leq \int_{-\infty}^x p_2(x') dx'$$

A definíció fontosságát az optimális döntés megválasztásában a következő tulajdonság adja: ha  $A_1$  sztochasztikusan dominálja  $A_2$ -t, akkor bármely  $U(x)$  monoton nem csökkenő hasznosságfüggvény esetén,  $A_1$  várható hasznossága legalább olyan nagy, mint  $A_2$  várható hasznossága. Azaz, ha egy cselekvést egy másik cselekvés minden attribútum esetén sztochasztikusan dominál, akkor az első cselekvés figyelmen kívül hagyható.

Úgy tűnhet, hogy a sztochasztikus dominancia meglehetősen matematikai megközelítést igényel, meghatározása pedig nagy mennyiségi, valószínűséggel kapcsolatos számítást. Valójában azonban számos esetben igen könnyen eldönthető. Tételezzük fel például, hogy az építési költség a népesebb központotól való távolságtól függ. A költség maga bizonytalan, de a nagyobb távolság nagyobb költséget jelent. Ha  $S_1$  közelebbi, mint  $S_2$ , akkor  $S_1$  dominálni fogja  $S_2$ -t a költség tekintetében. Bár ismertetésüktől eltekintünk, de léteznek algoritmusok, amelyek képesek kezelni az ilyen kvalitatív információkat is a bizonytalan változók közötti kapcsolatokról a kvalitatív valószínűségi hálók (**qualitative probabilistic networks**) felhasználásával, lehetővé téve, hogy egy rendszer a sztochasztikus dominancia alapján hozzon racionális döntéseket anélkül, hogy bármilyen numerikus értéket használna.



## A preferenciák rendszere és a többattribútumos hasznosság

Tételezzük fel, hogy  $n$  attribútumunk van, ahol minden egyiknek  $d$  különböző értéke lehet. A teljes  $U(x_1, \dots, x_n)$  hasznosságfüggvény megadásához legrosszabb esetben  $d^n$  érték szükséges. Ez a legrosszabb eset abban a helyzetben áll fent, amikor az ágens preferenciáiban nincs semmilyen szabályszerűség. A többattribútumos hasznosság-elmélet azon a feltételezésen alapul, hogy a legtöbb hasznosságfüggvény sokkal nagyobb strukturáltsággal rendelkezik. Az elfogadott alapvető megközelítés az, hogy a viselkedés preferenciáiban várható szabályszerűségeket azonosítunk, és az úgynevezett **reprezentációs tételeket (representation theorems)** felhasználva megmutatjuk, hogy az adott preferenciarendszerrel rendelkező ágens leírható a következő hasznosságfüggvénnyel:

$$U(x_1, \dots, x_n) = f[f_1(x_1), \dots, f_n(x_n)]$$

ahol  $f$  remélhetőleg olyan egyszerű függvény, mint az összeadás. Látható, hogy ez ahhoz hasonló, ahogy a valószínűségi hálókat használtuk az együttes valószínűségeloszlásfüggvény felbontására.

### Preferenciák bizonytalanság nélkül

Kezdjük a determinisztikus esettel. Emlékezzünk vissza, hogy determinisztikus környezetben az ágensnek egy értékfüggvénye van, : a cél ennek a tömör reprezentálása. A determinisztikus preferenciastruktúrában megtalálható alapvető szabályszerűséget **preferenciafüggetlenségnek (preference independence)** nevezzük. Két attribútum,  $X_1$  és  $X_2$  preferenciálisan független  $X_3$ -tól, ha a preferencia  $\langle x_1, x_2, x_3 \rangle$  és  $\langle x'_1, x'_2, x_3 \rangle$  között nem függ  $X_3$  konkrét értékétől.

Visszatérve a repülőteres problémára, ha egyebek mellett a *Zaj*, a *Költség* és a *Halálesetek* figyelembe veendő attribútumok voltak, felvethető, hogy a *Zaj* és a *Költség* preferenciálisan független a *Halálesetek*-tól. Ekkor például, ha előnyben részesítjük azt az állapotot, ahol 20 000 ember lakik a repülési útvonal közelében, és az építési költség 4 milliárd dollár, azzal az állappittal szemben, ahol 70 000 ember lakik a repülési útvonalon, és az építési költség 3,7 milliárd dollár, ha a biztonsági szint minden esetben 0,06 haláleset millió utas kilométerenként, akkor a preferenciánk a biztonsági szint 0,13 vagy 0,01 értéke esetében is ugyanaz lenne; és ugyanaz a függetlenség állna fenn bár-mely más *Zaj*, *Költség* értékpárra. Az is nyilvánvaló, hogy a *Költség* és a *Halálesetek* preferenciálisan függetlenek a *Zaj*-tól, és a *Zaj* és a *Halálesetek* preferenciálisan függetlenek a *Költség*-tól. Ekkor azt mondjuk, hogy a {*Zaj*, *Költség*, *Halálesetek*} kölcsönösen preferenciálisan függetlenek (KPF) (mutual preferential independence, MPI). A KPF azt állítja, hogy bár lehet, hogy minden egyik attribútum fontos, ez nem befolyásolja a többi attribútum közötti kompromisszumkötést.

A kölcsönös preferenciális függetlenség olyasvalami, mint egy hosszúra sikeredett szókapcsolat, de a közgazdász Debreu egy figyelemre méltó tételenek köszönhetően ez alapján az ágens értékfüggvényét nagyon egyszerű formában felírhatjuk (Debreu, 1960):



*Ha az  $X_1, \dots, X_n$  tulajdonságok kölcsönösen preferenciálisan függetlenek, akkor az ágens viselkedési preferenciája leírható a következő függvény maximalizálásával:*

$$V(x_1, \dots, x_n) = \sum_i V_i(x_i)$$

*ahol minden egyes  $V_i$  egy értékfüggvény, amely csak az  $X_i$  attribútumnak a függvénye.* Például igen valószínű, hogy a repülőtéri döntés meghozható a következő értékfüggvény alapján:

$$V(\text{Zaj}, \text{Költség}, \text{Halálesetek}) = -\text{Zaj} \times 10^4 - \text{Költség} - \text{Halálesetek} \times 10^{12}$$

Az ilyen típusú értékfüggvényt **additív értékfüggvénynek (additive value function)** nevezzük. Az additív értékfüggvények egy teljesen illeszkedő módszert adnak az ágens értékfüggvényének a leírására, és számos valósvilág-beli helyzetben érvényesek. Még ha a KPF nem is teljesül teljes mértékben, ahogy ez az attribútumok szélső értékei esetén megtörténhet, egy additív értékfüggvény még mindig jó közelítését adhatja az ágens preferenciáinak. Ez különösen igaz, ha a KPF csupán azon attribútumtartományokban sérül, amelyek a gyakorlatban csak kis valószínűséggel fordulnak elő.

## Preferenciák bizonytalansággal

Ha a tárgytartományban bizonytalanság is jelen van, a szerencsejátékok közötti preferenciákat is át kell gondolni, és a hasznosságfüggvény kiadódó tulajdonságait is meg kell érteni, nem csak az értékfüggvényekét. Ennek a problémának a matematikája igen bonyolulttá válik, ezért a főbb eredmények közül csak egyet mutatunk be, hogy érzékeltekessük az elérhető eredményeket. Az olvasónak a (Keeney és Raiffa, 1976) irodalmat ajánljuk, ami a terület igen alapos áttekintése.

A **hasznosságfüggetlenség (utility independence)** alapvető fogalma a preferenciák függetlenségét terjeszti ki szerencsejátékokra: az attribútumok **X** halmaza hasznosságfüggetlen az attribútumok **Y** halmazától, ha az **X** attribútumokon alapuló szerencsejátékok közötti preferenciák függetlenek az **Y**-beli attribútumokhoz rendelt értékeitől. Egy tulajdonsághalmaz kölcsönösen **hasznosságfüggetlen (KHF)** (**mutually utility-independent, MUI**), ha minden részhalmaz hasznosságfüggetlen a többi attribútumtól. Ismételten értelmesnek tűnik feltételezni, hogy a repülőtéri építkezés attribútumai KHF-ek.

A KHF-ből következik, hogy az ágens viselkedése leírható egy **multiplikatív hasznosságfüggvénnyel (multiplicative utility function)** (Keeney, 1974). A szorzatalakú hasznosságfüggvény általános alakját egy példán mutatjuk be három attribútum esetén. Az egyszerűség kedvéért, az  $U_i(X_i)$  jelölésére az  $U_i$ -t használjuk:

$$U = k_1 U_1 + k_2 U_2 + k_3 U_3 + k_1 k_2 U_1 U_2 + k_2 k_3 U_2 U_3 + k_3 k_1 U_3 U_1 + k_1 k_2 k_3 U_1 U_2 U_3$$

Bár ez nem tűnik egyszerűnek, három, egyenként egyetlen attribútumhoz kapcsolódó hasznosságfüggvényt tartalmaz, és csupán három állandót. Általában az  $n$  attribútumot tartalmazó KHF-et kielégítő problémát  $n$  számú, egyetlen attribútumú hasznossággal és  $n$  számú állandóval lehet modellezni. minden egyes, egyetlen attribútumot leíró hasznosságfüggvényt a többi attribútumtól függetlenül lehet megtervezni, és ez a kombináció garantáltan előállítja a teljes preferenciarendszert. Hogy tisztán additív függvényt kaphassunk, további feltételek szükségesek.

## 16.5. DÖNTÉSI HÁLÓK

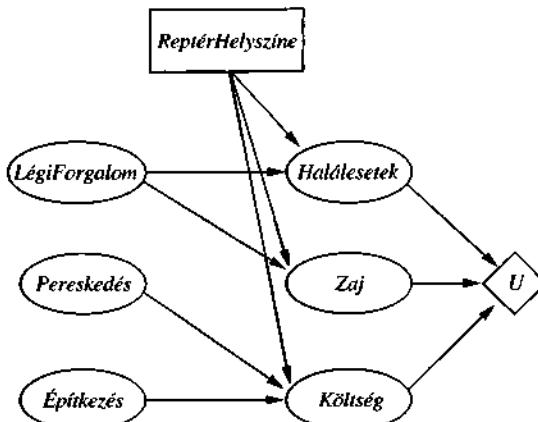
Ebben az alfejezetben egy általános mechanizmust fogunk megvizsgálni a racionális döntések meghozatalára. A felhasznált eszközt gyakran **hatásiagramnak** (*influence diagram*) hívják (Howard és Matheson, 1984), de mi a többet sugalló **döntési hálók** (*decision networks*) elnevezést fogjuk használni. A döntési hálók a cselekvésekhez és a hasznosságokhoz szükséges újabb csomóponttípusok hozzáadásával továbbfejlesztett valószínűségi hálók. A repülőtéri elhelyezési problémát fogjuk használni példának.

### Döntési problémák reprezentálása döntési hálókkal

Egy döntési háló a legáltalánosabb formájában az ágens jelenlegi állapotáról tartalmaz információkat, megadja az ágens lehetséges cselekvéseit, azt az állapotot, amit az ágens cselekvése eredményez, és ennek az állapotnak a hasznosságát. Tehát alapot szolgáltat az olyan hasznosságelvű ágensek megépítéséhez, amilyeneket a 2.4. alfejezetben vezettünk be először. A 16.5. ábra egy döntési hálót mutat a repülőter helyszíne megválasztásának problémájához. Ez bemutatja a három felhasználásra kerülő csomóponttípust:

- **A véletlen csomópontok (chance nodes)** (ovális) valószínűségi változókat jelölnek, pont úgy, ahogy a valószínűségi hálókban. Az ágens lehet bizonytalan az építési költségekben, a légi forgalom szintjében és a pereskedés lehetősége miatt, valamint a *Halálesetek*, a *Zaj* és a teljes *Költség* változókban, amelyek mindegyike szintén függ a kiválasztott helyszíntől. minden egyes véletlen csomópontnak van egy feltételes valószínűségi táblája (FVT), ami a szülőcsomópontok állapotaival indexelt. A döntési hálókban a szülőcsomópontok lehetnek döntési csomópontok és véletlen csomópontok. Kiemelendő, hogy a jelenlegi-állapot véletlen csomópontok bármelyike része lehet egy nagyobb valószínűségi hálónak, ami az építés költségét, a légi forgalom nagyságát vagy a perek lehetőségét becsüli meg.
- **A döntési csomópontok (decision nodes)** (négyszögek) azokat a beavatkozási pontokat reprezentálják, ahol a döntéshozónak lehetősége van a cselekvésre. Ebben az esetben a *ReptérHelyszíne* különböző értékeket vehet fel a megfontolás alatt álló helyszíneknek megfelelően. A választás befolyásolni fogja a kiadódó költséget, a biztonságot és a repülőter által kellett zajt. Ebben a fejezetben feltételezzük, hogy egyetlen döntéssel foglalkozunk (azaz egyetlen ilyen csomópont létezik). A 17. fejezet tárgyalja azokat az eseteket, ahol több mint egy döntést kell meghozni.
- **A hasznosságcsomópontok (utility nodes)** (rombusz) az ágens hasznosságfüggvényét reprezentálják.<sup>4</sup> A hasznosságcsomópontnak szülője az összes olyan változó, amelyek által leírt kimeneteli állapotok közvetlenül befolyásolják a hasznosságot. A hasznosságcsomóponthoz tartozik egy leírás, ami az ágens hasznosságát adja meg egy szülői attribútumokon definiált függvényteljesítményével. A leírás lehet a függvény egy táblázatos megadása, vagy lehet egy parametrikus additív vagy lineáris függvény.

<sup>4</sup> Ezeket a csomópontokat gyakran **értékesomópontoknak** (*value nodes*) nevezik az irodalomban. Mi jobbnak láttuk megtartani a különbözőt a hasznosság- és az értékfüggvények között, ahogyan ezt korábban indokoltuk, ezért lehet, hogy a kimeneti állapot egy szerencsejátékok reprezentál.



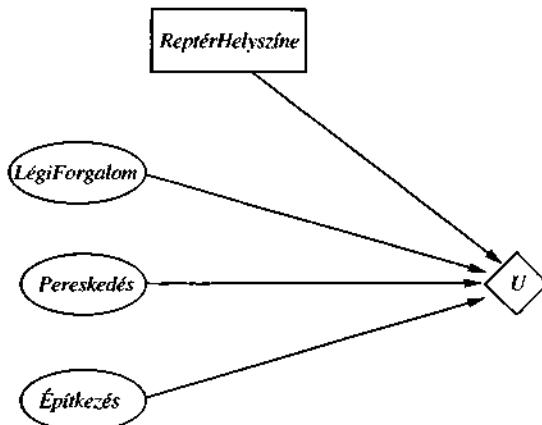
16.5. ábra. Egy egyszerű döntési hálózat a repülőtér-elhelyezési problémára

Sok esetben csupán egy egyszerűsített formát használunk. A jelölés ugyanaz marad, de az a véletlen csomópont, ami a kimeneti állapotot jelöli, elmarad. Ehelyett, a hasznosságcsomópontot közvetlenül a jelenlegi állapot csomópontokhoz és a döntési csomóponthoz köti. Ebben az esetben a hasznosságcsomópont ahelyett, hogy az állapotokon definiált hasznosságfüggvényt reprezentálná, az egyes cselekvésekre vonatkozó várható hasznosságokat reprezentálja, ahogy azt a (16.1) egyenlet definiálja. Ezért az ilyen táblákat **cselekvéshasznosság táblákknak (action-utility tables)** nevezzük. A 16.6. ábra a repülőtér probléma cselekvéshasznosság-reprezentációját mutatja.

Láthatjuk, hogy mivel a *Zaj*, a *Halálesetek* és a *Költség* véletlen csomópontok a 16.5. ábrán a jövőbeli állapotokra vonatkoznak, ezek soha nem kaphatnak értéket, mint tényváltozók. Így az egyszerűsített verzió, amiből elhagytuk ezeket a csomópontokat, minden olyan esetben használható, amikor az általánosabb forma használható. Bár az egyszerűsített forma kevesebb csomópontot tartalmaz, az elhelyezés kimeneteleihez kapcsolódó közvetlen leírások elhagyása azt eredményezi, hogy kevésbé rugalmas a körtülmények esetleges változásaira. Például a 16.5. ábrán a repülőgép zajszintjében bekövetkező változást a *Zaj* csomópont fejtételes valószínűségi táblájának megfelelő megváltoztatásával lehet modellezni, ezzel szemben a zajszennyezés fontosságának a megváltozását a hasznosságfüggvényben a hasznosságtábla megváltoztatásával modellezhetjük. Másfelől a cselekvéshasznosság diagramban – 16.6. ábra – létrejövő minden ilyenfajta változtatást a cselekvéshasznosság táblázat megváltoztatásának kell tükröznie. Lényegében a cselekvéshasznosság forma az eredeti forma egy szerkesztett verziója.

## Döntési hálók kiértékelése

A cselekvéset úgy választjuk ki, hogy a döntési hálót kiértékeljük a döntési csomópont minden lehetséges beállítására. Ha egyszer a döntési csomópontot beállítottuk, úgy fog viselkedni, mint egy véletlen csomópont, amit egy tényváltozónak állítottunk be. A döntési háló kiértékelését a következő algoritmus végzi:



**16.6. ábra.** A repülőtér-elhelyezési probléma egy egyszerűsített reprezentációja. A kimeneti állapotokhoz tartozó véletlen csomópontok hatását kiszámítottuk, majd a csomópontokat elhagytuk.

1. Állítsa be a tényváltozókat a jelenlegi állapotra.
2. A döntési csomópont minden egyes értékére:
  - (a) Állítsa be a döntési csomópontot erre az értékre.
  - (b) Számítsa ki az a posteriori valószínűségeket a hasznosságcsomópont szüleire, egy szabványos valószínűségi következtető algoritmust használva.
  - (c) Számítsa ki a cselekvés hasznosságát.
3. Térjen vissza a legnagyobb hasznosságértékkel.

Ez a valószínűségi hálók következtető algoritmusának egy nyilvánvaló kiterjesztése, és közvetlenül beilleszthető az ágens 13.1. ábrán megadott tervezésébe. A 17. fejezetben látni fogjuk, hogy több egymást követő cselekvés lehetősége a problémát sokkal érdekesebb teszi.

## 16.6. AZ INFORMÁCIÓ ÉRTÉKE

Az előző elemzésben feltételeztük, hogy minden lényeges információ, de legalábbis minden elérhető információ az ágens rendelkezésére áll. Gyakorlatban ez aligha történhet meg. A döntéshozatal egyik legfontosabb része, hogy tudjuk, milyen kérdést kell feltenni. Például az orvos nem várhatja el, hogy rendelkezésére álljon minden lehetséges diagnosztikai teszt és kérdés eredménye, amikor a páciens először belép a rendelőbe.<sup>5</sup> A tesztek gyakran költségesek és néha veszélyesek (mind közvetlenül, mind az általuk okozott késedelmek miatt). A fontosságuk két tényezőn múlik: egyrészt azon, hogy jelentősen jobb kezelés válik-e lehetővé a teszteredmények alapján, másrészt a különböző teszteredmények valószínűségén.

<sup>5</sup> Az Egyesült Államokban az egyetlen kérdés, amit ez előtt minden megkérdeznek, hogy a páciensnek van-e biztosítása.

Ez az alfejezet az **információérték-elmélettel** (**information value theory**) foglalkozik, ami lehetővé teszi az ágens számára annak kiválasztását, hogy milyen információt szerezzen meg. Az információ megszerzése **érzékelési cselekvésekkel** (**sensing action**) történik, ahogy a 12. fejezetben leírtuk. Mivel az ágens hasznosságfüggvénye ritkán vonatkozik az ágens belső állapotaira, jóllehet az érzékelési cselekvés célja a belső állapot megváltoztatása, ezért az érzékelési cselekvéseket az ágens ezt követő műveleteire gyakorolt hatása alapján kell kiértékelnünk. Az információérték-elmélet ezért a többlépcsős döntéshozatal egy speciális fajtája.

## Egy egyszerű példa

Tételezzük fel, hogy egy olajtársaság egy tengerifüróhely létesítésére szeretne jogokat vásárolni,  $n$  darab megkülönböztethetlen terület közül választva. Tételezzük továbbá fel, hogy kizárolag csak az egyik terület tartalmaz olajat  $C$  dollár értékben, és hogy a területek ára  $C/n$  dollár. Ha a társaság kockázatsemleges, akkor közömbös lesz számára, hogy vásárol-e területet, vagy sem.

Most tételezzük fel, hogy a szeizmológusok felkínálják a 3-as számú területre vonatkozó kutatásai eredményét a társaságnak, ami világosan megmutatja, hogy a terület tartalmaz-e olajat, vagy sem. Mennyit legyen hajlandó fizetni a társaság ezért az információért? E kérdés megválaszolásának az a módja, hogy megvizsgáljuk, hogy mit tenne a társaság, ha meglenne az információja:

- $1/n$  valószínűséggel a 3-as terület felmérése azt fogja mutatni, hogy a terület tartalmaz olajat. Ebben az esetben a társaság megvásárolja a 3-as területet  $C/n$  dollárért, és nyeresége  $C - C/n = (n - 1)C/n$  dollár lesz.
- $(n - 1)/n$  valószínűséggel a felmérés azt fogja mutatni, hogy a terület nem tartalmaz olajat, amely esetben a társaság egy másik területet vásárol meg. Ekkor az olajterület eltalálásának a valószínűsége a többi mező között  $1/n$ -ről  $1/(n - 1)$ -re változik, így a társaság várható nyeresége  $C/(n - 1) - C/n = C/n(n - 1)$  dollár lesz.

Most már kiszámíthatjuk a várható nyereséget, ha ismerjük a kutatás eredményét:

$$\frac{1}{n} \times \frac{(n-1)C}{n} + \frac{n-1}{n} \times \frac{C}{n(n-1)} = C/n$$

Tehát a társaságnak hajlandónak kell lennie, hogy akár  $C/n$  dollárt fizessen a szeizmológusnak az információért: az információ annyit ér, mint maga a terület.

Az információ értéke abból a tényből származik, hogy az **információval** valaki az aktuális helyzetben sokkal alkalmasabb cselekedeteket tud megválasztani. Az információval lehetséges válik a helyzetek megkülönböztetése, ezzel szemben az információ nélkül a lehetséges helyzetek feletti átlagot nézve kell cselekedni. Általánosan fogalmazva, az információ értékét a várható értékek különbsége definiálja az információ megszerzése előtt és után.

## Egy általános képlet

Az információ értékére egyszerű általános formulát adni. Rendszerint feltesszük, hogy pontos bizonyítékaink vannak egyes  $E_j$  valószínűségi változók értékeiről, így a **teljes információ értéke (value of perfect information) (TİE)** elnevezést használjuk.<sup>6</sup> Legyen az ágens jelenlegi tudása  $E$ . Ekkor az  $\alpha$  pillanatnyi legjobb cselekvés értékét a következő definiálja:

$$EU(\alpha|E) = \max_A \sum_i U(Eredmény_i(A))P(Eredmény_i(A)|Tesz(A), E)$$

és a legjobb cselekvés értéke (az  $E_j$  új bizonyíték megszerzése után)

$$EU(\alpha_{E_j}|E, E_j) = \max_A \sum_i U(Eredmény_i(A))P(Eredmény_i(A)|Tesz(A), E, E_j)$$

De  $E_j$  egy valószínűségi változó, aminek az értéke *jelenleg* nem ismert, így átlagolunk kell az összes lehetséges  $e_{jk}$  értékre, amelyeket  $E_j$  értékeként az  $E_j$ -re vonatkozó *jelenlegi* hiedelmünket felhasználva megkaphatunk. Az  $E_j$  megszerzésének értékét a következőképpen definiáljuk:

$$TİE_E(E_j) = \left( \sum_k P(E_j = e_{jk} | E) EU(\alpha_{e_{jk}} | E, E_j = e_{jk}) \right) - EU(\alpha | E)$$

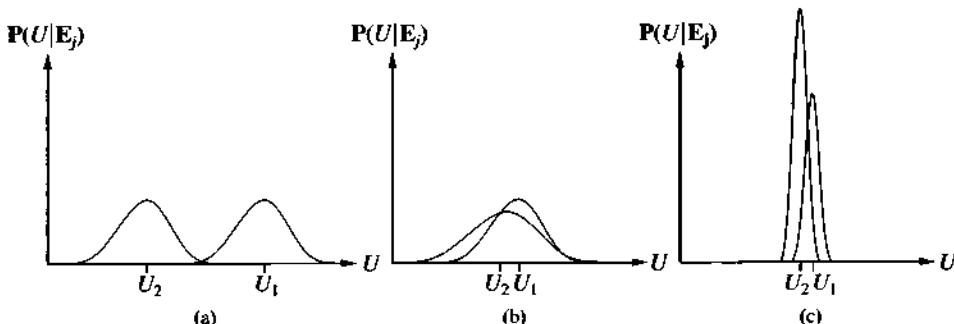
A formula megértése érdekében gondoljuk át azt az egyszerű esetet, mikor csak két cselekvés,  $A_1$  és  $A_2$  közül lehet választani. A cselekvések jelenleg várható hasznosságai  $U_1$  és  $U_2$ . Az  $E_j$  információ új várható hasznosságokat.  $U'_1$ -et és  $U'_2$ -t fog eredményezni a cselekvésekre, de  $E_j$  megszerzése előtt már ismerjük a függetlennek feltételezett  $U'_1$  és  $U'_2$  lehetséges értékeinek valószínűség-eloszlásait.

Tegyük fel, hogy  $A_1$  és  $A_2$  két különböző utat jelent a hegyeken keresztül, teli időben.  $A_1$  egy kényelmes, egyenes autópálya egy alacsony hágón át, míg  $A_2$  egy tekervényes földút a csúcsra keresztül. Csak ezt az információt ismerve  $A_1$  egyértelműen előnyben részesíthető, mivel elég valószínű, hogy a második utat lavinák zárájá le, miközben az első úton valószínűleg nincsenek torlódások.  $U_1$  ezért egyértelműen nagyobb, mint  $U_2$ . Az utak aktuális állapotáról  $E_j$  műholdjelentéseket kaphatunk, melyek  $U'_1$  és  $U'_2$  új várható értékeitől adnak az átkelőkre vonatkozóan. A várható hasznosság-értékek eloszlását a 16.7. (a) ábra mutatja. Nyilvánvalón ebben az esetben a költségek miatt nem éri meg a műholdas jelentéseket kikérni, mivel nagyon valószínűtlen, hogy ezek a jelentések a terv megváltozását eredményeznék. A tervezek változatlansága mellett azonban az információnak nincs értéke.

Most tételezzük fel, hogy két, kissé eltérő hosszúságú kanyargós földút közül választunk, és súlyosan sérült utasokat viszünk. Ekkor bár lehet, hogy  $U_1$  és  $U_2$  elég közel esnek egymáshoz, az  $U'_1$  és  $U'_2$  széles tartományon belül vehetnek fel értékeket. Jelentős esélye van annak, hogy a második út tiszta lesz, míg az első le lesz zárva, és ebben az esetben a hasznosságok különbsége nagyon nagy lesz. A TIÉ formula azt jelzi, hogy a műholdjelentéseket megéri kikérni. Ezt a helyzetet a 16.7. (b) ábra mutatja.

Tételezzük most azt fel, hogy nyári időben választunk a két földút közül, amikor a lavinák igen valószínűtlenek. Ebben az esetben a műholdjelentések azt mutathatják,

<sup>6</sup> Egy  $X$  változóra vonatkozó nem teljes információt modellezhetünk egy  $Y$  változóról szóló teljes információval, ahol az  $Y$  változó valószínűségi kapcsolatban van  $X$ -szel. Erre ad példát a 16.11. feladat.



**16.7. ábra.** Három általános eset az információ értékére. Az (a) esetben  $A_1$  majdnem teljes bizonyossággal jobb marad  $A_2$ -höz képest, így az információ nem szükséges. A (b) esetben a választás nem egyértelmű, és az információ döntő fontosságú. A (c) esetben a választás bizonytalan, de igazából nem számít, így az információ is kevésbé értékese.

hogy az egyik útvonal látványosabb, mint a másik, a virágzó hegyi rétek miatt, vagy talán csak nedvesebb az arra kanyargó patakok miatt. Ekkor igen valószínű, hogy ha megkapjuk ezt az információt, megváltoztatjuk a tervünket. Azonban ebben az esetben, a különbség a két út között még mindig igen kicsi, így nem foglalkozunk a jelentések megszerzésével. Ezt a helyzetet a 16.7. (c) ábra mutatja.

Összegzsűl azt mondhatjuk, hogy *az információ annyiban van értéke, amennyire valószínű, hogy a terv megváltozását okozza, és amennyire az új terv jelentősen jobb lesz, mint a régi.*



## Az információ értékének tulajdonságai

Érdekes kérdés, hogy lehetséges-e az, hogy az információ káros – lehet-e az információnak negatív várható értéke? Az a megérzésünk, hogy ez lehetetlen. Végül is legrosszabb esetben az információ egyszerűen figyelmen kívül hagyható, és úgy tehetünk, mintha soha nem kaptuk volna meg. Ezt igazolja a következő téTEL, ami bármely döntéselméleti agensre alkalmazható. *Az információ értéke nem negatív:*

$$\forall j, E \text{ } TIÉ}_E(E_j) \geq 0$$

A téTEL közvetlenül következik a *TIÉ* definíciójából, a bizonyítást gyakorló feladatként az olvasóra bízzuk (16.12. feladat). Fontos emlékezni arra, hogy a *TIÉ* a jelenlegi információs állapottól függ, ezért szerepel ez az alsó indexben. Ez megváltozhat, amint több információt szerünk. Szélsőséges esetben nulla is lehet, ha a kérdéses változó értéke már ismert. Így a *TIÉ* nem additív. Azaz

$$TIÉ_E(E_j, E_k) \neq TIÉ_E(E_j) + TIÉ_E(E_k) \quad (\text{általánosságban})$$

Azonban a *TIÉ* sorrendfüggetlen, aminek intuitíve nyilvánvalónak kell lennie. Azaz,

$$TIÉ_E(E_j, E_k) = TIÉ_E(E_j) + TIÉ_{E, E_j}(E_k) = TIÉ_E(E_k) + TIÉ_{E, E_k}(E_j)$$

A sorrendfüggetlenség megkülönbözteti az érzékelési és a hagyományos cselekvésekét, és egyszerűsíti egy érzékelési cselekvéssorozat értékének a kiszámítását.

## Egy információgyűjtő ágens megvalósítása

Egy értelmes ágensnek a felhasználóhoz ésszerű sorrendben kell kérdéseket intéznie, nem szabad lényegtelen kérdéseket felennie, az információ fontosságát a költségéhez kell viszonyítania, és a megfelelő ponton abba kell hagynia a kérdezősködést. Mindezen képességek elérhetők az információérték használatának a segítségével.

A 16.8. ábra egy olyan ágens teljes vázát mutatja, ami mielőtt cselekedne, képes intelligensen információkat gyűjteni. Időlegesen tételezzük fel, hogy minden megfigyelhető  $E_j$  tényváltozóhoz létezik egy hozzárendelt költség,  $Költség(E_j)$ , ami a bizonyíték tesztek, tanácsadók, kérdések és bármi egyéb módszer segítségével történő megszerzésének a költsége. Az ágens a legértékesebb információt kéri el, a költségeket is figyelembe véve. Feltételezzük, hogy a  $Kérés(E_j)$  cselekvés eredménye az, hogy a következő érzékelés az  $E_j$  értéket szolgáltatja. Ha nincs olyan megfigyelés, ami megérné a költséget, az ágens egy nem információgyűjtő cselekvést választ ki.

```
function INFORMÁCIÓ-GYŰJTŐ-ÁGENS(észlelés) returns egy művelet
  static: D, egy döntési hálózat

  olvassuk az észlelést a D döntési hálóból
  j ← TIÉ(E)-t maximalizáló érték – Költség(E_j)
  if TIÉ(E_j) > Költség(E_j)
    then return KÉRDEZ(E_j)
  else return a D-beli legjobb művelet
```

**16.8. ábra.** Egy egyszerű információgyűjtő ágens váza. Az ágens folyamatosan a legnagyobb információértékű megfigyelést választja, mindaddig, amíg a következő megfigyelés költsége kisebb, mint a várható haszna.

Az itt megadott ágens az információgyűjtés úgynevezett **rövidlátó (myopic)** formáját valósítja meg. Ez azért van, mert az ágens a  $TIÉ$  formulát rövidlátó módon használja, azaz úgy számítja ki az információ értékét, mintha csak egyetlen tényváltozót szerezne meg. Ha nincsen olyan tényváltozó, ami sokat segítene, lehet, hogy a rövidlátó ágens kapkodó módon belekezd egy cselekvésbe, pedig jobb lett volna előbb lekérdezni két vagy három változót, és csak aztán cselekedni. A rövidlátó szabályozás ugyanazon a heurisztikán alapul, mint a mohó keresés, és a gyakorlatban gyakran jól működik. (Például kimutatták, hogy diagnosztikai tesztek kiválasztásában jobb a teljesítménye, mint az orvosszakértőknek.) Azonban egy teljesen racionális információgyűjtő ágensnek véig kell gondolnia az összes lehetséges információkérés-szekvenciát, ami egy külső akcióban végződik, és ezeknek a kéréseknek az összes lehetséges kimenetelét. Mivel a második kérés értéke függ az első kérés kimenetelétől, az ágensnek fel kell derítenie a feltételes tervek terét, amint azt a 12. fejezetben leírtuk.

## 16.7. DÖNTÉSELMÉLETI SZAKÉRTŐ RENDSZEREK

Az 1950-es és az 1960-as években kifejlődött **döntéselemzés** (**decision analysis**) a döntéselmélet alkalmazását tanulmányozza konkrét döntési problémákon. Fő alkalmazási területe a kiemelt fontosságú területeken meghozandó döntések segítése, ahol a tétek igen nagyok, ilyen területek például az üzleti élet, a kormányzati munka, a törvényhozás, a katonai stratégia, az orvosi diagnosztika és a közegészségügy, a mérnöki tervezés és az erőforrás-gazdálkodás. A folyamat magában foglalja a lehetséges cselekvések és kimenetelek gondos tanulmányozását csakúgy, mint a kimenetelek közötti preferenciákét. A döntéselemzésben hagyományosan két szerepet tételeznek fel: a **döntéshozó** (**decision maker**) a kimenetelek közötti preferenciákat határozza meg, a **döntéselemező** (**decision analyst**) pedig a lehetséges cselekvéset, kimeneteket veszi számba, és kikérdezi a preferenciákat a döntéshozótól, hogy meghatározza a legjobb cselekvést. Az 1980-as évek elejéig a döntéselemzés célját abban látták, hogy az embereket segítsék olyan döntések meghozatalában, amik a tényleges preferenciáknak felelnek meg. Manapság egyre több és több döntési folyamat automatizált, és a döntéselemzést annak biztosítására használják, hogy az automatizált folyamatok megfelelően alakuljanak.

Ahogy erről a 14. fejezetben szoltunk, a korai szakértői rendszerekkel kapcsolatos kutatás inkább kérdések megválaszolására irányult, mint döntések meghozatalára. Azok a rendszerek, amelyek cselekvéset javasoltak a véleménynyilvánítás helyett, valójában ha-akkor szabályokkal érték ezt el, nem pedig a kimenetelek és preferenciák közvetlen reprezentációjával. A valósínűségi hálók megjelenése az 1980-as évek vége körül azonban lehetővé tette olyan nagy rendszerek megépítését, amelyek helyes valósínűségi következtetéseket származtattak a tényekből. A döntési hálók megjelenése azt jelentette, hogy olyan szakértői rendszereket lehet kifejleszteni, amelyek optimális döntéseket javasolnak a felhasználó preferenciáinak és az elérhető tényeknek megfelelően.

Egy rendszer, ami a hasznosságokat közvetlenül tartalmazza, elkerülheti a tanácsadói folyamatok egy gyakori csapdáját: a valósínűség és a fontosság összekeverését. Megszokott módszer volt például a korai orvosszakértői rendszerekben, hogy a lehetséges diagnózisokat a valósínűségük alapján sorba állították, és a legvalósínűbbet jelentették. Sajnos ez végzetes lehet! Általában a páciensek többségénél a két legvalósínűbb diagnózis többnyire az, hogy „semmi baj sincs” és „megfázott”, ha azonban a harmadik legvalósínűbb diagnózis egy páciensnél a tüdőrák, az már aggodalomra ad okot. Nyilvánvaló, hogy a teszteknek és a kezeléseknek mind a valósínűségektől, mind a hasznosságuktól kell függeniük.

A következőkben leírunk egy döntéselméleti szakértői rendszerhez tartozó tudásmérnöki folyamatot. Példaként egy orvosi problémát ismertetünk, amiben kezeléseket kell megválasztani bizonyos veleszületett szívrendellenességek esetén (Lucas, 1996).

A gyermekek 0,8 százaléka szívrendellenességgel születik, ami többnyire aortaszűkület (**aortic coarctation**). Ez a rendellenesség sebészi beavatkozással, érműtéttel (kitágítják az aortát egy ballont helyezve az artériába) vagy gyógyszerrel kezelhető. A feladat a kezelés megválasztása és annak eldöntése, hogy mikor végezzék el: minél fiatalabb egy gyermek, annál nagyobb a kockázata bizonyos kezeléseknek, de túl sokáig sem szabad vájni. A problémához egy döntéselméleti szakértői rendszer készíthető egy olyan csapatral, amelynek legalább egy orvosszakértő (egy gyermekkardiológus) és egy tudásmérnök

tagja van. A folyamat a következő lépésekre bontható (amelyek összehasonlíthatók a logikai rendszerek tervezésének lépéseihez a 8.4. alfejezetben).

**Alkosszon meg egy oksági modellt.** Határozza meg a lehetséges tüneteket, rendellenességeket, kezeléseket és kimeneteleket. Aztán jelezze közöttük haladó nyilakkal, hogy melyik rendellenesség milyen tüneteket okozhat, és melyik kezelés milyen rendellenességekre hatásos. Ezek egy része a tárgyterületi szakértő számára ismert lesz, más részük a szakirodalomból származhatnak. Gyakran a modell jól fog illeszkedni a szakkönyvek leírásaiban szereplő informális diagramokhoz.

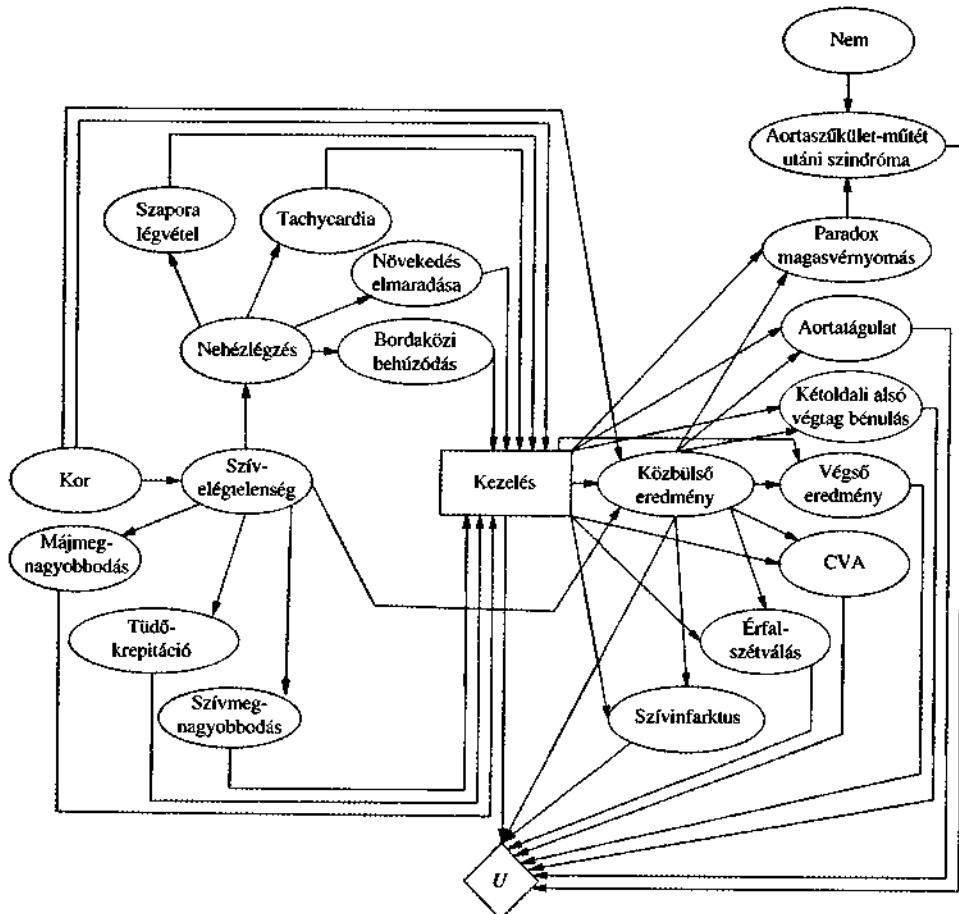
**Egyszerűsítse a kvalitatív döntési modellt.** Mivel a modellt kezelések közötti választásra tervezzük használni, és nem más célokra (mint például bizonyos rendellenességek/tünetek kombinációihoz rendelhető együttes valószínűségek kiszámítására), így gyakran elhagyhatunk változókat, amelyek nem befolyásolják a kezelések megválasztását. Gyakran a változókat fel kell bontani vagy össze kell olvasztani, hogy megfeleljen a szakértő gondolkodásmódjának. Például az eredeti aortaszűkületi modellben szerepelt egy *Kezelés* változó műtét, érsebészeti vagy gyógyszeres kezelés értékekkel, és egy önálló változó a kezelés *Időzítés*-ére. Azonban a szakértőnek nehéz volt ezeket külön kezelní, ezért össze kellett olvasztani, így a *Kezelés* olyan értékeit vehet fel, mint *műtét egy hónapon belül*. Ezt a modellt mutatja be a 16.9. ábra.

**Rögzítse a valószínűségeket.** A valószínűségek betegek adatbázisából, szakirodalmi tanulmányokból vagy szakértők szubjektív becsléseiből származhatnak. Azokban az esetekben, amikor nem megfelelő formában érhetők el a szakirodalomban a valószínűségek, akkor Bayes-szabálytal és vetítéssel számíthatjuk ki a kívánt értékeket. Bebizonyosodott, hogy a szakértők leginkább egy ok hatásának a valószínűségét tudják megbecsülni (például  $P(\text{nehézlégzés}|\text{szívelégtelenség})$ ) sokkal inkább, mint a másik irányban.

**Rögzítse a hasznosságokat.** Kevés számú kimenetel esetén ezek felsorolhatók és egyenként kiértékelhetők. Meghatározhatunk egy skálát a legrosszabbtól a legjobb kimenetig, mindegyikhez egy numerikus értéket rendelve, például -1000-et az elhalálozáshoz és 0-t a teljes felépüléshez. Ekkor a többi kimenetel már ezen a skálán elhelyezhető. Ezt elvégezheti egy szakértő, de jobb, ha a betegek (vagy gyermekek esetén a szüleik) is részt vesznek ebben, mivel a különböző embereknek különböző preferenciáik vannak. Exponenciálisan sok kimenetel esetén szükségünk van olyan módszerekre, hogy többattribútumú hasznosságfüggvényekkel kombinálhassuk őket. Például feltehetjük, hogy a különböző komplikációk negatív hasznossága additív.

**Ellenőrizze és finomítsa a modellt.** A rendszer kiértékeléséhez szükségünk van helyes (bernenet, kimenet) párokra; az úgynevezett referenciákra (**gold standard**), amihez lehet hasonlítani. Ez orvosszakértő rendszerek esetén azt jelenti, hogy összehívjuk az elérhető legjobb orvosszakértőket, néhány beteg esetét megmutatjuk nekik, diagnózis felállítására és kezelési javaslat meghatározására kérjük őket. Majd megnézzük, hogy a rendszer mennyire illeszkedik a javaslataikhoz. Ha csak kevéssé, akkor megpróbáljuk a rosszul teljesítő részeket azonosítani és megjavítani. Hasznos lehet a rendszer „visszafelé” tesztelni. Ahelyett hogy a rendszerbe a tüneteket vinnénk be, és diagnózist kérnék, vigyük be a diagnózist, mint például szívelégtelenség, és vizsgáljuk meg a tünetek, mint például szapora szívverés, jóvolt valószínűségeit, és hasonlítsuk ezt össze szakirodalmi adatokkal.

**Végezzen érzékenységvizsgálatot.** Ez a fontos lépés azt ellenőri, hogy a legjobb döntés érzékeny-e a megadott valószínűségek és hasznosságok kis változására, ehhez



16.9. ábra. Az aortaszükület hatásdiagramja (Peter Lucas hozzájárulásával)

szisztematikusan változtassuk meg ezeket a paramétereket, és futtassuk le a kiértékelést újra. Ha kis változások jelentősen eltérő döntést eredményeznek, akkor lehet, hogy megéri több energiát rászánni, hogy pontosabb adatokat gyűjtünk. Ha minden változtatás ugyanahhoz a döntéshez vezet, akkor a felhasználó biztos lehet, hogy ez a helyes döntés. Az érzékenységvizsgálat különösen fontos lehet, mivel az egyik fő kritikája a szakértői rendszerek valószínűségi megközelítésének, hogy nagyon nehéz a szükséges numerikus valószínűségek megbecsélése. Az érzékenységvizsgálat gyakran kimutatja, hogy sok numerikus értéket elég közelítőleg meghatározni. Például bizonytalanok lehetünk a  $P(\text{tachycardia})$  a priori valószínűségen, de ha kipróbálunk különböző értékeket erre a valószínűségre, és minden egyes esetben a javasolt cselekmény a hatásdiagram alapján ugyanaz, akkor kevésbé kell foglalkoznunk ezzel az ismerethiánnyal.

## 16.8. ÖSSZEFOGLALÁS

Ez a fejezet megmutatta, hogy hogyan kombinálhatjuk össze a hasznosságelméletet a valószínűségekkel, lehetővé téve az ágensnek olyan cselekvések kiválasztását, amelyek maximalizálni fogják a várható teljesítményét.

- A **valószínűség-elmélet (probability theory)** előírja, hogy az ágens a tényei alapján milyen hiedelmekkel rendelkezzen, a **hasznosságelmélet (utility theory)** azt írja elő, hogy az ágens mit akar, a **döntésemellet (decision theory)** pedig összeilleszti a kettőt, és előírja, hogy az ágens mit tegyen.
- A döntésemellet felhasználásával építhetünk egy olyan rendszert, ami a döntések meghozatalakor figyelembe veszi az összes lehetséges cselekvést, és azt választja ki, ami a legjobb várható kimenetelt adja. Egy ilyen rendszert **racionális ágensnek (rational agent)** nevezhetünk.
- A hasznosságelmélet megmutatja, hogy az ágens, amelynek szerencsejátékok közötti preferenciái eleget tesznek bizonyos egyszerű axiómáknak, leírható egy hasznosságfüggvényel; továbbá az ágens a cselekvéseit úgy választja meg, mintha a várható hasznosságát maximalizálná.
- A **többattribútumú hasznosságelmélet (multiattribute utility theory)** olyan hasznosságokkal foglalkozik, amelyek az állapotok több, különböző attribútumától függenek. A **sztochasztikus dominancia (stochastic dominance)** egy különösen hasznos technika egyértelmű döntések meghozatalára még az attribútumok pontos hasznosságértékei nélkül is.
- A **döntési hálók (decision networks)** egy egyszerű formalizmust biztosítanak a döntési problémák leírására és megoldására. Ezek a valószínűségi hálók magától értetődő kiterjesztései döntési és hasznossági csomópontokkal a véletlen csomópontokon túl.
- Bizonyos esetekben a probléma megoldásához hozzátarozik a többletinformáció megszerzése a döntés meghozatala előtt. Az **információ értékét (value of information)** úgy definiáljuk, mint a várható hasznosság javulását az információ nélküli helyzethez hasonlítva.
- Azok a **szakértő rendszerek (expert systems)**, amelyek hasznosságinformációkat is tartalmaznak, többletképességgel rendelkeznek a tisztán következtető rendszerekhez képest. Azon túl, hogy döntéseket tudnak hozni, döntethetnek az információ megszerzéséről annak értéke alapján, és kiszámíthatják döntéseiknek a valószínűségek és hasznosságok kis megváltozására vonatkozó érzékenységét.

## Irodalmi és történeti megjegyzések

A maximális várható hasznosság elvének az egyik legkorábbi, bár végtelen hasznosságokat is tartalmazó szokatlan alkalmazása Pascal fogadása volt, melyet először mint a *Port-Royal Logic* egy részét Arnauld közölt (Arnauld, 1662). Daniel Bernoulli volt az első, a szentpétervári paradoxont vizsgálva, aki felismerte a szerencsejátékokra vonatkozó preferenciák felmérésének a fontosságát, megállapítva, hogy „*egy dolog értékét nem az árának kell meghatároznia, hanem a hasznosságának, amit eredményez*” (Daniel Bernoulli, 1738, eredeti kiemelés). Jeremy Bentham javasolta a **hedonisztikus**

kalkulust (**hedonistic calculus**) az „örömök” és „bánatok” összehasonlítására, amellett érvelve, hogy minden döntésnek (nem csak a pénzügyieknek) redukálhatónak kell lennie hasznosságok összehasonlítására (Bentham, 1823).

A numerikus hasznosságok származtatása a preferenciákból Ramsey-nél jelent meg először (Ramsey, 1931); a jelenlegi szövegben a preferenciákra megadott axiómák megfogalmazásukban közelebb állnak azokhoz az axiómákhoz, amelyek egy újrafeldelezés során a *Theory of Games and Economic Behavior*ben (Neumann és Morgenstern, 1944) jelentek meg. Ezeknek az axiómáknak kiváló bemutatását adja Howard a kockázatpreferenciák tárgyalása során (Howard, 1977). Ramsey szubjektív valószínűségeket (nem pontosan hasznosságokat) vezetett le az ágens preferenciáiból; mostanában Savage és Jeffrey vitt végbe hasonló konstrukciót (Savage, 1954; Jeffrey, 1983). Von Winterfeldt és Edwards moderebb szempontból ismerteti a döntéselemzést és annak az emberi preferenciastruktúrákkal való kapcsolatát (Von Winterfeldt és Edwards, 1986). A mikrohalál hasznosságértéket Howard elemzi (Howard, 1989). Az *Economist* egy 1994-es felmérése egy emberi élet értékét 750 000 és 2,6 millió dollár közzé teszi. Azonban Richard Thaler iracionális variánsokat talált arra nézve, hogy mennyit hajlandó valaki fizetni, hogy elkerüljön egy halálos kockázatot, ahhoz képest, hogy mennyiért hajlandó felvállalni ugyanazt a kockázatot. 1/1000 eséllyel a válaszadó maximum 200 dollárt fizetne, hogy elkerülje a kockázatot, míg 50,000 dollárért sem hajlandó felvállalni ezt a kockázatot (Thaler, 1992).

A QALY-t sokkal szélesebb körben használták orvosi és társadalmi döntések meghozatalánál, mint a mikrohalált, lásd (Russell, 1990), ami jellemző példája azoknak az érveléseknek, hogy a QALY-t használó várható hasznosság növekedésének alapján a közigézségügyben alapos változtatásokra van szükség.

*A Decisions with Multiple Objectives: Preferences and Value Tradeoffs* (Keeney és Raiffa, 1976) c. könyv részletes bevezetést ad a többattribútumú hasznosságelméletbe. Ez olyan módszerek korai számítógépes megvalósítását írja le, amelyekkel a többattribútumú hasznosságfüggvényhez a szükséges paraméterek kikérdezhetők, és hosszasan tárgyalja az elmélet valós alkalmazásait. Az MI-n belül a legelső hivatalos az MVH elvre Wellman cikke volt (Wellman, 1985), amely egy URP-nek (Utility Reasoning Package) nevezett rendszert tartalmaz, ami képes kezelni a preferenciák függetlenségéről és feltételes függetlenségről szóló állításokat, hogy elemezze a döntési probléma struktúráját. A sztochasztikus dominanciát a kvalitatív valószínűségi modellel Wellman vizsgálta alaposan (Wellman, 1988; 1990a). Wellman és Doyle egy előzetes vázlátat adja annak, hogy hogyan lehet felhasználni hasznosság-függetlenség relációk komplex halmazát a hasznosságfüggvény strukturált modelljének megalkotásához, hasonlóan ahhoz, ahogyan a valószínűségi háló az együttes valószínűségeloszlás-függvény strukturált modelljét adja (Wellman és Doyle, 1992). Bacchus és Grove, valamint La Mura és Shoham további eredményeket közöl ezek kapcsolatban (Bacchus és Grove, 1995; 1996; La Mura és Shoham, 1999).

A döntésmélet már az 1950-es évek óta szabványos eszköz a közigazdaságtanban, a pénzügyben és a menedzsment tudományban. Az 1980-as évekig az egyszerű döntési szituációk reprezentálására a döntési fák voltak a fő eszközök. A döntési hálókat vagy hatásiagramokat Howard és Matheson vezette be (Howard és Matheson, 1984), bár a szükséges fogalmakat már korábban kifejlesztette egy csoport (benne Howard és Matheson) az SRI-nél (Miller és társai, 1976). Howard és Matheson eljárása azonban egy közbenső lépést, a döntési hálónak egy döntési fára történő átfirását tartalmazta,

ahol általános esetben a fa exponenciális méretű volt. Shachter kifejlesztett egy eljárást, ami közvetlenül a döntési hálókon alapult anélkül, hogy egy közbeeső döntési fát hozna létre (Shachter, 1986). Ez az algoritmus volt az első, amely teljes következetést tudott elvégezni többszörösen összekötött Bayes-hálókban. Egy mostani munkában Nilsson és Lauritzen összekapcsolta a döntési hálóhoz tartozó algoritmusokat a Bayes-hálóknál kifejlesztett klaszterezési algoritmusokkal (Nilsson és Lauritzen, 2000). Oliver és Smith gyűjteménye számos hasznos cikket tartalmaz a döntési hálókról, mint ahogy a *Networks* folyóirat 1990-es különkiadása is hasznos cikkekkel áll (Oliver és Smith, 1990). Döntési hálókról és hasznosságmodellezésről szóló cikkek a *Management Science*-ben is rendszeresen megjelennek.

Az információérték-elméletet Ron Howard elemezte először (Howard, 1966). Ez a cikke a következő megjegyzéssel ér véget: „Ha az információérték-elmélet és a hozzá kapcsolódó döntéselméleti struktúrák a jövőben nem képezik igen nagy részét a mérnökök képzésének, akkor a mérnöki szakma azt fogja látni, hogy hagyományos szerepét, a tudományos és közigazdasági erőforrásoknak az emberiség előnyére történő kezelését, egy másik szakma elorozta.” A hivatalos forradalom a vezetési módszerekben a mai napig nem következett be, bár ez változhat amint az információérték-elmélet felhasználása a bayesi szakértő rendszerekben elterjedtebbé válik.

Az orvosi döntéseknel történő korai alkalmazásuktól eltekintve meglepően kevés MI-kutató alkalmazta a 13. fejezetben tárgyalta döntéselméleti eszközöket. A kevés kivételek egyike Jerry Feldman volt, aki a döntéselméletet a látás problémájára (Feldman és Yakimovsky, 1974) és a tervkészítés problémájára (Feldman és Sproull, 1977) alkalmazta. Az 1980-as években aztán hirtelen újra megnőtt az érdeklődés az MI részéről a valószínűségi módszerek iránt, a döntéselméleti szakértő rendszerek széles körben elterjedtekkel váltak (Horvitz és társai, 1988) – valójában 1991-től még az *Artificial Intelligence* fedőlapja is egy döntési hálót ábrázolt, bár érve művészeti szabadságukkal egyes nyilak irányát megváltoztatták.

## Feladatok

- 16.1.** (David Heckerman alapján, de a magyar kiadású Guinness Rekordok Könyvére alapozva [Guinness Rekordok Könyve 1991, Solaris Kft., Budapest].) Ez a feladat az **Almanac Game**-hez kapcsolódik, amit döntéselemzők használnak a numerikus becslések kalibrálására. Adjon becslést az alábbi problémákra, azaz azt a számot adja meg, amiről úgy gondolja, hogy ugyanolyan valószínűsséggel lehet túl nagy is, mint túl kicsi. Próbálja meg eltalálni a 25 percentilises becslést is, azaz azt a számot, amiről úgy gondolja, hogy 25% esélye van, hogy túl nagy lesz, és 75% esélye, hogy túl kicsi. Ugyanígy tippelje meg a 75 percentilisest is. (Így minden egyes kérdésre három becslést kell megadnia összesen – egy kisebbet, a mediánt és egy nagyobb értéket.)
- Melyik évben születtek meg az első (névadó) sziámi iker?
  - Hány helyi értéig képes valaki emlékezetből visszaidézni a  $\pi$  számot?
  - Hány rák volt az eddig megfigyelt legnagyobb krill rajban?
  - A kőkorszak óta – a háborúktól és a katasztrófáktól eltekintve – az elhalállozások hány százalékáért felelős a maláriaszúnyog?

- (e) Az állatkertekben végzett mérések szerint egy kifejlett csimpánz hányszor erősebb egy átlagos normális férfinál?
- (f) Az eddigi legmagasabb felállított karácsonya magassága?
- (g) Milyen hosszú a Csendes-óceánt átszelő legrövidebb hajóút hossza?
- (h) A Föld átlagos sűrűsége hányszorosa a vízének?
- (i) Egy évben átlagosan hány esetben észlelnék szeizmikus tevékenységet, ebből mennyi érezhető, és mennyi okoz kárt?
- (j) Milyen magas a hőmérséklet a Holdon a legjobban napsütötte helyen (Egyenlítőre merőlegesen)?

A helyes válaszok a fejezet utolsó feladata után találhatók meg. A döntés-elemzés szempontjából nem az az érdekes, hogy az ön mediántippjei milyen közel vannak a valódi válaszokhoz, hanem inkább az, hogy milyen gyakran van a valódi érték az ön által tippelt 25%-os és 75%-os határ között. Ha ez az esetek felénél így van, akkor a határai pontosak. De ha ön az emberek többségéhez hasonló, akkor a kelleténél magabiztosabb, és az esetek felénél kevesebb szer lesz az igazi érték az ön által tippelt határokon belül. Ezen gyakorlással „javíthat”, és így hasznosabb információt nyújthat a döntéshozzáshoz. Próbálja ki ezt a második kérdéssort, hogy lássa, bekövetkezett-e valamelyen javulás:

- (a) Becslések szerint Picasso hánynak képet, illetve vázlatot festett az élete során?
- (b) Hány statisztai szerepel Sir Richard Attenborough *Ghandi* c., 1982-ben készült filmjének temetési jelenetében?
- (c) Milyen magas a világ legmagasabb építménye (varsói rádiotorony)?
- (d) Milyen hosszú a transzsibériai vasútpálya?
- (e) Hány utas fordult meg az O’Hara chicagói reptörön 1988-ban?
- (f) A világ leglassabban mozgó mechanikus szerkezete (óra) hány év alatt tesz meg egy fordulatot?
- (g) A dobozos sör piacra dobásának éve?
- (h) A világ legnagyobb „tojásgyárában”, a Croton Egg Farmon Amerikában a cég tyúkjai hánynak tojást tojnak naponta?
- (i) Milyen volt a népsűrűség 1968-ban Macao portugál tartományban, a világ legsűrűbben lakott helyén?
- (j) A Man-szigeten üléssezik a megszakítás nélkül leghosszabb ideje fennálló törvényhozás. Az idén hánynak éves?
- (k) Mikor helyezték üzembe az első forgalomirányító jelzőlámpákat (Londonban)?
- (l) Mikor alakult a Nemzetközi Labdarúgó Szövetség (a FIFA)?

- 16.2.** Az állami szerencsejáték 1 dollárba kerül. Két lehetséges díj van: 10 dollár 1/50 valószínűsséggel és 1 000 000 dollár 1/2 000 000 valószínűsséggel. Mennyi a szerencsejáték jegy várható pénzügyi értéke? Mikor racionális (ha van ilyen helyzet egyáltalán) jegyet vásárolni? Pontosabban – mutassa ezt be egy hasznosságokat tartalmazó egyenlettel. Azt feltételezheti, hogy  $U(10 \text{ dollár}) = 10 \times U(1 \text{ dollár})$ , de  $U(1 000 000 \text{ dollár})$ -ra hasonlót nem lehet feltételezni. Szociológiai tanulmányok mutatják, hogy az alacsonyabb jövedelmű emberek aránytalan számú sorsjegyet

vásárolnak. Mit gondol, ez azért van, mert rosszabb döntéshozók, vagy mert eltérő hasznosságfüggvényük van?

- 16.3.** J. Bernoulli 1738-ban vizsgálta a szentpétervári paradoxont, ami a következő. Lehetősége nyílik részt venni egy szerencsejátékban, amelyben egy szabályos érmét dobnak fel ismételten az első fej előfordulásáig. Ha az első fej az  $n$ -edik dobásnál következik be, akkor a nyereménye  $2^n$  dollár.
- Mutassa meg, hogy ennek a játéknak a várható pénzügyi értéke végtelen.
  - Mennyit fizetne ön, személy szerint a részvételért?
  - Ezt a nyilvánvaló paradoxont Bernoulli azzal a javaslattal oldotta meg, hogy a pénz hasznossága logaritmikus skálát követ (azaz  $U(S_n) = a \log_2 n + b$ , ahol  $S_n$  azt az állapotot jelöli, hogy  $n$  dollárunk van). Mi a játék várható hasznossága ezzel a feltevéssel?
  - Mi az a maximum összeg, amit racionális lenne kifizetni a részvételért, feltéve, hogy a kezdeti vagyon  $k$  dollár?
- 16.4.** Becsülje meg a saját hasznosságfüggvényét különböző pénznövekményekre. Ezt úgy teheti meg, hogy egy sor preferenciatesztet futtat egy meghatározott  $M_1$  összeg és egy  $[p, M_2; (1-p), 0]$  szerencsejáték között. Válasszon különböző  $M_1$  és  $M_2$  értékeket, és változassza addig  $p$ -t, ameddig nem lesz ön számára közömbös, hogy melyiket választja a kettő közül. Ábrázolja a hasznosságfüggvényt.
- 16.5.** Írjon számítógépes programot, ami automatizálja a 16.4. feladat folyamatát.   
 Próbálja ki a programot különböző háterű és politikai irányultságú embereken. Elemezze eredményei konziszenciáját az összes személy esetén és egyetlen személy esetén is.
- 16.6.** Mennyit ér egy mikrohalál önnek? Fejlesszen ki protokollt ennek meghatározására. Fogalmazzon meg kérdéseket abból a szempontból, hogy mennyit ér a kockázat elkerülése, és abból a szempontból is, hogy mennyire vállalja a kockázatot.
- 16.7.** Mutassa meg, hogyha  $X_1$  és  $X_2$  preferenciálisan független  $X_3$ -től, és  $X_3$  és  $X_2$  preferenciálisan független  $X_1$ -től, akkor szükségszerűen  $X_1$  és  $X_3$  is preferenciálisan független  $X_2$ -től.
- 16.8.** Ez a feladat a 16.5. ábrán bemutatott repülőtéri elhelyezési probléma elemzését fejezi be.
- Adjon meg ésszerű, a témakörhöz kapcsolódó változókat, valószínűségeket és hasznosságokat a háló számára, feltételezve, hogy három helyszín van.
  - Oldja meg a döntési problémát.
  - Mi történik, ha a műszaki fejlődés következtében a repülőgépek csak fele akkora zajt okoznak?
  - És ha a zaj elkerülése háromszor fontosabbá válik?
  - Számolja ki a TIÉ mennyiséget a LégiForgalom, a Pereskédés és az Építkezés esetén a modellben.

- 16.9.** Ismételje meg a 16.8. feladatot, felhasználva a 16.6. ábrán látható cselekvés-hasznosság-reprezentációt.
- 16.10.** Melyik feltételes valószínűségi tábla bejegyzésre a legérzékenyebb a hasznosság a 16.8. és a 16.9. feladatban konstruált repülőtéri elhelyezési diagramok esetén, ha az elérhető tényeket ismerjük.
- 16.11.** (Judea Pearl-től átvett feladat [Pearl, 1988].) Egy használt autót vásárló eltérő költségek mellett különböző teszteket végeztethet (például megrugdossa a gumikat, autószerelőhöz viszi a kocsit), majd a tesztek eredménye alapján eldönti, hogy melyik autót vegye meg. Feltételezzük, hogy a vásárló akkor veszi meg a  $c_1$  autót, ha elég idő van legalább egy teszt elvégzésére, továbbá, hogy  $c_1$ -nek  $t_1$  a tesztje, és ez 50 dollárba kerül.
- A kocsi jó állapotban ( $q^+$  minőség) vagy rossz állapotban ( $q^-$  minőség) lehet, és a tesztek segíthetnek a kocsi állapotának meghatározásában. A  $c_1$  kocsi 1500 dollárba kerül, piaci értéke azonban 2000 dollár, ha jó állapotban van; ha nem, akkor 700 dollár szükséges a jó állapotba hozásához. A vásárló úgy bocsáli, hogy 70% eséllye van, hogy a kocsi jó állapotban van.
- (a) Rajzolja fel a döntési hálót, ami ezt a problémát reprezentálja.
- (b) Számítsa ki a várható nettó nyereséget  $c_1$  megvásárlása esetén, ha nincsenek tesztek.
- (c) A tesztek azzal a valószínűsséggel jellemezhetők, hogy a kocsi átmegy-e a teszten vagy elbukik, attól függően, hogy jó vagy rossz állapotban van. A következő információkkal rendelkezünk:
- $$P(\text{Átmegy}(c_1, t_1) | q^+(c_1)) = 0,8$$
- $$P(\text{Átmegy}(c_1, t_1) | q^-(c_1)) = 0,35$$
- Felhasználva a Bayes-tételt, számítsa ki annak valószínűségét, hogy a kocsi átmegy (vagy elbukik) a teszten, és így azt a valószínűséget, hogy jó vagy rossz állapotban van adott tesztimenetel esetén.
- (d) Számítsa ki az optimális döntést, ha átmegy, illetve ha elbukik, és számítsa ki a várható hasznosságukat.
- (e) Számítsa ki a teszt információértékét, és vezesse le a vásárló optimális tervét.
- 16.12.** Bizonyítsa be, hogy az információ értéke nem negatív és sorrendfüggetlen, amint azt a 16.6. alfejezetben állítottuk. Magyarázza el, hogyan lehetséges, hogy valaki rosszabb döntést hoz az információ megkapása után, mint amit a megkapása előtt hozott volna.
- 16.13.** Módosítsa és egészítse ki a Bayes-hálós kódot a kódtárban, hogy képes legyen  döntési hálókat létrehozni és kiértékelni, illetve kiszámítani az információ értékét.

A 16.1. feladat válaszai: Első sorozat: 1811, 40 ezer, 10 millió, 50%, 5-6, 67,36 m, 17 550 km, 5,515, 500 000, 100 000, 1000, 117,2 °C. Második sorozat: 13 500, 300 000, 646,28 m, 9438 km, 56 281 000, 25 753, 1935, 3,7 millió, 28 343 fő/km<sup>2</sup>, 1020, 1868, 1904.

# 17. KOMPLEX DÖNTÉSEK MEGHOZATALA

Ebben a fejezetben olyan módszereket vizsgálunk meg, amelyekkel eldönthetjük, hogy mit tegyünk a mai napon, feltételezve, hogy holnap is lehetőségiink lesz a cselekvésre.

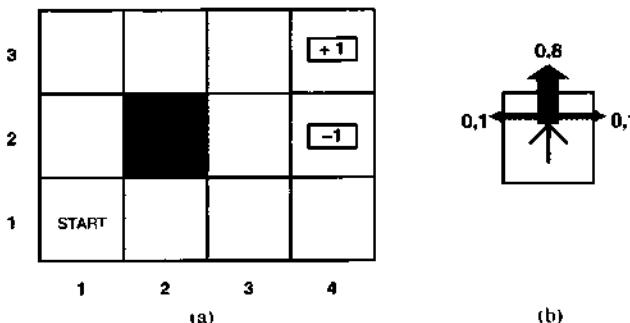
Ebben a fejezetben a döntéshozatalhoz kapcsolódó számítási kérdéseket tárgyaljuk meg. Míg a 16. fejezetben egyszeri vagy epizodikus döntési problémákkal foglalkoztunk, ahol az egyes cselekvések kimeneteleinek a hasznossága jól ismert volt, a 17. fejezetben szekvenciális döntési problémákkal (**sequential decision problems**) fogunk foglalkozni, ahol az ágens hasznossága a döntések sorozatától függ. A szekvenciális döntési problémák, amelyek hasznosságot, bizonytalanságot és érzékelést is magukban foglalnak, a keresés és tervezés problémáját általánosítják, melyeket a II., illetve a IV. részben írtunk le. A 17.1. alfejezet kifejtí, hogyan definiálhatunk szekvenciális döntési problémákat, a 17.2. és a 17.3. alfejezet elmagyarázza ezek megoldását, hogy optimális viselkedést eredményezzenek, amely kiegynésűlyozza a bizonytalan környezetben meghozott cselekvések kockázatát és jutalmát. A 17.4. alfejezet kiterjeszti ezeket az ötleteket a részlegesen megfigyelhető környezetekre, és a 17.5. alfejezet egy teljes tervezési módot fejleszt ki részlegesen megfigyelhető környezetben lévő döntéselméleti ágensek tervezésére, összekapcsolva a dinamikus Bayes-hálókat a 15. fejezetből a 16. fejezetbeli döntési hálókkal.

A fejezet második része többágenses környezeteket tárgyal. Ilyen környezetekben az optimális viselkedés fogalma sokkal bonyolultabbá válik az ágensek közötti interakció miatt. A 17.6. alfejezet ismerteti a játékelmélet (**game theory**) alapötleteit, ideértve azt az elkövetést is, hogy egy racionális ágensnek lehet, hogy véletlenszerűen kell viselkednie. A 17.7. alfejezet azt vizsgálja meg, hogyan tervezhetők olyan többágenses rendszerek, amelyekben az ágensek egy közös célt tudnak megvalósítani.

## 17.1. SZEKVENCIÁLIS DÖNTÉSI PROBLÉMÁK

### Egy példa

Tételezzük fel, hogy egy ágens a 17.1. (a) ábrán látható  $4 \times 3$ -as környezetben helyezkedik el. A kezdő állapotból indulva minden időpontban választania kell egy cselekvést. A környezettel való interakció leáll, ha az ágens eléri a +1-gyel vagy a -1-gyel jelölt állapotot. Az egyes helyzetekben az elérhető cselekvések nevei: *Fel*, *Le*, *Balra*, *Jobbra*. Egyelőre feltételezzük, hogy a környezet **teljesen megfigyelhető** (**fully observable**), azaz az ágens mindenkor ismeri a helyzetét.



**17.1. ábra.** (a) Egy egyszerű  $4 \times 3$ -as környezet, ami az ágens számára szekvenciális döntési probléma. (b) A környezet állapotátmenet-modelljének illusztrálása: a „szándékolt” kimenet 0.8 valószínűséggel következik be, de 0.2 valószínűséggel az ágens oldalra mozdul a szándékolt irányhoz képest. A fallal való ütközéskor nincsen mozgás. A két végállapot jutalma a jelzett +1 és -1, az összes többi állapot jutalma -0.04.

Ha a probléma determinisztikus volna, a megoldás könnyű lenne: [Fel, Fel, Jobbra, Jobbra, Jobbra]. Sajnos a környezet nem maradna minden szinkronban ezzel a megoldással, mivel a cselekvések megbízhatatlanok. A véletlenszerű mozgás egy általunk elfogadott modellt a 17.1. (b) ábra mutatja be. minden cselekvés 0.8 valószínűséggel éri el a kívánt hatását, de a maradék esetben a cselekvések az ágenst a kívánt iránytól jobbra mozgatják. Továbbá, ha az ágens falba ütközik, akkor ugyanazon a mezőn marad. Például ha az (1, 1) kezdő négyzeten áll, a Fel cselekvés az ágenst az (1, 2)-re mozgatja 0.8 valószínűséggel, de 0.1 valószínűséggel a (2, 1)-re kerül, és 0.1 valószínűséggel balra megy, ahol is fálnak ütközik és (1, 1)-en marad. Egy ilyen környezetben a [Fel, Fel, Jobbra, Jobbra, Jobbra] sorozat  $0.8^5 = 0.32768$  valószínűséggel kerüli meg az akadályokat, és éri el a (4, 3) célállapotot. Igen kis eséllyel az is megtörténhet, hogy a célt a másik úton keresztül éri el.  $0.1^4 \times 0.8$ , ami összességében 0.32776 valószínűséget jelent (lásd 17.1. feladat).

Az egyes állapotokban végrehajtott egyes akcióknak a kimeneti valószínűségeit **állapotátmenet-modellnek** (**transition model**) nevezzük. (vagy csak „modell”-nek, ha nem értelmezavaró). A  $T(s, a, s')$  jelölést fogjuk használni annak a valószínűségnak a jelölésére, hogy az  $s$  állapotban az  $a$  cselekvés végrehajtása  $s'$  állapotot eredményez. Feltesszük, hogy az átmenetek teljesítik a 15. fejezetben megfogalmazott **Markov-tulajdonságot**, azaz, hogy  $s'$  elérése  $s$ -ből csak  $s$ -től függ. más korábbi állapotoktól már nem. Egyelőre a  $T(s, a, s')$ -t tekinthetjük a valószínűségek egy nagy háromdimenziós táblázatának. Később, a 17.5. alfejezetben látni fogjuk, hogy az állapotátmenet-modell reprezentálható **dinamikus Bayes-hálókkal** (**dynamic Bayesian network**), csakúgy mint a 15. fejezetben.

Hogy teljessé tegyük a feladat környezetének a definiálását, meg kell adnunk az ágens hasznosságfüggvényét. Mivel a döntési probléma szekvenciális, a hasznosságfüggvény az állapotok sorozatától – a **környezeti történettől** (**environment history**) – fog függni, nem pedig egyetlen állapottól. A fejezetben később megvizsgáljuk az ilyen hasznosságfüggvények általános megadását: egyelőre egyszerűen kikötjük, hogy az ágens minden  $s$  állapotban egy  $R(s)$  **jutalmat** (**reward**) kap, ami lehet pozitív vagy negatív, de mindenképpen korlátos. A konkrét példánkban a jutalom minden állapotban

-0.04. kivéve a végállapotokat (ahol +1 vagy -1). Egy környezeti történet hasznossága (egyelőre) egyszerűen a kapott jutalmak összege. Például ha az ágens a +1 állapotot 10 lépés után éri el, akkor az összhasznossága 0.6 lesz. A -0.04 negatív jutalom arra öszönzi az ágenst, hogy minél gyorsabban érje el a (4, 3)-at, így ez a probléma a 3. fejzetbeli keresési problémák egy sztochasztikus általánosítása. Ez más megfogalmazásban úgy hangzik, hogy az ágens nem élvezи az életet ebben a környezetben, és olyan gyorsan ki akar kerülni, amilyen gyorsan csak lehet.

Egy teljesen megfigyelhető környezetben megadott szekvenciális döntési problémát Markov-állapotátmennet-modelljével és additív jutalmakkal **Markov döntési folyamat**nak neveznek (**MDF**) (**Markov decision process**). Egy MDF-öt a következő három összetevő határoz meg:

Kezdőállapot:  $S_0$

Állapotátmennet-modell:  $T(s, a, s')$

Jutalomfüggvény:<sup>1</sup>  $R(s)$

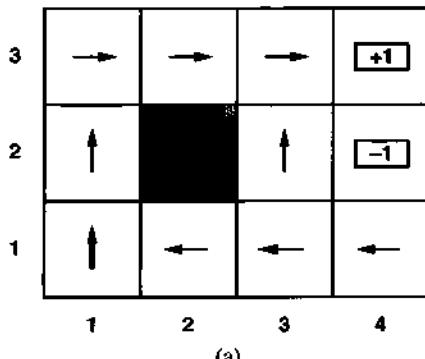
A következő kérdés az, hogyan néz ki egy megoldás a problémára? Azt már láttuk, hogy egy rögzített cselekvéssorozat nem oldja meg a problémát, mivel az ágens végül más állapotba jut a célállapot helyett. Ezért egy megoldásnak minden, az ágens által elérhető állapotra elő kell írnia, hogy az ágens mit tegyen. Egy ilyenfajta megoldás az **eljárásmód (policy)**. Egy eljárásmódot általában  $\pi$ -vel jelölünk, és a  $\pi$  eljárásmód az  $s$  állapotban a  $\pi(s)$  cselekvést javasolja. Ha az ágens egy teljes eljárásmóddal rendelkezik, akkor függetlenül attól, hogy egy cselekvésnek mi is a kimenetele, az ágens mindenig tudni fogja, mit tegyen a következő alkalommal.

Egy adott eljárásmód kezdőállapotból induló végrehajtása minden alkalommal egy különböző környezeti történetet eredményez a környezet sztochasztikus volta miatt. Egy eljárásmód minőségét ezért az adott eljárásmód generálta lehetséges környezeti történetek várható hasznosságával mérik. Az **optimális eljárásmód (optimal policy)** az az eljárásmód, ami a legnagyobb várható hasznosságot eredményezi. Az optimális eljárásmódot  $\pi^*$ -gal jelöljük. A  $\pi^*$  ismeretében az ágens úgy dönti el, hogy mit tegyen, hogy figyelembe veszi az aktuális érzékelését, ami az aktuális  $s$  állapotot közli vele, majd végrehajtja a  $\pi^*(s)$  cselekvést. Az eljárásmód az ágens függvényét explicit módon reprezentálja, és ezért az voltaképpen egy egyszerű reflexágens leírása, amit egy hasznosságalapú ágens által használt információkból számítunk ki.

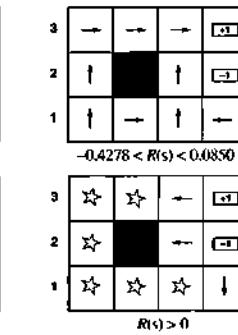
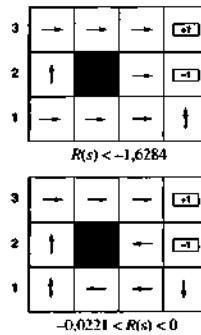
A 17.1. ábrán mutatott világ optimális eljárását a 17.2. (a) ábra mutatja. Látható, hogy mivel egy lépés költsége meglehetősen kicsi ahhoz a bùntetéshez viszonyítva, hogy véletlenül a (4, 2) pozícióra kerülünk, az optimális eljárásmód a (3, 1) állapot esetén óvatos. Inkább egy hosszú kerülő út megtételét javasolja, mint a rövid átvágást a (4, 2) kockázatával.

A kockázat és a jutalom egyensúlyának megváltozása függ az  $R(s)$  értékétől a nem végső állapotoknál. A 17.2. (b) ábrán az  $R(s)$  értékének négy különböző intervallumához tartozó optimális eljárásmódot találhatunk. Ha  $R(s) \leq -1.6284$ , akkor az élet anynyira elviselhetetlen, hogy az ágens egyenesen a legközelebbi kijárathoz tart, annak

<sup>1</sup> Az MDF egyes definíciói megengedik, hogy a jutalom függjen a cselekvéstől és a kimeneteltől is, így a jutalomfüggvény  $R(s, a, s')$  alakú. Ez néhány környezet leírását egyszerűsíti, de a problémát alapvetően nem változtatja meg.



(a)



(b)

**17.2. ábra.** (a) Egy optimális eljárásmód a sztochasztikus környezetre  $R(s) = -0,04$  esetén nem végállapotknál. (b) Optimális eljárásmódok  $R(s)$  négy különböző értéktartománya esetén.

ellenére, hogy ez  $-1$  értékű. Amikor  $-0,4278 \leq R(s) \leq -0,0850$ , az élet elég kellemetlen; az ágens a  $+1$  állapothoz vezető utat választja, vállalva annak kockázatát, hogy esetleg a  $-1$  állapotba kerül. Nevezetesen az ágens a  $(3, 1)$ -ben választja az átvágást. Amikor az élet csak kevésbé bátoros ( $-0,0221 < R(s) < 0$ ), az optimális eljárásmód nem vállal *semmilyen kockázatot*. A  $(4, 1)$  és  $(3, 2)$  állapotokban az ágens teljesen elkerüli a  $-1$  állapotot, így nem tud véletlenül beleesni, még ha ez azt is jelenti, hogy elég sokszor beveri a fejét a falba. Végül, ha  $R(s) > 0$ , akkor az élet kifejezetten elvezethető, és az ágens *mindkét kijáratot elkerüli*. Mindaddig, amíg a  $(4, 1)$ ,  $(3, 2)$  és  $(3, 3)$  állapotokban a cselekvések azok, amelyek az ábrán láthatók, minden eljárásmód optimális, és az ágens végétlen teljes jutalmat ér el, mivel soha nem lép be a végállapotba. Meglepő módon megmutatható, hogy hat más optimális eljárásmód is létezik az  $R(s)$  különböző intervallumaira. A 17.7. feladat kéri majd ezek megkeresését.

A kockázat és a jutalom óvatos egyensúlyozása az MDF-ek egy olyan meghatározó tulajdonsága, ami nem jelentkezik determinisztikus keresési problémáknál; másfelől ez meghatározó tulajdonsága számos valós világ-beli döntési problémának. Emiatt az MDF-eket számos tudományterületen tanulmányozzák, ideértve az MI-t, az operációkutatást, a közigazdaságtant és a szabályozáselméletet. Algoritmusok tücatjait javasolták optimális eljárásmódok kiszámítására. A 17.2. és 17.3. alfejezetben leírunk a legfontosabb algoritmuscsaládok közül kettőt. Először azonban be kell fejeznünk a hasznosságok és eljárásmódok vizsgálatát a szekvenciális döntési problémák esetében.

## Optimalitás szekvenciális döntési problémákban

A 17.1. ábrán látható MDF-példában az ágens teljesítményét a meglátogatott állapotokban kapott jutalmak összege mérte. Ez a teljesítménymérték nem önkényes, de nem is az egyetlen lehetőség. Ez az alfejezet a lehetséges alternatív teljesítménymértékeket vizsgálja – azaz alternatív hasznosságfüggvényeket a környezeti történéseken, amit úgy jelölünk, hogy  $U_h([s_0, \dots, s_n])$ . Az alfejezet a 16. fejezetből származó ötleteken alapul, és a technikai részleteket is bemutatja; a főbb pontokat a végén összegezzük.

Az első megválaszolandó kérdés az, hogy **véges horizont** (**finite horizon**) vagy **végtelen horizont** (**infinite horizon**) van a döntéshozatalnál. A véges horizont azt jelenti, hogy létezik egy *rögzített*  $N$  idő, ami után semmi nem érdekes – a játéknak vége, mondhatni. Így  $U_h([s_0, \dots, s_{N+k}]) = U_h([s_0, \dots, s_N])$  minden  $k > 0$  esetén. Például tegyük fel, hogy az ágens (3, 1)-ból indul a 17.1. ábra  $4 \times 3$ -as világában, és tegyük fel, hogy  $N = 3$ . Ekkor, hogy a +1 állapot elérésének legalább az esélye meglegyen, az ágensnek egyenesen felé kell tartani, és az optimális cselekvés a *Fel*. Ezzel szemben, ha  $N = 100$ , akkor bőségesen van idő a biztonságos utat követni *Balra* menve. Azaz, véges horizont esetében az optimális cselekvés egy adott állapotban idővel változhat. Azt mondjuk, hogy az optimális eljárásmódban véges horizont esetében **nemstacionárius** (**nonstationary**). Rögzített időkorlát hiányában ezzel szemben nincs ok különböző viselkedésre ugyanabban az állapotban más és más időpontokban. Így az optimális cselekvés csak az aktuális állapottól függ, és az optimális eljárásmódban **stacionárius** (**stationary**). A végtelen horizontú esethez tartozó eljárásmódot ezért egyszerűbbek, mint a véges horizontú esethez tartozók, és ebben a fejezetben mi főként a végtelen horizontú esettel foglalkozunk.<sup>2</sup> Vegyük észre, hogy a „végtelen horizont” nem jelenti szükségszerűen azt, hogy az összes állapot sorozat végtelen; minden annyit jelent, hogy nincs egy rögzített határ. Nevezetesen, egy végtelen horizontú MDF-ben lehetnek végállapotot tartalmazó véges állapot sorozatok.

A következő előtörendő kérdés az állapot sorozatok hasznosságának a kiszámítása. Tekinthetjük ezt a kérdést a **többattribútumú hasznosságelmélet** (**multiattribute utility theory**) egy kérdésének (lásd 16.4. alfejezet) azzal, hogy minden  $s_i$  állapotot az  $[s_0, s_1, s_2, \dots]$  állapot sorozat egy attribútumának veszünk. Egy egyszerű kifejezés eléréséhez, ami az attribútumokból épül fel, fel kell tételeznünk valamilyen preferenciafüggetlenséget. A legtermészetesebb feltevés az, hogy az ágens preferenciái az állapot sorozatok között **stacionáriusok** (**stationary**). A preferenciákra vonatkozó stacionaritás a következőket jelenti: ha két állapot sorozat,  $[s_0, s_1, s_2, \dots]$  és  $[s'_0, s'_1, s'_2, \dots]$ , ugyanazzal az állapottal kezdődik (például  $s_0 = s'_0$ ), akkor a két sorozat a preferencia-sorrendjének ugyanannak kell lennie, mint az  $[s_1, s_2, \dots]$  és az  $[s'_1, s'_2, \dots]$  sorozatoknak. Ez tehát azt jelenti, hogy ha egy holnapról kezdődő lehetséges jövőbeli eseménysort egy másik eseménysorral szemben preferálunk, akkor ez nem változhat, ha az eseménysor a mai naptól kezdődne. A stacionaritás egy meglehetősen ártalmatlannak tűnő feltevés, nagyon erős következményekkel: bizonyítható, hogy stacionaritás esetén a sorozatokhoz csak kétféle módon rendelhetők hasznosságok:

### 1. Additív jutalmak (additive rewards): egy állapot sorozat hasznossága ekkor

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

A 17.1. ábra  $4 \times 3$ -as világában additív jutalmakat használ. Vegyük észre, hogy az additivitást hallgatólagosan kihasználtuk az utak költségsűrűségeinek felhasználásakor a heurisztikus keresési algoritmusokban (4. fejezet).

### 2. Leszámított jutalmak (discounted rewards): egy állapot sorozat hasznossága ekkor

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

<sup>2</sup> Ez a teljesen megfigyelhető környezetekre igaz. Látni fogjuk, hogy a részlegesen megfigyelhető környezetekben a végtelen horizont esete nem annyira egyszerű.

ahol a  $\gamma$  leszámítolási tényező (**discount factor**) egy 0 és 1 közötti szám. A leszámítolási tényező fejezi ki egy ágens preferenciáját a jelenlegi és a jövőbeli jutalmak között. Amikor  $\gamma$  közel 0, akkor a távoli jövőbeli jutalmakat jelentéktelennek tekinti. Amikor  $\gamma$  értéke 1, akkor a leszámított jutalmak pontosan megegyeznek az additív jutalmakkal, így az additív jutalmak a leszámított jutalmak speciális esetei. A leszámítolás jó modellnek tűnik mind az állati, mind az emberi időbeli preferenciákra. Egy  $\gamma$  leszámítolási tényező ekvivalens egy  $(1/\gamma) - 1$  kamatlábbal.

A fejezet hátralevő részében leszámított jutalmakat fogunk feltételezni, aminek okai hamarosan világosak lesznek, bár néha megengedjük a  $\gamma = 1$  értéket.

A végtelen horizont elfogadása mögött azonban rejlik egy probléma: ha a környezet nem tartalmaz egy végállapotot, vagy ha az ágens soha nem jut végállapotba, akkor az összes környezettörténet végtelen hosszú lesz, és a hasznosságok additív jutalmakkal általában végtelenek lesznek. Abban egyetérthetünk, hogy a  $+\infty$  jobb, mint a  $-\infty$ , de két  $+\infty$  hasznosságú állapot sorozat összehasonlítása már bonyolultabb. Három megoldás kínálkozik, amelyek közül kettőt már láttunk:

1. Leszámított jutalmakkal egy végtelen sorozat hasznossága véges. Valójában, ha a jutalmakra létezik egy  $R_{\max}$  korlát és  $\gamma < 1$ , akkor azt kapjuk, hogy

$$U_h([s_0, s_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R_{(s_t)} \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max} / (1 - \gamma) \quad (17.1)$$

felhasználva a végtelen mértani sorozat összegképletét.

2. Ha a környezet tartalmaz végállapotokat, és ha garantált, hogy az ágens végül bekerül az egikbe, akkor soha nem lesz szükségünk végtelen sorozatok összehasonlítására. Egy eljárásmódot, ami garantáltan végállapotba juttat, véges eljárásmódnak (**proper policy**) nevezünk. Véges eljárásmódonál használhatjuk a  $\gamma = 1$ -t (azaz az additív jutalmakat). A 17.2. (b) ábrán látható első három eljárásmód véges, a negyedik azonban nem. Egy nem véges eljárásmód végtelen teljes jutalmat ér el a végállapotktól való távolmaradással, amikor a nem végállapotok jutalma pozitív. A nem véges eljárásmódonakra additív jutalmaknál nem minden működnek az MDF-eket megoldó alapalgoritmusok, ami jó ok a leszámított jutalmak használatára.
3. Végtelen sorozatok összehasonlítására egy másik lehetőség az időegységenkénti átlagjutalom (**average reward**) felhasználása. Tegyük fel, hogy a  $4 \times 3$ -as világban az  $(1, 1)$  mező jutalma 0.1, míg más nem végállapotoké 0.01. Ekkor egy olyan eljárásmódnak, ami minden tőle telhetőt megtesz, hogy az  $(1, 1)$ -ben maradjon, nagyobb lesz az átlagos jutalma, mint annak, amelyik máshol marad. Az átlagos jutalom hasznos kritérium bizonyos problémákra, de az átlagjutalomra tervezett algoritmusok meghaladják e könyv kereteit.

Összegezve, a leszámított jutalmak használata jár a legkevesebb bonyodalommal az állapot sorozatok kiértékelésében. A végső lépés annak megmutatása, hogyan válaszszunk az eljárásmódot között, észben tartva azt, hogy egy adott  $\pi$  eljárásmód nem egyetlen állapot sorozatot generál, hanem a legkülönfélébb lehetséges állapot sorozatokat, mindegyiket egy adott valószínűséggel, amit a környezet állapot átmenet-modellje határoz meg. Így egy eljárásmód értéke a (sorozatonként) elérő leszámított jutalmak várható értéke, ahol a várható érték képzése az összes állapot sorozat felett

történik, ami az eljárás végrehajtása esetén előfordulhat. Egy  $\pi^*$  optimális eljárás-mód ekkor

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right] \quad (17.2)$$

A következő két alfejezet algoritmusokat ír le az optimális eljárásmódok megtalálására.

## 17.2. ÉRTÉKITERÁCIÓ

Ebben az alfejezetben az optimális eljárásmód kiszámítására mutatunk be egy algoritmust, az úgynevezett értékiterációt (**value iteration**). Az alapötlet az, hogy kiszámítjuk minden egyes állapot hasznosságát, majd az állapothasznosságokat felhasználjuk az egyes állapotoknál az optimális cselekvés megválasztásához.

### Az állapotok hasznossága

Az állapotok hasznosságát az állapot sorozatok hasznosságán keresztül definiáljuk. Nagyjából azt modhatjuk, hogy egy állapot hasznossága a belőle kiinduló állapot-sorozatok várható hasznosságával egyenlő. Nylvánvalóan az állapot sorozatok függnek a végrehajtott eljárásmódtól, így elsőként egy adott  $\pi$  eljárásmódra definiáljuk a hasznosságot.  $U^\pi(s)$ -t. Jelöljük  $s_t$ -vel az ágens állapotát a  $\pi$  eljárásmód  $t$  lépésnél végrehajtása után, ekkor azt kapjuk, hogy

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s \right] \quad (17.3)$$

Felhasználva ezt a definíciót, egy állapot valódi hasznossága az  $U^\pi(s)$ , amit  $U(s)$ -sel jelölünk. Ez a leszámított jutalmak várható értéke akkor, amikor az ágens egy optimális eljárásmódot hajt végre. Vegyük észre, hogy az  $U(s)$  és az  $R(s)$  igen eltérő mennyiségek: az  $R(s)$  a „rövid távú” jutalom az  $s$ -ben tartózkodásért, míg az  $U(s)$  a „hosszú távú” összjutalom  $s$ -től kezdve. A 17.3. ábrán láthatók a hasznosságok a  $4 \times 3$ -as világban. Látható, hogy a hasznosságok nagyobbak a +1-es kijárathoz közel állapotoknál, mivel kevesebb lépés szükséges a kijárat eléréséhez.

Az  $U(s)$  hasznosságfüggvény lehetővé teszi az ágensnek, hogy a cselekvéseit a 16. fejezetben szereplő maximális várható hasznosság elve alapján válassza meg; azt a cselekvést választja, amelyik maximalizálja a bekövetkező állapot várható hasznosságát:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') U(s') \quad (17.4)$$

Most azonban, ha egy állapot hasznossága a leszámított jutalmak várható értékének összege attól a kiindulóponttól kezdve, akkor közvetlen kapcsolat áll fenn egy állapot hasznossága és a szomszédainak a hasznossága között: *egy állapot hasznossága az állapotban tartózkodás közvetlen jutalmának és a következő állapot várható leszámított hasznosságának az összege, feltéve, hogy az ágens az optimális cselekvést választja.* Azaz egy állapot hasznossága a következő:



3	0,812	0,868	0,918	+ 1
2	0,762		0,660	-1
1	0,705	0,655	0,611	0,388

1      2      3      4

17.3. ábra. Az állapotok hasznosságai a  $4 \times 3$ -as világban,  $\gamma = 1$  és a nem végállapotoknál  $R(s) = -0.04$  esetén

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (17.5)$$

A (17.5) egyenletet **Bellman-egyenletnek** (**Bellman equation**) nevezik Richard Bellman tiszteletére (Bellman, 1957). Az állapotok hasznosságai – mint a következő állapot sorozatok várható hasznossága a (17.3) egyenlet szerint – a Bellman-egyenletek egy rendszerének a megoldásai. Valójában ezek egyértelmű megoldások, ahogyan a következő két alfejezetben megmutatjuk.

Nézzük meg a  $4 \times 3$ -as világ Bellman-egyenleteinek egyikét. Az (1, 1) állapothoz tartozó egyenlet:

$$\begin{aligned} U(1, 1) &= -0.04 + \gamma \max \{ && (Fel) \\ &0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1) && (Balra) \\ &0.9U(1, 1) + 0.1U(1, 2) && (Le) \\ &0.9U(1, 1) + 0.1U(2, 1) && (Jobbra) \\ &0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1) && (Jobbra) \end{aligned}$$

A 17.3. ábrán látható számok behelyettesítésével azt láthatjuk, hogy a *Fel* a legjobb cselekvés.

## Az értékiteráció algoritmus

A Bellman-egyenletek képezik az alapját az MDP-ek megoldására szolgáló értékiteráció algoritmusnak. Ha  $n$  lehetséges állapot van, akkor  $n$  Bellman-egyenlet létezik, minden egyik állapotra egy. Az  $n$  egyenlet  $n$  ismeretlenet tartalmaz – az állapotok hasznosságát. Így a hasznosságok megállapításához ezen egyenletek együttesét szeretnékn megoldani. Azonban van egy probléma: az egyenletek nemlineárisak, mivel a „max” operátor nemlineáris operátor. Míg a lineáris egyenletrendszerek hatékonyan megoldhatók lineáris algebrai eszközökkel, a nemlineáris egyenletrendszerek már problematikusabbak. Ekkor iteratív módszerrel próbálkozhatunk. Kezdéskor tetszőleges kezdeti értékeket választunk a hasznosságoknak, kiszámítjuk az egyenletek jobb oldalát, majd behelyettesítjük a bal oldalra – így frissítve az egyes állapotok hasznosságát a szomszédjainak a hasznosságával. Ezt addig ismétljük, ameddig el nem érünk

egy egyensúlyi helyzetet. Jelölje  $U_i(s)$  az  $s$  állapot hasznosságértékét az  $i$ -edik iterációban. A **Bellman-frissítésnek** (**Bellman update**) nevezett iterációs lépés a következőképpen néz ki:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \quad (17.6)$$

Ha a Bellman-frissítést végtelen sokszor alkalmazzuk, garantált, hogy egyensúlyi helyzetet érünk el (lásd következő alfejezet), és ekkor a végső hasznosságértékeknek a Bellman-egyenletek megoldását kell adniuk. Valójában ezek *egyértelmű* megoldások is, és a hozzájuk tartozó eljárásmód optimális (amit a (17.4) egyenlettel nyerhetünk). Az algoritmus, amit ÉRTÉKITERÁCIÓ-nak nevezünk, a 17.4. ábrán látható.

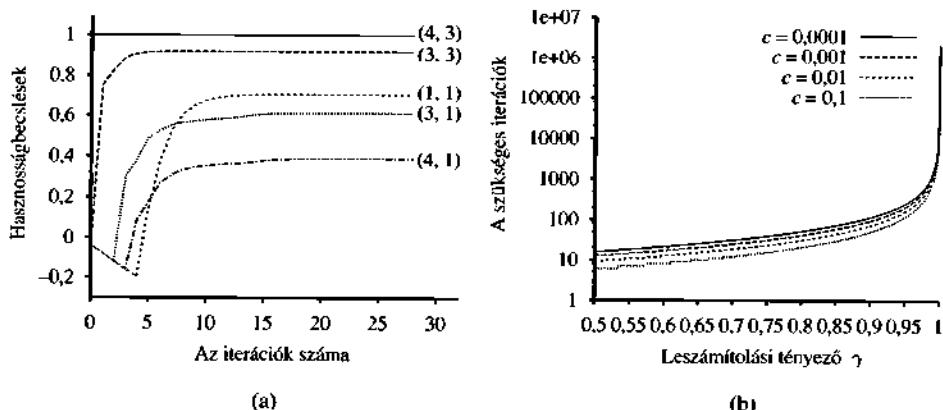
```

function ÉRTÉKITERÁCIÓ(mdf,  $\epsilon$ ) returns egy hasznosságfüggvény
  inputs: mdf, egy MDF  $S$  állapotokkal,  $T$  állapotátmenet-modellel,  $R$  jutalomfüggvénnyel,
     $\gamma$  leszámítási tényezővel
     $\epsilon$ , maximális megengedett hiba az állapotok hasznosságában
  local variables:  $U$ ,  $U'$  hasznosságértékek vektorai  $S$ -beli állapotokhoz, kezdetben nulla
     $\delta$ , a maximális változás egy állapot hasznosságában egy iteráción belül

  repeat
     $U \leftarrow U'$ ;  $\delta \leftarrow 0$ 
    for each  $s$  állapotra in  $S$  do
       $U'[s] \leftarrow R[s] + \gamma \max_{s'} T(s, a, s') U[s']$ 
      if  $|U'[s] - U[s]| > \delta$  then  $\delta \leftarrow |U'[s] - U[s]|$ 
    until  $\delta < \epsilon(1 - \gamma)/\gamma$ 
  return  $U$ 

```

17.4. ábra. Az értékiteráció algoritmus az állapotok hasznosságának kiszámítására. A leállási feltétel a (17.8) egyenletből származik.



17.5. ábra. (a) A grafikononkon kiválasztott állapotok hasznosságainak a fejlődése látható az értékiteráció felhasználásánál. (b) A szükséges értékiterációk száma ( $k$ ), hogy a hiba garantáltan legfeljebb  $\epsilon = cR_{\max}$  legyen,  $c$  különböző értékeinél, mint a  $\gamma$  leszámítási tényező függvénye.

Az értékiterációt alkalmazhatjuk a 17.1. (a) ábra  $4 \times 3$ -as világára. Nulla kezdeti értékekről indulva, a hasznosságok alakulása a 17.5. (a) ábrán látható. Vegyük észre, ahogy a (4, 3) állapottól különböző távolságra lévő állapotok negatív jutalmakat hoznak fel egészen addig, amíg egy bizonyos ponton utat találnak a (4, 3)-hoz, amikortól is a hasznosságok elkezdenek növekedni. Az értékiteráció algoritmusára gondolhatunk úgy, mint lokális frissítések általi *információterjesztésre* az állapottérben.

## Az értékiteráció konvergenciája

Azt állítottuk, hogy az értékiteráció végül a Bellman-egyenletek egy egyértelmű megoldásához konvergál. Ebben az alfejezetben kifejtjük, miért is történik ez. Ennek során bevezetünk néhány hasznos matematikai fogalmat, és olyan módszereket kapunk, amelyek megbecsülik a hasznosságfüggvény hibáját, ha az algoritmust hamar állítottuk le: ennek hasznosságát az adja, hogy nem kell végtelen hosszan futtatnunk az algoritmust. Az alfejezet a technikai részleteket is bemutatja.

Az értékiteráció konvergenciájának meghatározásában használt alapfogalom az összehúzás (**contraction**). Az összehúzás nagyjából egy olyan egy változós függvény, ami két különböző bemeneti értékre alkalmazva, két olyan kimeneti értéket ad, amelyek „egymáshoz közelebbiek”, mint az eredeti értékek, legalább egy bizonyos állandó mennyiséggel. Például a „kettővel való osztás” egy összehúzás, mivel bármely két szám elosztása után a különbségük megfeleződik. Vegyük észre, hogy a „kettővel való osztásnak” van egy fix pontja, nevezetesen a nulla, ami a függvény alkalmazásával nem változik. Ebből a példából az összehúzás két fontos tulajdonságát vehetjük észre:

- Egy összehúzásnak egyetlen fix pontja van: ha két fix pontja lenne, nem kerülnének egymáshoz közel a függvény alkalmazásakor, így nem is volna összehúzás.
- A függvény tetszőleges argumentumra történő alkalmazásakor a függvény értéknek közelebb kell kerülniük a fix ponthoz (mivel a fix pont nem mozdul el), így egy összehúzás ismételt alkalmazása a fix pontot adja határértékben.

Most tegyük fel, hogy a Bellman-frissítést (a (17.6) egyenletet) egy  $B$  operátornak tekintjük, aminek egyszeri alkalmazása az összes állapot hasznosságát frissíti. Jelölje  $U_i$  az összes állapot  $i$ -edik iterációbeli hasznosságának a vektorát. Ekkor a Bellman-frissítési egyenletet felírhatjuk úgy, hogy

$$U_{i+1} \leftarrow BU_i$$

Most egy olyan módszerre van szükségünk, amellyel a hasznosságvektorok távolságát mérhetjük. A **maximum normát** (**max norm**) fogjuk használni, ami egy vektor hosszát a legnagyobb komponensének hosszával méri:

$$\|U\| = \max_s |U(s)|$$

Ezzel a definícióval, a két vektor közötti „távolság”  $\|U - U'\|$  az összetartozó elemek közötti különbségek maximuma. Az alfejezet fő eredménye a következő: Legyen  $U_i$  és két tetszőleges hasznosságvektor. Ekkor

$$\|BU_i - BU'_i\| \leq \gamma \|U_i - U'_i\| \quad (17.7)$$



Azaz, a Bellman-frissítés egy összehúzás egy  $\gamma$  tényezővel a hasznosságvektorok terében. Így az értékiteráció mindenkor a Bellman-egyenletek egy egyértelmű megoldásához konvergál.

Nevezetesen, a (17.7) egyenletben lecserélhetjük  $U_i^*$ -t a valódi  $U$  hasznosságokkal, amire  $BU = U$ . Ekkor a következő egyenlőtlenséget kapjuk

$$||BU_i - U|| \leq \gamma ||U_i - U||$$

Ezért, ha  $||U_i - U||$ -ra mint az  $U_i$  becslés hibájára tekintünk, látjuk, hogy a hiba minden iterációban legalább egy  $\gamma$  tényezővel csökken. Ez azt jelenti, hogy az értékiteráció exponenciálisan gyorsan konvergál. A szükséges iterációk számát egy adott  $\varepsilon$  hibahatár elérésehez a következőképpen számíthatjuk ki: először, a (17.1) egyenlet szerint, az összes állapot hasznossága korlátos  $\pm R_{\max}$  ( $1 - \gamma$ ) értékkel. Ez azt jelenti, hogy a maximális kezdeti hiba  $||U_0 - U|| \leq 2R_{\max}(1 - \gamma)$ . Tegyük fel, hogy  $N$  iterációt végezzünk ahhoz, hogy a hiba legfeljebb  $\varepsilon$  értékű legyen. Mivel a hiba minden egyes alkalommal legalább  $\gamma$ -szorosára csökken, annak kell teljesülnie, hogy  $\gamma^N 2R_{\max}(1 - \gamma) \leq \varepsilon$ . Ha vesszük a két oldal logaritmusát, az adódik, hogy

$$N = \lceil \log(2R_{\max}/\varepsilon(1 - \gamma)) / \log(1/\gamma) \rceil$$

számú iteráció elegendő. A 17.5. (b) ábrán látható, hogyan változik  $\gamma$  függvényében az  $N$  különböző értékű  $\varepsilon, R_{\max}$  hányadosok esetében. A jó hír az, hogy az exponenciális gyorsaságú konvergencia miatt  $N$  nem függ túlságosan az  $\varepsilon, R_{\max}$  hányadostól. A rossz hír az, hogy  $N$  gyorsan nő, ahogy  $\gamma$  közel kerül 1-hez. Így gyors konvergenciát úgy érhettünk el, ha  $\gamma$ -t kicsivé tesszük, de ez valójában az ágensnek rövid távú horizontot biztosít, és az ágens cselekvéseinek hosszú távú hatásait figyelmen kívül hagyhatja.

Az előző bekezdés hibakorlátja ad bizonyos tájékozódást az algoritmus futási idejét befolyásoló tényezőkről, de gyakran túlságosan óvatos módszert jelent annak eldöntésére, hogy mikor álljon le az iteráció. Az utóbbi célra felhasználhatunk egy korlátot, ami a hibát az egyes iterációkban a Bellman-frissítés nagyságához kapcsolja. Az összehúzási tulajdonságból ((17.7) egyenlet) meg lehet mutatni, hogy ha a frissítés kicsi (azaz egyetlen állapot hasznossága sem változik nagyon), akkor a hiba a valódi hasznosságfüggvényhez képest szintén kicsi. Pontosabban,

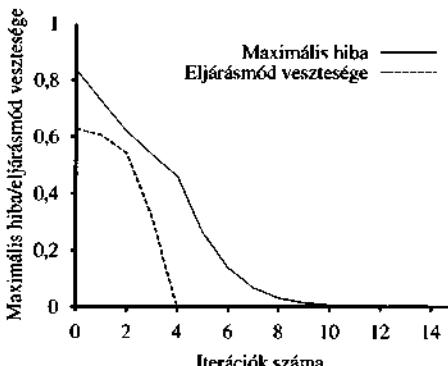
$$\text{ha } ||U_{i+1} - U|| < \varepsilon(1 - \gamma)/\gamma, \text{ akkor } ||U_{i+1} - U|| < \varepsilon \quad (17.8)$$

Ez a leállási feltétel szerepel az ÉRTÉKITERÁCIÓ algoritmusában a 17.4. ábrán.

Eddig az értékiteráció algoritmusa által visszaadott hasznosságfüggvény hibáját elemezük. Az ágenst azonban igazán az érdeklő, hogyan fog boldogulni, ha egy ilyen döntésfüggvény alapján hoz döntéseket. Tegyük fel, hogy az értékiteráció  $i$ -edik iterációja után az ágens a valódi  $U$  hasznosságra egy  $U_i$  becslést kap, és az  $U_i$ -n alapuló egylépéses előrenézés felhasználásával (ahogy a (17.4) egyenletben) megkapja a  $\pi_i$  MVH-eljárásmódját. Vajon a kiadódó viselkedés megközelítőleg lesz-e olyan jó, mint az optimális viselkedés? Ez döntő kérdés minden valós ágens számára, és bizonyítható, hogy a válasz igen. Jelölje  $U^{\pi_i}(s)$  azt a hasznosságot, ami a  $\pi_i$  végrehajtásakor adódik  $s$ -ből kiindulva. Ekkor az  $||U^{\pi_i} - U||$  az eljárásmód vesztesége (policy loss) annak a maximális értéke, amit az ágens veszíthet  $\pi_i$ -t végrehajtva az optimális  $\pi^*$  eljárásmód helyett. A  $\pi_i$  eljárásmód veszteségét az  $U_i$ -beli hibához a következő egyenlőtlenség kapcsolja:

$$\text{ha } ||U_i - U|| < \varepsilon, \text{ akkor } ||U^{\pi_i} - U_i|| < 2\varepsilon\gamma, (1 - \gamma) \quad (17.9)$$





17.6. ábra. A hasznosságbecslés maximális hibája  $\|U_i - U\|$  és az eljárásmód vesztesége  $\|U^{\pi_i} - U\|$  az optimális eljárásmódozhoz viszonyítva az értékiteráció iterációsámnak függvényében

A gyakorlatban sokszor előfordul, hogy  $\pi_i$  jóval hamarabb optimálissá válik, mielőtt  $U_i$  konvergált volna. A 17.6. ábrán látható, ahogy a  $4 \times 3$ -as környezetben a maximális  $U_i$ -beli hiba és az eljárásmód vesztesége az értékiteráció folyamatának előrehaladtával nullához közelít,  $\gamma = 0,9$  értéknél. A  $\pi_i$  eljárásmód már  $i = 4$ -nél optimális, pedig az  $U_i$ -beli maximális hiba még 0,46.

Most már minden rendelkezésünkre áll, hogy az értékiterációt a gyakorlatban is felhasználjuk. Tudjuk, hogy a helyes hasznosságokhoz konvergál, a hasznosság becsléseinek hibájára korlátokat tudunk adni, ha véges számú iteráció után leállunk, és korlátokat tudunk adni az eljárásmód veszteségére, ami a becsült hasznosságértékekhez tartozó MVH-eljárásmód végrehajtása miatt lép fel. A végső megjegyzés, hogy az összes eredmény feltétele ebben az alfejezetben a  $\gamma < 1$ -gyel való leszámítás. Ha  $\gamma = 1$  és a környezet tartalmaz végállapotokat, akkor bizonyos technikai feltételek teljesülése esetén konvergenciaeredmények és hibakorlátok hasonló halmaza származtatható.

### 17.3. ELJÁRÁSMÓD-ITERÁCIÓ

Az előző alfejezetben megfigyeltük, hogy optimális eljárásmódot akkor is kaphatunk, amikor a hasznosságfüggvény becslése pontatlan. Ha egy cselekvés egyértelműen jobb, mint a többi, akkor a releváns állapotok hasznosságainak a pontos nagyságát nem szükséges precízen tudnunk. Ez a megérzés egy alternatív módot javasol az optimális eljárásmód megkeresésére. Az **eljárásmód-iterációs (policy iteration)** algoritmus egy  $\pi_0$  kezdeti eljárásmódtól indulva a következő két lépést váltogatja:

- **Eljárásmód-értékelés (policy evaluation):** Egy adott  $\pi_i$  eljárásmódnál számítsuk ki  $U_i = U^{\pi_i}$ , az egyes állapotok hasznosságát mintha  $\pi_i$  volna végrehajtva.
- **Eljárásmód-javítás (policy improvement):** Számítsunk ki egy új  $\pi_{i+1}$  MVH-eljárásmódot, felhasználva az  $U_i$ -n alapuló egylépéses előrenézést (mint a (17.4) egyenletben).

Az algoritmus leáll, amikor az eljárásmód-javítás lépése nem eredményez változást a hasznosságokban. Ennél a pontnál tudjuk, hogy az  $U_i$  hasznosságfüggvény a Bellman-frissítés egy fix pontja, így a Bellman-egyenletek egy megoldása, és  $\pi_i$ -nek egy optimális eljárásmódnak kell lennie. Mivel csak véges sok eljárásmód létezik a véges állapotterben, és megmutatható, hogy minden iteráció jobb eljárásmódot eredményez, az eljárásmód-iterációnak le kell állnia. Az algoritmus a 17.7. ábrán látható.

Az eljárásmód-javítás lépése nyilvánvalóan egyértelmű, de hogyan valósítsuk meg az eljárásmód-értékelést. Kiderül, hogy ennek elvégzése sokkal egyszerűbb, mint a szabványos Bellman-egyenletek megoldása (amit az értékiteráció végez el), mivel az eljárásmód egy cselekvést minden állapotban rögzít. Az  $i$ -edik iterációban a  $\pi_i$  eljárásmód a  $\pi_i(s)$  cselekvést írja el. Ez azt jelenti, hogy a (17.5) Bellman-egyenlet egy egyszerűsített változatával állunk szemben, ami az  $s$  hasznosságát ( $\pi_i$  mellett) a következőképpen kapcsolja a szomszédai hasznosságához:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \quad (17.10)$$

Például tegyük fel, hogy  $\pi_i$  a 17.2. (a) ábrán látható eljárásmód. Ekkor azt kapjuk, hogy  $\pi_i(1, 1) = Fel$ ,  $\pi_i(1, 2) = Fel$  és így tovább, illetve az egyszerűsített Bellman-egyenletek:

$$U_i(1, 1) = -0,04 + 0,8U_i(1, 2) + 0,1U_i(1, 1) + 0,1U_i(2, 1)$$

$$U_i(1, 2) = -0,04 + 0,8U_i(1, 3) + 0,2U_i(1, 2)$$

$$\vdots$$

A lényeges az, hogy ezek az egyenletek lineárisak, mivel a „max” operátort eltávolítottuk. Ha az állapotok száma  $n$ ,  $n$  lineáris egyenlet adódik  $n$  ismeretlennel, ami egzakt módon megoldható  $O(n^3)$  időben standard lineáris algebrai módszerekkel.

```

function ELJÁRÁSMÓD-ITERÁCIÓ(mdf) returns egy eljárásmód
  inputs: mdf, egy MDF S állapotokkal, T állapotámenet-modellel
  local variables: U, U' hasznosságértékek vektorai S-beli állapotokhoz, kezdetben zérók
    π, egy eljárásmód vektor állapotokkal indexelve, kezdetben véletlen

  repeat
    U  $\leftarrow$  ÉRTÉKMEGHATÁROZÁS(π, U, mdf)
    változatlan?  $\leftarrow$  igaz
    for each s állapotra in S do
      if  $\max_a \sum_{s'} T(s, a, s') U[s'] > \sum_{s'} T(s, \pi[s], s') U[s']$  then
        π[s]  $\leftarrow \operatorname{argmax}_a \sum_{s'} T(s, a, s') U[s']$ 
      változatlan?  $\leftarrow$  hamis
    until változatlan?
    return π

```

17.7. ábra. Az eljárásmód-iterációs algoritmus egy optimális eljárásmód kiszámítására

Kis állapotterekre az eljárásmód-kiértékelés egzakt megoldómódszerekkel gyakran a leghatékonyabb megközelítés. Nagy állapotterekre az  $O(n^3)$  idő megengedhetetlen lehet. Szerencsére nem szükséges egzakt eljárásmód-kiértékelést végezni. Ehelyett elvégezhetünk bizonyos számú egyszerűsített értékiterációs lépést (egyszerűsített, mivel az eljárásmód rögzített), hogy a hasznosságok elfogadhatóan jó becsléséhez jussunk. Az egyszerűsített Bellman-frissítés ehhez a folyamathoz a következő

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

amit  $k$ -szor ismétlünk, hogy a következő hasznosságbecslés előálljon. Az adódó algoritmust **módosított eljárásmód-iterációnak** (**modified policy iteration**) nevezzük. Ez gyakran sokkal hatékonyabb, mint a szabványos eljárásmód-iteráció vagy értékiteráció.

Az eddig leírt algoritmusok megkövetelik a hasznosságok vagy eljárásmódok összes állapotbeli egyszerre történő frissítését. Megmutatható, hogy ez nem szigorúan szükséges. Valójában minden iterációnál kiválasztjuk az állapotok egy *tetszőleges* részhalmazát, és alkalmazhatjuk rá *bármelyik* fajta frissítést (eljárásmód-javítást vagy egyszerűsített értékiterációt). Ezt az általánosított algoritmust **aszinkron eljárásmód-iterációnak** (**asynchronous policy iteration**) nevezzük. A kezdeti eljárásmód és a hasznosságfüggvény adott feltételek mellett megválasztásánál az aszinkron eljárásmód-iteráció garantáltan konvergál az optimális eljárásmódhoz. A feldolgozandó állapotok megválasztásának szabadsága azt jelenti, hogy hatékonyabb heurisztikus algoritmusokat tervezhetünk – például olyan algoritmusokat, amelyek azon állapotok értékének frissítésére koncentrálnak, amelyeket egy jó eljárásmód valószínűleg elér. A valós életben ez nagyon hasznosnak tűnik: ha valakinek nincs szándékában a mélybe venni magát egy szikláról, akkor nem kell aggódással töltenie az időt a bekövetkező állapotok pontos értékéről.

## 17.4. RÉSZLEGESEN MEGFIGYELHETŐ MARKOV DÖNTÉSI FOLYAMATOK

A Markov döntési folyamatok leírása a 17.1. alfejezetben feltételezte, hogy a környezet **teljesen megfigyelhető** (**fully observable**). E mellett a feltételezés mellett az ágens mindig tudja, hogy melyik állapotban van. Ez együttesen az állapotátmenet-modell Markov-tulajdonságának a feltevésével azt eredményezi, hogy az optimális eljárásmód csak az aktuális állappottól függ. Amikor a környezet csak **részlegesen megfigyelhető** (**partially observable**), a helyzet nem ennyire átlátható. Az ágens nem tudja feltételelni, hogy melyik állapotban is van, így nem hajthatja végre az adott állapothoz javasolt  $\pi(s)$  cselekvést. Továbbá, egy  $s$  állapot hasznossága és az  $s$ -beli optimális cselekvés nemcsak  $s$ -től függ, hanem attól is, hogy az ágens mennyit tud, amikor  $s$ -ben van. Ebből kifolyólag a **részlegesen megfigyelhető MDF-eket (RMMDF)** (**partially observable MDPs, POMDPs**) általában sokkal nehezebb dolgoknak tartják, mint a hagyományos MDF-eket. Az RMMDF-ek azonban nem megkerülhetők, mivel a valódi világ is egy RMMDF.

Példaként gondoljuk át újra a 17.1. ábra  $4 \times 3$ -as világát, de most tegyük fel, hogy az ágensnek *egyáltalan nincsenek érzékelői*, és *elképzelése sincs* arról, hogy hol lehet. Pontosabban, tegyük fel, hogy az ágens kezdeti állapota egyenlő valószínűséggel vala-

melyike a kilenc nem végállapotnak (17.8. (a) ábra). Világos, hogz ha az ágens tudná, hogy (3, 3)-ban van, akkor *Jobbra* mozgást hajtana végre; ha *tudná*, hogy (1, 1)-ben van, akkor *Fel* mozogna; de mivel bárhol lehet, mit is kellene tennie? Egy lehetséges válasz, hogy az ágensnek először olyan cselekvéseket kell végrehajtani, amelyek csökkentik a bizonytalanságát, és csak ezután kell megpróbálni a +1 kijárat felé tartani. Például ha az ágens öt alkalommal *Balra* mozgást hajt végre, akkor igen valószínű, hogy a bal oldali falnál van (17.8. (b) ábra). Ezután öt alkalommal *Fel* mozgást hajt végre, akkor igen valószínű, hogy a felső falnál van, valószínűleg a bal felső sarokban (17.8. (c) ábra). Végül öt *Jobbra* mozgást hajt végre, ekkor jó eséllyel – körülbelül 77,5%-kal – eléri a +1 kijáratot (17.8. (d) ábra). A *Jobbra* mozgás ezt követő folytatása az esélyét 81,8%-ra növeli. Ez az eljárásmód így meglepően biztonságos, de ebben az esetben az ágens igen lassan éri el a kijáratot, és a várható hasznossága csupán 0,08 körül van. Az optimális eljárásmód, amit hamarosan leírunk, sokkal jobban teljesít.

<table border="1"> <tr><td>0,111</td><td>0,111</td><td>0,111</td><td><b>0,000</b></td></tr> <tr><td>0,111</td><td></td><td></td><td><b>0,000</b></td></tr> <tr><td>0,111</td><td>0,111</td><td><b>0,111</b></td><td><b>0,000</b></td></tr> </table>	0,111	0,111	0,111	<b>0,000</b>	0,111			<b>0,000</b>	0,111	0,111	<b>0,111</b>	<b>0,000</b>	<table border="1"> <tr><td>0,300</td><td>0,010</td><td>0,006</td><td><b>0,000</b></td></tr> <tr><td>0,221</td><td></td><td>0,059</td><td><b>0,012</b></td></tr> <tr><td>0,371</td><td>0,012</td><td>0,008</td><td><b>0,000</b></td></tr> </table>	0,300	0,010	0,006	<b>0,000</b>	0,221		0,059	<b>0,012</b>	0,371	0,012	0,008	<b>0,000</b>	<table border="1"> <tr><td>0,622</td><td>0,221</td><td>0,071</td><td><b>0,024</b></td></tr> <tr><td>0,005</td><td></td><td>0,003</td><td><b>0,022</b></td></tr> <tr><td>0,003</td><td>0,024</td><td>0,003</td><td><b>0,000</b></td></tr> </table>	0,622	0,221	0,071	<b>0,024</b>	0,005		0,003	<b>0,022</b>	0,003	0,024	0,003	<b>0,000</b>	<table border="1"> <tr><td>0,005</td><td>0,007</td><td>0,018</td><td><b>0,775</b></td></tr> <tr><td>0,034</td><td></td><td>0,007</td><td><b>0,106</b></td></tr> <tr><td>0,005</td><td>0,006</td><td>0,008</td><td>0,030</td></tr> </table>	0,005	0,007	0,018	<b>0,775</b>	0,034		0,007	<b>0,106</b>	0,005	0,006	0,008	0,030
0,111	0,111	0,111	<b>0,000</b>																																																
0,111			<b>0,000</b>																																																
0,111	0,111	<b>0,111</b>	<b>0,000</b>																																																
0,300	0,010	0,006	<b>0,000</b>																																																
0,221		0,059	<b>0,012</b>																																																
0,371	0,012	0,008	<b>0,000</b>																																																
0,622	0,221	0,071	<b>0,024</b>																																																
0,005		0,003	<b>0,022</b>																																																
0,003	0,024	0,003	<b>0,000</b>																																																
0,005	0,007	0,018	<b>0,775</b>																																																
0,034		0,007	<b>0,106</b>																																																
0,005	0,006	0,008	0,030																																																
(a)	(b)	(c)	(d)																																																

17.8. ábra. (a) Az ágens elhelyezkedésének kezdeti valószínűségi eloszlása. (b) Ötszörö *Balra* mozgás után. (c) Ötszörö *Fel* mozgás után. (d) Ötszörö *Jobbra* mozgás után

Az RMMDF-ek kezeléséhez először is megfelelően kell őket definiálni. Egy RMMDF-nek az elemei ugyanazok, mint egy MDF-nek – egy  $T(s, a, s')$  állapotámenet-modell és egy  $R(s)$  jutalomfüggvény –, de van egy  $O(s, o)$  **megfigyelési modellje (observational model)** is, ami az  $s$  állapotban az  $o$  megfigyelés érzékelésének a valószínűségét adja meg.<sup>3</sup> Például az érzékelő nélküli ágensünknek csak egyetlen megfigyelése van (az üres megfigyelés), ami minden állapotban 1 valószínűséggel bekövetkezik.

A 3. és 12. fejezetben nemdeterminisztikus és részleges megfigyelhető tervkészítési problémákat tanulmányoztunk, és megállapítottuk, hogy a **hiedelmi állapot (belief state)** – az aktuális állapotok halmaza, amelyekben az ágens lehet – kulcsfogalom a megoldások leírására és kiszámítására.

Az RMMDF-ekben a fogalmat kissé finomítjuk. Egy  $b$  hiedelmi állapot most egy valószínűség-eloszlás lesz az összes lehetséges állapot felett. Például a 17.8. (a) ábra kezdeti hiedelmi állapota úgy írható, hogy  $\left(\frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0.0\right)$ . A  $b$  hiedelmi állapot által az aktuális  $i$  állapothoz rendelt valószínűséget  $b(s)$ -sel jelöljük. Az ágens úgy számíthatja ki a jelenlegi hiedelmi állapotát, mint egy **feltételes eloszlást** az aktuális állapotok felett, az eddigi megfigyelések és cselekvések sorozatának ismeretében. Ez lényegében a **szűrési (filtering)** feladat (lásd 15. fejezet). A rekurzív szűrési alapegyenlet (lásd (15.3) egyenlet a 633. oldalon) megmutatja, hogy hogyan számolhatjuk ki az új hiedelmi állapotot

<sup>3</sup> A megfigyelési modell alapvetően azonos az időbeli folyamatok érzékelő modelljével (**sensor model**) a 15. fejezetből. Ahogyan az MDF-ek jutalomfüggvényénél, a megfigyelési modell szintén függhet a cselekvéstől és a kimeneteli állapottól, de ez most sem egy alapvető változás.

az előző hiedelmi állapotból és az új megfigyelésből. Az RMMDF-eknél a jelölés eltérő, és meg kell gondolni még a cselekvést is, de az eredmény lényegében ugyanaz. Ha  $b(s)$  volt az előző hiedelmi állapot, és az ágens az  $a$  cselekvést hajtja végre és az  $o$  megfigyelést érzékeli, akkor az új hiedelmi állapot a következő:

$$b'(s') = \alpha O(s', o) \sum_s T(s, a, s') b(s) \quad (17.11)$$

ahol  $\alpha$  egy normalizációs konstans, ami a hiedelmi állapot 1-re összegzését biztosítja. Ezt az egyenletet úgy rövidíthetjük, hogy  $b' = \text{ELŐRE}(b, a, o)$ .



A RMMDF-ek megértéséhez szükséges lényegi meglátás pedig ez: *az optimális cselekvés csak az ágens aktuális hiedelmi állapotától függ*. Így az optimális eljárásmód leírható a hiedelmi állapotok cselekvésekre történő  $\pi^*(b)$  leképzésével. Ez nem függ az ágens aktuális állapotától, amiben tartózkodik. Ez hasznos tulajdonság, mivel az ágens nem ismeri az aktuális állapotát; csupán a hiedelmi állapotát. Ekkor egy RMMDF ágens döntési ciklusa a következő:

1. Adott jelenlegi  $b$  hiedelmi állapot esetén hajtsuk végre az  $a = \pi^*(b)$  cselekvést.
2. Fogadjuk az  $o$  megfigyelést.
3. Állítsuk a jelenlegi állapothiedelmet  $\text{ELŐRE}(b, a, o)$ -ra, és ismételjük meg a ciklust.

Most úgy gondolhatunk az RMMDF-ekre, mint amelyek keresést igényelnek a hiedelmi állapotok terében, pontosan úgy, ahogy az érzékelő nélküli és az eshetőségi problémákra vonatkozó módszerek a 3. fejezetben. A fő különbség az, hogy az RMMDF hiedelmi állapot tere *folytonos*, mivel egy RMMDF hiedelmi állapot egy valószínűségi eloszlás. Például a hiedelmi állapot a  $4 \times 3$ -as világ esetében egy pont a 11 dimenziós folytonos térből. Egy cselekvés megváltoztatja a hiedelmi állapotot is, nemcsak a fizikai állapotot, így kiértékelése aszerint az információ szerint törtenik, amit a cselekvés eredményeként az ágens megszerez. Az RMMDF-ekbe ezért beletartozik az információ értéke is (16.6. alfejezet) mint a döntési probléma egy komponense.

Nezzük meg tüzetesebben a cselekvések kimenetelét. Nevezetesen, számítsuk ki annak valószínűségét, hogy az ágens a  $b$  hiedelmi állapotból a  $b'$  hiedelmi állapotba jut az  $a$  cselekvés végrehajtása után. Ekkor, ha ismernénk a cselekvést és a bekövetkező megfigyelést, akkor a (17.11) egyenlet a hiedelmi állapot egy determinisztikus frissítését adná:  $b' = \text{ELŐRE}(b, a, o)$ . Természetesen a bekövetkező megfigyelés még nem ismert, így az ágens a számos lehetséges hiedelmi állapot egyikébe,  $b'$ -be kerülhet az előforduló megfigyelés függvényében. Az  $o$  érzékelésének valószínűsége, feltéve hogy a  $b'$  hiedelmi állapotban az  $a$  cselekvés került végrehajtásra, az összes olyan  $s'$  aktuális állapot feletti összegzéssel adódik, amelyeket az ágens elérhet:

$$\begin{aligned} P(o|a, b) &= \sum_{s'} P(o|a, s', b) P(s'|a, b) \\ &= \sum_{s'} O(s', o) P(s'|a, b) \\ &= \sum_{s'} O(s', o) \sum_s T(s, a, s') b(s) \end{aligned}$$

Jelölje  $\tau(b, a, b')$  annak a valószínűségét, hogy  $b$ -ből  $b'$ -be jutunk egy  $a$  cselekvéssel. Ennek felhasználásával ekkor

$$\begin{aligned}\tau(b, a, b') &= P(b'|a, b) = \sum_o P(b'|o, a, b)P(o|a, b) \\ &= \sum_o P(b'|o, a, b) \sum_{s'} O(s', o) \sum_s T(s, a, s')b(s)\end{aligned}\quad (17.12)$$

ahol  $P(b'|o, a, b) 1$ , ha  $b' = \text{Előre}(b, a, o)$ , egyébként pedig 0.

A (17.2.) egyenletet felfoghatjuk egy állapotátmenet-modellnek a hiedelmi állapotban. Definiálhatunk egy jutalomfüggvényt is a hiedelmi állapotokra (azaz azon aktuális állapotok várható jutalmát, amelyekben az ágens lehet):

$$\rho(b) = \sum_s b(s)R(s)$$

Így láthatóan  $\tau(b, a, b')$  és  $\rho(b)$  együtt egy megfigyelhető MDF-et definiál a hiedelmi állapotok terén. Továbbá megmutatható, hogy egy  $\pi^*(b)$  optimális eljárásmód erre az MDF-re, szintén optimális eljárásmód az eredeti RMMDF-re. Máshogy fogalmazva, egy RMMDF megoldása a fizikai állapotterén redukálható egy MDF megoldására a hozzá tartozó hiedelmi állapotban. Ez a tény talán kevésbé meglepő, arra gondolva, hogy a hiedelmi állapot definíció szerint az ágens számára minden megfigyelhető.

Vegyük észre, hogy bár az RMMDF-eket sikeresen redukáltuk az MDF-ekre, a kapott MDF-eknek folytonos (és általában sokdimenziós) állapotterük van. A 17.2. és 17.3. alfejezetekben leírt algoritmusok egyike sem alkalmazható közvetlenül ilyen MDF-ekre. Kiderült, hogy az érték- és eljárásmód-iterációt kifejleszthetők olyan változatai, amelyek folytonos állapotú MDF-ekre is alkalmazhatók. Az alapötlet az, hogy egy  $\pi(b)$  eljárásmód reprezentálható a hiedelmi állapot tér olyan részeinek egy halmazaként, amelyek mindegyikéhez egy adott optimális cselekvés tartozik.<sup>4</sup> Az értékfüggvény  $b$  különböző lineáris függvényét rendeli az egyes térrészekhez. minden érték- és eljárásmód-iterációs lépés finomít a térrészek határain, és új térrészeket vezethet be.

Az algoritmus részletei meghaladják a könyv kereteit, de megadjuk a megoldást az érzékelőmentes  $4 \times 3$ -as világra. Az optimális eljárásmód a következő:

[Balra, Fel, Fel, Jobbra, Fel, Fel, Jobbra, Fel, Fel, Jobbra, Fel, Jobbra, Fel, ...]

Az eljárásmód egy sorozat, mivel ez a probléma determinisztikus a hiedelmi állapotok terében – nincsenek megfigyelések. Azt a „trükköt” alkalmaztuk, hogy az ágens egyetlen Balra mozgással biztosítja, hogy nincs (4, 1)-ben, azért, hogy aztán meg lehetős biztonsággal mozoghasson Fel és Jobbra a +1 kijárat eléréséhez. Az ágens a +1 kijáratot 86,6% gyakorisággal éri el, és sokkal gyorsabban, mint az alfejezet elején megadott eljárásmód esetén, így a várható hasznossága 0,38, szemben az előző 0,08-as értékével.

Bonyolultabb RMMDF-eknél, amikor megfigyelések is vannak, nagyon nehéz (valójában PSPACE-beli – azaz valóban nagyon nehéz) közelítőleg optimális eljárásmódokat találni. A problémák néhány tucat változóval gyakran kivitelezhetetlenek. A következő alfejezet egy előző közelítő módszert ír le RMMDF-ek megoldására, ami az előrenéző keresésen alapul.

<sup>4</sup> Bizonyos RMMDF-ekre az optimális eljárásmóhoz végtelen számú térrész tartozik, így az egyszerű térrészek listája megközelítés nem működik, és ötletesebb módszerek szükségesek még egy közelítés megtalálásához is.

## 17.5. DÖNTÉSELMÉLETI ÁGENSEK

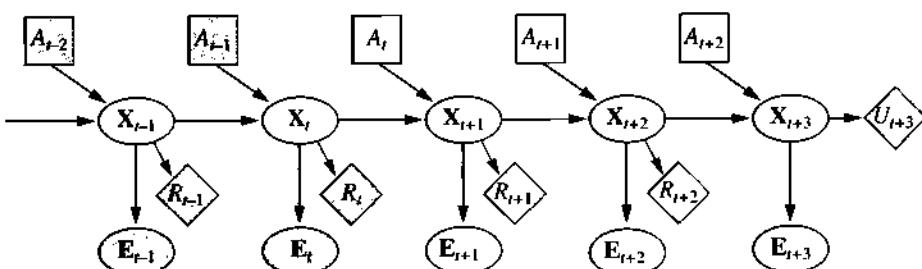
Ebben a fejezetben egy átfogó szemléletmódot vázolunk fel az ágensek megtervezésére részlegesen megfigyelhető, sztochasztikus környezetek esetén. A tervezés alapelemei már ismertek:

- Az állapotátmenet- és megfigyelési modellek egy **dinamikus Bayes-hálóval** (**dynamic Bayesian network**) ábrázoljuk (a 15. fejezetben leírtak szerint).
- A dinamikus Bayes-háló kiegészült döntési és hasznossági csomópontokkal, amelyek megegyeznek a 16. fejezetben szereplő döntési hálókban használatosakkal. A kirajzolódó modellt **dinamikus döntési hálónak** vagy **DDH-nak** (**dynamic decision network, DDN**) nevezük.
- Egy szűrési algoritmust használunk az új érzékelések és cselekvések befogadására és a hiedelmi állapot reprezentációjának a frissítésére.
- Döntéseket hozunk lehetséges cselekvéssorozatok elgondolásával és a legjobb kiválasztásával.

Egy dinamikus Bayes-háló felhasználásának elsődleges haszna az állapotátmenet- és érzékelő modellek ábrázolásánál az, hogy felbontja az állapotleírást valószínűségi változók egy halmazára, hasonlóan ahhoz, ahogy a tervkészítési algoritmusok logikai ábrázolást vesznek igénybe az állappotter felbontására, amit a keresési algoritmusok használnak. Az ágens-terv ezért a 2. fejezetben vázolt hasznossággalapú ágens egy gyakorlati megvalósítása.

Mivel dinamikus Bayes-hálókat használunk, visszatérünk a 15. fejezetbeli jelöléshez, ahol  $X_t$  az állapotváltozók egy halmazát jelölte egy  $t$  időpillanatban,  $E_t$  pedig a bonyítéktékváltozókat. Így ahol ebben a fejezetben eddig  $s_t$ -t használtunk (a  $t$ -beli állapotra), mostantól  $X_t$ -t fogunk használni. A  $t$ -beli cselekvést  $A_t$ -vel fogjuk jelölni, így a  $T(s, a, s')$  állapotátmenet-modell ugyanaz, mint a  $P(X_{t+1}|X_t, A_t)$ , és az  $O(s, o)$  megfigyelési modell ugyanaz, mint a  $P(E_t|X_t)$ . A  $t$  időpontban kapott jutalmat jelöljük  $R_t$ -vel, és  $U_t$ -vel az állapot hasznosságát a  $t$  időpontban. E szerint a jelölés szerint egy dinamikus döntési háló a 17.9. ábrán látható módon néz ki.

A dinamikus döntési hálók tömör leírást biztosítanak nagy RMMDF-ekre, így bár-mely RMMDF algoritmus bemenetéül szolgálhatnak, ideértve az érték- és eljárásmód-



**17.9. ábra.** Egy dinamikus döntési háló általános struktúrája. Az ismert értékű változók árnyékoltak. A jelenlegi időpont  $t$ , és az ágensnek el kell döntenie, hogy mit tegyen – azaz választania kell egy értéket  $A_t$  számára. A háló három jövőbeli lépésre van kibontva, és a jövőbeli jutalmakat, valamint az előrelátó horizonton lévő állapot hasznosságát ábrázolja.

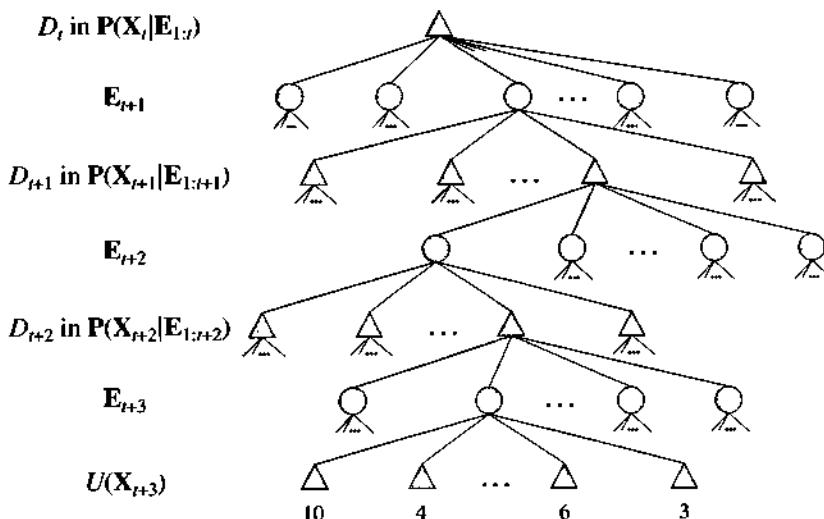
iterációra való módszereket is. Ebben az alfejezetben az előrenéző módszerekre összpontosítunk, amelyek cselekvéssorozatokat képzelnak el a jelenlegi hiedelmi állapotból kiindulva, nagyon hasonlóan a 6. fejezetbeli játékok algoritmusaihoz. A 17.9. ábrán a háló három jövőbeli lépésig elképzelve szerepel; a jelenlegi és a jövőbeli döntések és a jövőbeli megfigyelések és jutalmak mind ismeretlenek. Figyeljük meg, hogy a háló csomópontokat tartalmaz az  $X_{t+1}$  és  $X_{t+2}$ -beli *jutalmakra*, de *hasznosságot* az  $X_{t+3}$  esetében. Ez azért van így, mert az ágensnek maximalizálnia kell az összes jövőbeli jutalom (leszámított) összegét, és  $U(X_{t+3})$  reprezentálja az  $X_{t+3}$ -hoz tartozó és az összes ezt követő jutalmat. Ahogyan a 6. fejezetben, most is feltesszük, hogy  $U$ -nak csak egy közelítő alakja érhető el: ha elérhetők lennének pontos hasznosságértékek, akkor nem lenne szükség 1 lépésnél hosszabb előretekintésre.

A 17.9. ábrán látható DDH-hoz tartozó háromlépéses előrenéző keresési fa egy részét a 17.10. ábra mutatja. Mindegyik háromszög alakú csomópont egy olyan hiedelmi állapot, amiben az ágens egy  $A_{t+i}$  döntést hoz  $i = 0, 1, 2, \dots$  esetén. A kerek csomópontok a környezet válaszaihoz tartoznak, nevezetesen hogy minden  $E_{t+i}$  megfigyelés fog felbukkanni. Látható, hogy nincsen véletlen csomópont, ami a cselekvések kimeneteléhez tartozna. Ez azért van így, mert egy cselekvésnél a hiedelmi állapot frissítése a konkrét kimeneteltől függetlenül determinisztikus.

A hiedelmi állapot az egyes háromszög csomópontoknál kiszámítható egy szűrési algoritmusnak a hozzávezető megfigyelések és cselekvések sorozatán való alkalmazásával. Ezzel a módszerrel, az algoritmus figyelembe veszi azt a tényt, hogy egy  $A_{t+i}$  döntésnél az ágens számára elérhetők lesznek az  $E_{t+1}, \dots, E_{t+i}$  érzékelések, még ha a  $t$  időpontban nem is tudja, hogy ezek az érzékelések mi is lesznek. Ezen a módon a döntéselméleti ágens automatikusan figyelembe veszi az információ értékét és információgyűjtő cselekvéseket hajt végre, ha szükséges.

Egy döntés a keresési fából a levelek hasznosságértékének hátrafelé terjesztésével nyerhető ki, a véletlen csomópontknál átlagot, a döntési csomópontokban pedig maximumot számítva. Ez hasonló a véletlen csomópontokkal rendelkező játékfákra vonatkozó VÁRHATÓMINIMAX algoritmushoz, kivéve, hogy (1) nem csak levélállapotokban lehet jutalom, és (2) a döntési csomópontok hiedelmi állapotokhoz, és nem aktuális állapotokhoz tartoznak. A  $d$  mélységű kimerítő keresés időkomplexitása  $O(|D|^d \cdot |E|^d)$ , ahol  $|D|$  az elérhető cselekvések száma,  $|E|$  pedig a lehetséges megfigyelések száma. Olyan problémáknál, ahol a  $\gamma$  leszámítási tényező nincs közel 1-hez, gyakran elelegendő sekély keresés az optimálishez közeli döntés megtalálásához. Az is lehetséges, hogy az átlagszámítás lépését a lehetséges megfigyelések mintavételezésével közelítjük, az összes lehetséges megfigyelés feletti összegzés helyett. Számos más módja van jó közelítő megoldások gyors megtalálásának, de ezeket a 21. fejezetre hagyjuk.

A dinamikus döntési háló alapú döntéselméleti ágenseknek számos előnye van más, egyszerűbb, a korábbi fejezetekben bemutatott ágensfelépítésekhez képest. Nevezetesen, részlegesen megfigyelhető és sztochasztikus környezeteket kezelnek, és könnyen felülbírálják a „terveiket” a váratlan események kezeléséhez. Megfelelő érzékelő modellekkel képesek kezelni az érzékelőmeghibásodást, és képesek megtervezni az információ begyűjtését. Az idő szorításában és komplex környezetekben „teljesítményromlásuk fokozatos”, különöző közelítő technikák kihasználásával. Így jogos a kérdezés, hogy mi is hiányzik még? A DDH-alapú algoritmusunknak a legkomolyabb hiányossága az előrefelé kereséstől való függés, ahogyan a II. részbeli állapottér-keresési



17.10. ábra. A 17.9. ábrán látható DDH előrenéző megoldásának egy része

algoritmusoknak is. A IV. részben elmagyaráztuk, hogy részlegesen rendezett, absztrakt tervez mérlegelésének képessége célirányos kereséssel hogyan biztosította a probléma-megoldó képesség lényegi növekedését, különösen a tervkönyvtárakkal összeilleszte. Ezeket a módszereket megkíséreltek a valóságossági területre is kiterjeszteni, de ez eddig nem bizonyult hatékonynak. Egy második, kapcsolódó probléma a DDH nyelv alapvető kijelentéslogikai természete. Szeretnénk, ha az elsőrendű valóságossági nyelvek alképzéseit (lásd 14.6. alfejezet) ki tudnánk terjeszteni a döntéshozatal problémájára. A jelenlegi kutatások azt mutatják, hogy ez a kiterjesztés lehetséges, és jelentős előnyöket rejt magában, ahogy ezt a fejezet végi megjegyzéknél megtárgyaljuk.

## 17.6. TÖBBÁGENSES DÖNTÉSEK: A JÁTÉKELMÉLET

Ez a fejezet a bizonytalan környezetbeli döntéshozatalra összpontosított. De mi a helyzet akkor, ha a bizonytalanság más ágenseknek és azok döntéseinek köszönhető? És ha ezeknek az ágenseknek a döntéseit a mi döntéseink befolyásolják? Ezzel a kérdéssel egyszer már foglalkoztunk, amikor a 6. fejezetben a játékokat tanulmányoztuk. Ott azonban többfordulós, teljes információs játékokkal foglalkoztunk, amelyekre a minimax keréséssel optimális lépések találhatók. Ebben az alfejezetben a **játékelmélét** (**game theory**) azon vonatkozásait tanulmányozzuk, amelyek felhasználhatók szimultán játékok elemzésére. Egyszerűsítés céljából elsőként olyan játékokat nézünk meg, amelyek csak egy lépés hosszúak. A „játék” szó és az egyetlen lépésre való korlátozás ezt triviálisan tüntetheti fel, de valójában a játékelmélét nagyon komoly döntéshozatali helyzetekben használják – ideértve a bankcsődeljárásokat, a távközlési frekvenciasávok aukcióját, a termékfejlesztési és ármegállapítási döntésekét, illetve a honvédelmi kérdéseket – olyan helyzetekben, amelyek dollármilliárdok és életek százezreinek a sorsát érintik. A játékelmélét legalább kétféle módon használható fel:

- Ágenstervezés (agent design):** A játékelmélet képes elemezni az ágensek döntéseit, és kiszámítani az egyes döntések várható hasznosságát (azon feltevés mellett, hogy más ágensek a játékelmélet szerint optimálisan cselekednek). Például a kétujjas snóblijá-tékban (**two-finger Morra**) két játékos,  $O$  és  $E$  egyidejűleg felmutatja egy vagy két ujját. Legyen az ujjak összes száma  $f$ . Ha  $f$  páratlan, akkor  $O$  kap  $f$  dollárt  $E$ -től, és ha  $f$  páros, akkor  $E$  kap  $f$  dollárt  $O$ -tól. A játékelmélet képes meghatározni a legjobb stratégiát egy racionális ellenféllel szemben, és az egyes játékosok várható hasznát.<sup>5</sup>
- Működési mód tervezés (mechanism design):** Amikor a környezetet számos ágens népesíti be, előfordulhat, hogy a környezet szabályait (azaz a játékot, amit az ágenseknek játszaniuk kell) olyannak határozzák meg, hogy az összes ágens együttes haszná akkor maximális, amikor minden egyik ágens egy saját hasznát maximáló játékelméleti megoldást követ. Például a játékelmélet segíthet olyan protokollok tervezésében az internetes forgalomirányítók számára, hogy minden irányítónak olyan ösztönzői legyenek, hogy a globális átvitel maximális legyen. A működési mód tervezés arra is felhasználható, hogy olyan intelligens többágenses rendszereket (**multiagent systems**) hozunk létre, amelyek komplex problémákat elosztott módon oldanak meg, anélkül, hogy az egyes ágenseknek ismerniük kellene a megoldandó teljes problémát.

Egy játékot a játékelméletben a következő alkotóelemek definiálnak:

- Játékosok (player)** vagy ágensek, akik döntéseket hoznak. A legnagyobb figyelem a kétszemélyes játékokra irányult, bár az  $n$  személyes játékok  $n > 2$ -re szintén gyakoriak. A játékosokra nagybetűs neveket használunk, mint Aliz és Bendegúz vagy  $O$  és  $E$ .
- Cselekvések (actions)**, amiket a játékosok választhatnak. A cselekvéseknek kisbetűs neveket választunk, mint az *egy* vagy a *tanúskodik*. Lehet, hogy a játékosok a cselekvéseknek ugyanazt a halmozatot érik el, de az is lehet, hogy különbözött.
- Jutalommatrix (payoff matrix)**, ami az összes játékos cselekvéseinek minden egyik kombinációjára, minden játékos számára megadja a hasznosságot. A jutalommatrix a kétujjas snóblijá-tékhoz a következő:

	$O: \text{egy}$	$O: \text{kettő}$
$E: \text{egy}$	$E = 2, O = -2$	$E = -3, O = 3$
$E: \text{kettő}$	$E = -3, O = 3$	$E = 4, O = -4$

Például a jobb alsó sarokban az látható, amikor  $O$  a kettő cselekvést és  $E$  szintén a kettő cselekvést választja, a jutalom  $E$ -nek 4,  $O$ -nak pedig -4.

Minden játékosnak a játékban választania kell, és aztán végrehajtani egy **stratégia** (**strategy**) (ami a 16. fejezetben szereplő eljárásmód elnevezése a játékelméletben). A **tiszta stratégia (pure strategy)** egy determinisztikus eljárásmód, ami minden helyzethez előír egy végrehajtandó konkrét cselekvést; az egylépéses játékokban a tiszta

<sup>5</sup> A snóblijáték az **ellenőrzési játék** (*inspection game*) egy szabadidős változata. Ilyen játékokban az ellenőr megválaszt egy napot egy létesítmény (mint például egy vendéglő vagy egy biológiai fegyvergyár) ellenőrzésére, és a létesítmény működtetője szintén megválaszt egy napot, amikor az összes kellemetlen dolgot elrejt. Az ellenőr győz, ha a napok különböznek, és a létesítmény fenntartója, ha ugyanazok.

stratégia csupán egyetlen cselekvés. A játékok elemzése vezetett el a kevert stratégiaiak (*mixed strategy*) gondolatához, amely eljárásmódban egy konkrét cselekvés véletlenszerűen kerül kiválasztásra egy adott eloszlásból a cselekvések felett. Azt a kevert stratégiát, ami  $p$  valószínűséggel az a cselekvést, egyébként pedig a  $b$ -t választja, úgy jelöljük, hogy  $[p: a; (1 - p): b]$ . Például a kétujjas snóbjátékban egy lehetséges kevert stratégia a  $[0,5: \text{egy}; 0,5: \text{kettő}]$ . A **stratégiaprofil** (*strategy profil*) egy hozzárendelés, ami minden játékoshoz egy stratégiát rendel; adott stratégiaprofilnál a játék **kimenetele** (*outcome*) egy numerikus érték minden játékoshoz.

Egy játék **megoldása** (*solution*) egy olyan stratégiaprofil, amiben minden játékos egy racionális stratégiát fogad el. Látni fogjuk, hogy a játékelméletben a legfontosabb kérdés a „racionális” jelentésének a definiálása akkor, amikor minden játékos csak egy részét választja meg a kimenetelt meghatározó stratégiaprofilnak. Fontos azt felismerni, hogy a kimenetelek a játék egyes meneteinek a konkrét eredményei, míg a megoldások a játék elemzéséhez használt elméleti konstrukciók. Megmutatjuk, hogy bizonyos játékoknak csak a kevert stratégiák között van megoldása. Ez azonban nem jelenti azt, hogy egy játékosnak szó szerint egy kevert stratégiát kell elfogadnia a racionális viselkedéshez.

Gondoljuk át a következő történetet: két állítólagos betörőt, Alizt és Bendegúzt tetten érik egy betörés helyszínéhez közel, és a rendőrség külön-külön kihallgatja őket. Mindketten tudják, hogy ha mindenketten beismérlik a bűncselekményt, akkor 5 év börtönt kapnak betörésért, de ha mindenketten tagadnak, akkor csak 1 évet kapnak. Ilopott holmik birtoklásának enyhébb vádjával. Azonban a rendőrség külön-külön alkut ajánl mindenkettónek: ha a partneredről tanúsítod, hogy egy betörőbanda vezetője, akkor szabadon engednek, míg a partnered 10 évet kap. Ekkor Aliz és Bendegúz az úgynevezett **fogolydilemmával** (*prisoner's dilemma*) szembesülnek: tanúskodniuk kell vagy tagadniuk? Racionális ágensként Aliz és Bendegúz is a saját várható hasznosságát akarja maximálni. Tegyük fel, hogy Aliz teljességgel hidegen hagyja partnerének a sorsa, így az ő hasznossága a saját börtönben töltött éveinek számával arányosan csökken, függetlenül attól, hogy mi történik Bendegúzzal. Bendegúz teljesen hasonlóan érez. A racionális döntés meghozatalához mindenketten megalkotják a következő jutalommatrixot:

	<i>Aliz:tanúskodik</i>	<i>Aliz:tagad</i>
<i>Bendegúz:tanúskodik</i>	$A = -5, B = -5$	$A = -10, B = 0$
<i>Bendegúz:tagad</i>	$A = 0, B = -10$	$A = -1, B = -1$

Aliz a következőképpen elemzi a jutalommatrixot: Tegyük fel, hogy Bendegúz tanúskodik. Ha tanúskodom, 5 évet kapok, és 10 évet, ha nem, ebben az esetben a tanúskodás jobb. Másrésztől, ha Bendegúz tagad, akkor 0 évet kapok, ha tanúskodom, és 1 évet, ha tagadok, így a tanúskodás ebben az esetben is jobb. Így minden esetben számonra jobb tanúskodni, így ez az, amit tennem kell.

Aliz felfedezte, hogy a **tanúskodás** a játék **domináns stratégiája** (*dominant strategy*). Azt mondjuk, hogy egy  $p$  játékos egy  $s$  stratégiája **erősen dominálja** (*strongly dominate*) az  $s'$  stratégiát, ha az  $s$  kimenetele  $p$  számára jobb, mint az  $s'$  kimenetele, a többi játékos minden stratégiaválasztása esetén. Egy  $s$  stratégia **gyengén dominálja**

(weakly dominates)  $s'$ -t, ha  $s$  jobb, mint  $s'$  legalább egy stratégiaprofilnál, és nem rosszabb a többiben. Egy domináns stratégia olyan stratégia, ami az összes többit dominálja. Egy erősen dominált stratégiát irrationális követni, és ha létezik, irrationális eltérni a domináns stratégiától. Racionális volta miatt Aliz a domináns stratégiát választja. Már csak egy kevés terminológiára van szükségünk a továbbhaladáshoz: egy kimenetet **Pareto-optimálisnak** (**Pareto optimal**) nevezünk,<sup>6</sup> ha nincs másik kimenetel, amit az összes játékos preferálna. Egy kimenetet **Pareto-dominált** (**Pareto dominated**) egy másik kimenetel által, ha az összes játékos a másik kimenetelt preferálná.

Ha Aliz okos és racionális, folytatni fogja az érvelést a következőképpen: Bendegúz domináns stratégiája szintén a tanúskodás. Így ő is tanúskodni fog, és mindenkiten 5 évet kapunk. Amikor mindegyik játékosnak van domináns stratégiája, ezek kombinációját **domináns stratégiai egyensúlynak** (**dominant strategy equilibrium**) nevezzük. Általában egy stratégiaprofil akkor van **egyensúlyi helyzetben** (**equilibrium**), ha egyik játékos sem nyer stratégiája váltással, az összes többi játékos stratégiájának változatlansága mellett. Egy egyensúlyi helyzet lényegében egy **lokális optimum** (**local optimum**) az eljárásmódok terében: egy csúcs teteje lejtőkkel minden dimenzióban, ahol a dimenziók egy játékos stratégiai választásaihoz tartoznak.

A **dilemma** a fogolydilemmában az, hogy az egyensúlyi helyzet kimenetele minden játékosnak rosszabb, mint az a kimenetel, amit akkor érnének el, ha mindenkiten megtagadnák a tanúskodást. Másképpen fogalmazva, az egyensúlyi megoldás kimenete Pareto-dominált a (*tagad, tagad*)-nak a (-1, -1)-es kimenetelével.

Van-e módja Aliznak és Bendegúznak a (-1, -1)-es kimenetelt elérni? Az bizonyosan **megengedett lehetőség** mindenkitőjüknek, hogy megtagadják a tanúskodást, de **valószínűleg** lehetőség. Akármelyik játékos is kezdene elmélkedni azon, hogy a *tagadást* választja, felismeri, hogy jobban teszi, ha a *tanúskodik*. Az egyensúlyi helyzetnek ez a vonzereje.

A matematikus John Nash (1928-) bebizonyította, hogy *minden játéknak van egy itt definiált típusú egyensúlyi helyzete*. Ezt a tiszteletre most már Nash-egyensúlynak nevezik. Nyilvánvaló, hogy egy domináns stratégiai egyensúly **Nash-egyensúly** (**Nash equilibrium**) (lásd 17.9. feladat), de nem minden játéknak vannak domináns stratégiái. Nash tétele azt mondja ki, hogy egyensúlyi stratégiák akkor is léteznek, ha nincs domináns stratégia.

A fogolydilemmánál csak a (*tanúskodik, tanúskodik*) stratégiaprofil Nash-egyensúlyi. Nehezen látható, hogy racionális játékosok hogyan tudják elkerülni ennek a kimenetelét, mivel bármely javasolt nem egyensúlyi megoldásnál legalább egy játékos kísértésben lenne, hogy változtasson a stratégiáján. Játékelméleti kutatók egyetértenek, hogy a Nash-egyensúly fennállása szükséges feltétel egy megoldás fennállásához – bár abban már nem értenek egyet, hogy ez elégsges feltétel-e.

A (*tanúskodik, tanúskodik*) megoldás könnyűszerrel elkerülhető, ha valamilyen módon megváltoztatjuk a játékot (vagy a játékosokat). Módosíthatjuk például egy olyan iterált játékra, amiben a játékosok tudják, hogy újra találkozni fognak (de az döntő, hogy bizonytalannak legyenek abban, hogy hányszor fognak még újra találkozni). Vagy ha az ágensek morális meggyőződése erősíti az együttműködést és az igazságosságot, megváltoztathatjuk a jutalommatrixot úgy, hogy tükrözze az egyes ágensek többiekkel való

<sup>6</sup> A Pareto-optimalitás a közgazdász Vilfredo Paretóról (1848–1923) kapta nevét.

együttműködésének a hasznosságát. Később látni fogjuk, hogy az ágens olyatén megváltoztatása, hogy a számítási kapacitása korlátos, ahelyett hogy teljesen racionális következtetésre lenne képes, szintén befolyásolja a kimenetelt, mivel azt árulhatja el az egyik ágensnek, hogy a másik racionálitása korlátos.

Most nézzünk meg egy játékot, aminek nincs domináns stratégiája. Az Acme nevű videójáték-hardver gyártójának döntenie kell, hogy a következő játékgép DVD-ket vagy CD-ket használjon. Eközben a videójáték-szoftver gyártójának, Bestnek is döntenie kell, hogy DVD-n vagy CD-n adja ki a következő játékát. A nyereség mindenkiüknek pozitív, ha egyetértenek, és negatív, ha eltérnek, ahogy a következő jutalommatrixból ez látható:

	<i>Acme:dvd</i>	<i>Acme:cd</i>
<i>Best:dvd</i>	$A = 9, B = 9$	$A = -4, B = -1$
<i>Best:cd</i>	$A = -3, B = -1$	$A = 5, B = 5$

Erre a játékra nincs domináns stratégiai egyensúly, de létezik két Nash-egyensúly: (*dvd, dvd*) és (*cd, cd*). Tudjuk, hogy ezek Nash-egyensúlyok, mivel akármelyik játékos is vált egyoldalúan egy különböző stratégiára, az rosszabb fog állni. Most az ágenseknek a problémája a következő: *több elfogadható megoldás is létezik, de ha az egyes ágensek különböző megoldást választanak, akkor a kiadódó stratégiaprofil egyáltalán nem lesz megoldás, és minden ágens veszteséget fog elszennedni*. Hogyan egyezhetnek meg egy megoldásban? Egy válasz erre, hogy minden ágensek a Pareto-optimális megoldást kell választania (*dvd, dvd*); azaz leszűkíthetjük a „megoldás” definícióját az egyértelmű Pareto-optimális Nash-egyensúlyokra, feltéve, hogy létezik ilyen. minden játéknak van legalább egy Pareto-optimális megoldása, de egy játéknak lehet több egyensúlyi pontja, de lehet, hogy ezek nem egyensúlyi pontok. Például beállíthatjuk a jutalmakat a (*dvd, dvd*)-nél 5-re 9 helyet. Ebben az esetben két egyenlő Pareto-optimális egyensúlyi pont létezik. A közöttük való választáshoz az ágensek vagy találhatnak, vagy kommunikálnak, ami vagy egy megállapodás kimondását jelenti, ami sorrendezi a megoldásokat a játék megkezdődése előtt, vagy tárgyalást jelent, hogy egy kölcsönösen előnyös megoldás alakuljon ki a játék folyamán (ami egy többlepéses játék részeként megjelenő kommunikációs cselekvések létével jelentené). A kommunikáció így pontosan ugyanazokból az okokból jelenik meg, mint a többágenses tervezésnél a 12. fejezetben. Az ilyen játékokat, ahol a játékosoknak szükséges kommunikálniuk, koordinációs játékoknak (coordination games) nevezzük.

Láttuk, hogy egy játéknak lehet több mint egy Nash-egyensúlya: honnan tudjuk, hogy minden játéknak van legalább egy? Az lehetséges, hogy egy játéknak nincs tiszta stratégiájú Nash-egyensúlya. Gondoljunk át például egy tetszőleges tiszta stratégiájú profilt a kétujjas snóbijátékhöz (727. oldal). Ha az ujjak összértéke páros, akkor *O* szeretné váltni; ha páratlan, akkor pedig *E* szeretné váltni. Így egy tiszta stratégiájú profil nem lehet egyensúlyi, és kevert stratégiákat kell megnéznünk.

De melyik kevert stratégiát? 1928-ban Neumann kifejlesztett egy módszert az optimális kevert stratégia megkeresésére kétszemélyes zérusösszegű játékokra. A zérusösszegű

játék<sup>7</sup> (**zero-sum games**) olyan játék, amiben a jutalmak a jutalom mátrix elemeiben nulla összegződnek. Világos, hogy a snóblijáték ilyen játék. A kétszemélyes zérusösszegű játékoknál tudjuk, hogy a jutalmak egyenlők és ellentétesek, így csak az egyik játékos jutalmait kell figyelembe vennünk, aki a maximáló lesz (ahogy a 6. fejezetben is). A snóblijátknál az  $E$  páros játékos választjuk maximálónak, így a jutalom mátrixot az  $U_E(e, o)$  értékekkel definiálhatjuk – a jutalom  $E$ -nek, ha  $E$   $e$ -t hajt végre és  $O$   $o$ -t.

Neumann módszerét **maximin technikának** nevezik, és a következő módon működik:

- Tegyük fel, hogy úgy változtatjuk meg a szabályokat, hogy elsőként  $E$ -nek kötelező felfedni a stratégiáját, majd  $O$  következik. Így egy fordulókon alapuló játékot kapunk, amire a standard **minimax** algoritmust alkalmazhatjuk a 6. fejezetből. Tegyük fel, hogy ez egy  $U_{E,O}$  kimenetelt ad. Nyilvánvaló, hogy ez a játék  $O$ -nak kedvez, így a játék igazi hasznossága ( $E$  szempontjából) legalább  $U_{E,O}$ . Például ha csak tiszta stratégiákat nézünk, a minimax játékfának a gyökérbeli értéke  $-3$  (lásd 17.11. (a) ábra), így tudjuk, hogy  $U \geq -3$ .
- Most tegyük fel, hogy a szabályokat úgy változtatjuk meg, hogy elsőként  $O$ -nak kötelező felfednie a stratégiáját, majd  $E$  következik. Ennek a játéknak a minimax értéke  $U_{E,O}$ , és mivel ez a játék  $E$ -nek kedvez, tudjuk, hogy  $U$  legfeljebb  $U_{E,O}$ . Tiszta stratégiáknál az érték  $+2$  (lásd 17.11. (b) ábra), így tudjuk, hogy  $U \leq +2$ .

Ezt a két érvelést összekapcsolva láthatjuk, hogy a megoldás valódi hasznossága ( $U$ ) eleget fog tenni annak, hogy

$$U_{E,O} \leq U \leq U_{O,E} \quad \text{vagy ebben az esetben} \quad -3 \leq U \leq 2$$

Az  $U$  értékének pontos megállapításához az elemzésünkben át kell térdi a kevert stratégiákra. Elsőként vegyük észre a következőt: *ha az első játékos felfedte a stratégiáját, a második nem veszthet azzal, hogy tiszta stratégiát követ.* Az ok egyszerű: ha a második játékos egy  $[p: \text{egy}; (1-p): \text{kettő}]$  kevert stratégiát játszik, akkor a várható hasznossága a tiszta stratégiák hasznosságainak,  $u_{\text{egy}}$ -nek és  $u_{\text{kettő}}$ -nek egy lineáris kombinációja,  $(p \cdot u_{\text{egy}} + (1-p) \cdot u_{\text{kettő}})$ . Ez a lineáris kombináció nem haladhatja meg az  $u_{\text{egy}}$  és az  $u_{\text{kettő}}$  közül a nagyobbat, így a második játékos is játszhat egy tiszta stratégiát.

Ezt az észrevételt emlékezetünkben tartva, a minimax fáknál képzelhetjük azt, hogy a gyökérnél végtelen sok ág van, amelyek az első játékos által választható végtelen sok kevert stratégiához tartoznak. Ezek mindegyike egy kettős elágazású csomóponthoz vezet, amelyek a második játékos tiszta stratégiához tartoznak. Ezeket a végtelen fákat úgy ábrázoljuk véges módon, hogy a gyökérnél egy „parametrizált” választás van:

- Ha  $E$  lép először, a helyzet a 17.11. (c) ábrán látható.  $E$  a gyökérnél  $[p: \text{egy}; (1-p): \text{kettő}]$  szerint játszik, és  $O$   $p$  értéke alapján választ egy lépést. Ha  $O$   $\text{egy}$ -et választ, a várható jutalom ( $E$ -nek)  $2p - 3(1-p) = 5p - 3$ ; ha  $O$   $\text{kettő}$ -t választ, a várható jutalom  $-3p + 4(1-p) = 4 - 7p$ . Ezeket a jutalmakat mint egyeneseket tüntethetjük fel a grafikonon, ahol  $p$  0 és 1 között lehet az  $x$  tengelyen, ahogyan az

<sup>7</sup> Általánosabb az **állandó összegű játékok** (**constant-sum games**) fogalma, amelyekben a játékbeli minden elemnél az összeg egy c állandót ad. Egy  $n$  személyes állandó összegű játék egy zérusösszegű játékká konvertálható  $c/n$  kivonásával minden jutalomból. Igy a sakk, a tradicionális jutalmakkal, mint 1 a győzelemért.  $\frac{1}{2}$  a döntetlenért és 0 a veszteségért, technikailag egy állandó összegű játék  $c = 1$ -gel, de könnyen egy zérusösszegű játékká konvertálható, ha minden jutalomból kivonunk  $\frac{1}{2}$ -et.

a 17.11. (e) ábrán látható.  $O$ , a minimalizáló, mindenkor az alsót fogja választani a két egyenes közül, amit az ábrán a vastag egyenes jelez. Így a legjobb, amit  $E$  tehet a gyökérnél, hogy  $p$ -t a metszéspontnál választja meg, aminek értéke:

$$5p - 3 = 4 - 7p \Rightarrow p = 7/12$$

A hasznosság  $E$  számára ebben a pontban  $U_{E,O} = -1/12$ .

- Ha  $O$  lép először, a helyzet a 17.11. (d) ábrán látható.  $O$  a gyökérnél [ $q$ : *egy*;  $(1-q)$ : *kettő*] szerint játszik, és  $E$  a  $q$  értéke alapján választ egy lépést. A jutalmak  $2q - 3(1-q) = 5q - 3$  és  $-3q + 4(1-q) = 4 - 7q$ .<sup>8</sup> Ismét látható a 17.11. (f) ábrán, hogy a legjobb, amit  $O$  a gyökérnél tehet, hogy a metszéspontot választja:

$$5q - 3 = 4 - 7q \Rightarrow q = 7/12$$

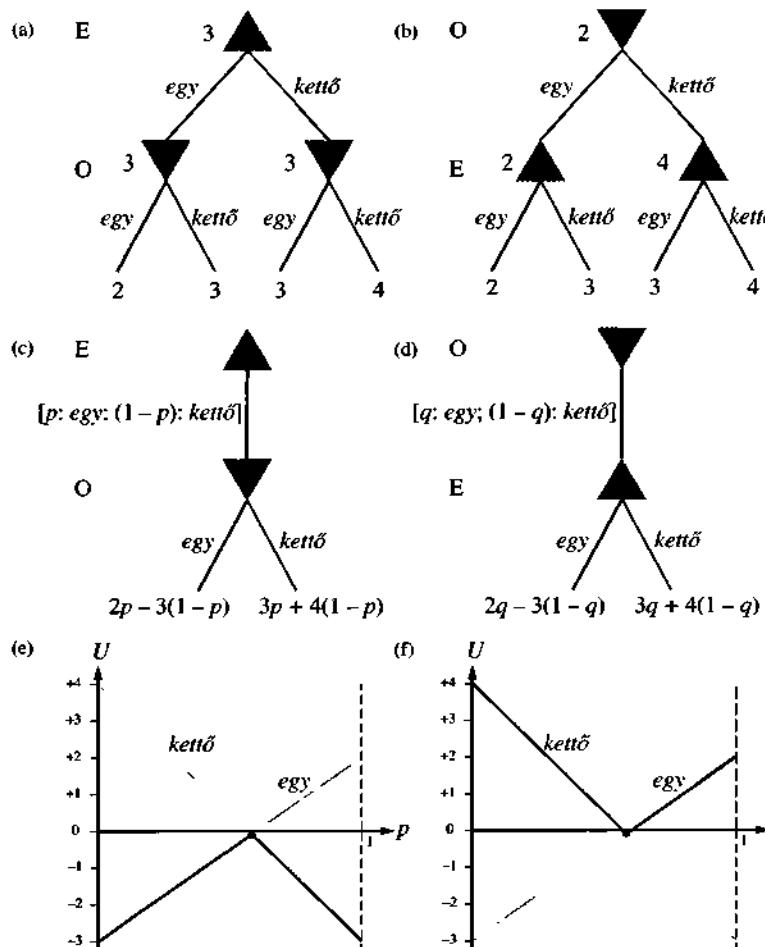
A hasznosság  $E$  számára ebben a pontban  $U_{O,E} = -1/12$ .

Most már tudjuk, hogy a játék valódi hasznossága  $-1/12$  és  $-1/12$  között helyezkedik el, azaz pontosan  $-1/12$ ! (A tanulság az, hogy ebben a játékban jobb  $O$ -nak lenni, mint  $E$ -nek.) Továbbá, a valódi hasznosságot a [7/12: *egy*; 5/12: *kettő*] kevert stratégiával lehet elérni, amit mindenki játékosnak érdemes követnie. Ezt a stratégiát a játék **maximin egyensúlyának** nevezik (**maximin equilibrium**), és ez egy Nash-egyensúly. Vegyük észre, hogy az egyensúlyi kevert stratégia minden komponensének ugyanaz a várható hasznossága. Ebben az esetben mind az *egy*-nek, mind a *kettő*-nek ugyanaz a várható hasznossága,  $-1/12$ , mint magának a kevert stratégiának.

 A snóblijátékra vonatkozó eredményünk Neumann általános eredményének egy példája: minden kétszemélyes, zérusösszegű játéknak van egy *maximin* egyensúlya, amikor kevert stratégiák lehetségesek. Továbbá, egy zérusösszegű játékban minden Nash-egyensúly mindenki játékos számára egy maximin. A maximin egyensúly megtalálásának általános algoritmusára zérusösszegű játékokban kissé bonyolultabb, mint amit a 17.11. (e) és 17.11. (f) ábra sugallhatna. Ha  $n$  cselekvés lehetséges, akkor egy kevert stratégia egy pont az  $n$  dimenziós térben, és az egyenesei hipersíkok lesznek. Az is lehetséges, hogy a második játékos számára bizonyos tiszta stratégiák másokkal domináltak, így ezek nem optimálisak az első játékos *egyetlen* stratégiája ellen sem. Az összes ilyen stratégia eltávolítása után (amit lehet, hogy ismétlődően kell elvégezni), az optimális választás a gyökérnél a legmagasabb (vagy a legalacsonyabb) metszéspontja a megraktárolt hipersíkoknak. Ennek a választásnak a megtalálása **lineáris programozási feladat** (**linear programming**): egy célfüggvény maximálása lineáris kényszerek mellett. Ilyen problémák szabványos technikákkal megoldhatók polinomiális időkomplexitással a cselekvések számában (és a juttalomfüggvény megadásához használt bitek számában, ha precízek akarunk lenni).

A kérdés továbbra is megválaszolatlan, mit kell egy racionális ágensnek konkrétan *tennie* ha a snóblijáték egy menetét játsza? A racionális ágens le fogja vezetni azt a tényt, hogy a [7/12: *egy*; 5/12: *kettő*] egy *maximin* egyensúlyi stratégia, azt fogja feltelevezni, hogy egy racionális ellenfélénél ez a tudás kölcsönös. Az ágens használhat egy 12 oldalú kockát vagy egy véletlenszám generátort, hogy e szerint a kevert stratégia

<sup>8</sup> Az egy egybeesés, hogy ezek az egyenletek ugyanazok, mint a  $p$ -re vonatkozók; az egybeesés amiatt lép fel, mert  $U_E(\text{egy}, \text{kettő}) = U_E(\text{kettő}, \text{egy}) = -3$ . Ez azt is megmagyarázza, hogy az optimális stratégia miért ugyanaz mindenki játékosnál.



17.11. ábra. (a) és (b): Minimax játékfák a kétujjas snóbili játszékhez, ha a játékosok a fordulókban tiszta stratégiáját játszanak. (c) és (d): Parametrikált játékfák, ahol az első játékos kevert stratégiát játszik. A jutalmak függnek a kevert stratégia ( $p$  és  $q$ ) valószínűségi paramétereitől. (e) és (f): A valószínűségi paraméter bármely konkrét értékénél a második játékos a két cselekvés közül a „jobbkat” fogja választani, így az első játékos kevert stratégiájának az értékét a vastag vonal jelöli. Az első játékos a kevert stratégia valószínűségi paraméterét a metszéspontnál fogja megválasztani.

szerint sorsoljon, amely esetben a várható jutalom  $E$  számára  $-1/12$ . Vagy az ágens egyszerűen dönthet úgy, hogy minden 'egy'-et vagy 'kettő'-t játszik. A várható jutalom bármely esetben  $-1/12$  marad  $E$  számára. Érdekes, hogy egy konkrét cselekvés egyoldalú megválasztása nem veszélyeztetni a várható jutalmat, de ha a másik ágnes számára ismerté válik egy ilyen egyoldalú döntés, az már befolyásolja a várható jutalmat, hiszen az ellenfél ennek megfelelően igazíthatja a stratégiáját.

A megoldások (azaz Nash-egyenlőségek) megtalálása nem zéró összegű véges játékokra kissé bonyolultabb. Az általános megközelítés kétrépések: (1) Soroljuk fel a cselekvések összes lehetséges részhalmazát, amelyek kevert stratégiákat alkothatnak. Például

elsőként próbálkozzunk az összes olyan stratégiaprofillal, amelyekben mindegyik játékos egyetlen cselekvést használ, aztán azokkal, amelyekben mindegyik játékos egy vagy két cselekvést használ és így tovább. Ez a cselekvések számában exponenciális, és így csak viszonylag kis játékoknál alkalmazható. (2) minden (1)-ben felsorolt stratégiaprofilnál ellenőrizzük, hogy az egyensúlyi-e. Ez olyan egyenletek és egyenlőtlenségek egy rendszerének a megoldásával érhető el, amelyek hasonlók a zérusösszegű játékban használtakhoz. Két játékosra ezek az egyenletek lineárisak, és alapvető lineáris programozási technikákkal megoldhatók, de három vagy több játékosnál már nem-lineárisak, és esetleg nagyon nehezen oldhatók meg.

Eddig csak egylépéses játékokat vizsgáltunk. A legegyszerűbb többlépéses játék az **ismétlődő játék** (*repeated game*), amelyben a játékosok újra és újra ugyanazzal a választással szembesülnek, de az egyes alkalmakkor már ismert az összes játékos korábbi választásainak a története. Az ismétlődő játéknál egy stratégiaprofil egy cselekvésválasztást ad meg mindegyik játékos számára, minden időpontban és a korábbi választások minden lehetséges történetére. Ahogyan az MDF-eknél, a jutalmak az időben additívák.

Gondoljuk át a fogolydilemma ismétlődő változatát. Együtt fog-e működni Aliz és Bendegúz, visszautasítva a tanúskodást, azt tudva, hogy újra fognak találkozni? A válasz a találkozás részleteitől függ. Tegyük fel, hogy Aliz és Bendegúz tudja, hogy pontosan 100 menetben játszanak fogolydilemmásat. Ekkor mindenketen tudják, hogy a 100. menet nem lesz ismételt játék – azaz, a kimenetelének nincs hatása jövőbeli menetekre –, és ezért ebben a menetben mindenketen a domináns stratégiát, a *tanúskodás-t* választják. De a 100. menet meghatározásával a 99. menetnek nem lehet hatása következő menetekre; így ennek szintén lesz egy domináns stratégiai egyensúlya a (*tanúskodik*, *tanúskodik*)-nál. Indukcióval mindenketét játékos a tanúskodást fogja választani minden menetben, fejenként összesen 500 év börtönt kapva. Elterő megoldásokat nyerhetünk az interakció szabályainak a megváltoztatásával. Tegyük fel, hogy minden menet után 99% esélye van, hogy a játékosok újra találkoznak.

Ekkor a menetek várható száma még mindig 100, de egyik játékos sem tudja biztosan, hogy melyik menet lesz az utolsó. Ezen feltételek mellett együttműködőbb viselkedés lehetséges. Például a *tagadás* mindegyik játékosnak egyensúlyi stratégia mindaddig, amíg a másik játékos nem választ *tanúskodás-t*. Ezt a stratégiát **örökös büntetésnek (perpetual punishment)** nevezik. Tegyük fel, hogy mindenketét játékos ezt a stratégiát fogadta el, és ez kölcsönösen ismert. Ekkor mindenkor, amíg egyik játékos sem választotta a *tanúskodás-t*, bármely időpontban a várható jövőbeli teljes jutalom mindegyik játékosnak

$$\sum_{t=0}^{\infty} 0,99^t (-1) = -100$$

Az a játékos, aki a *tanúskodás-t* választja, egy 0 pontszámot nyer  $-1$  helyett a soron következő lépésben, de a teljes várható jövőbeli jutalma a következőre változik:

$$0 + \sum_{t=1}^{\infty} 0,99^t (-5) = -499,5$$

Ezért egyik lépésnél sincs indíték eltérni a (*tagad*, *tagad*)-tól. Az örökösbüntetés a fogolydilemma egy „kölcsönösen biztosított megsemmisítési” stratégiája: alighogy az egyik játékos a tanúskodás mellett dönt, ez biztosítja, hogy mindenketét játékos nagy vesz-

teséget fog elszennedni. De ez csak akkor elrettentő, ha a másik játékos elhiszi, hogy ezt a stratégiát fogadtuk el – vagy legalábbis hogy elfogadhattuk.

Vannak más stratégiák, amelyek megbocsátóbbak. A leghíresebb a „*szemet szemért*” (*tit-for-tat*), ami szerint *tagadás*-sal kell kezdeni, majd megismételni a másik játékos előző lépését az összes elkövetkező lépésben. Így Aliz tagadna, ameddig Bendegúz tagadna, és a Bendegúz tanúskodása utáni lépésben ő is tanúskodna, de visszatérne a tagadáshoz, ha Bendegúz visszatér. Bár nagyon egyszerű, ez a stratégia igen robusztusnak és hatékonynak bizonyult stratégiák széles köre ellen.

Különböző megoldásokat az ágensek megváltoztatásával is kaphatunk, a találkozás szabályainak változtatása helyett. Tegyük fel, hogy az ágensek véges állapotú gépek  $n$  állapottal, és egy  $m > n$  lépéses játékot játszanak. Az ágensek így nem képesek reprezentálni a hátramaradó lépések számát, így azt ismeretlenként kell kezelni. Így ők nem képesek az indukciós vezetésre, és lehetőségük van a kedvezőbb (*tagad, tagad*) esetet elérni. Ebben az esetben a tudatlanság áldás – vagy inkább az, hogy elhitessük az ellenféllel, hogy tudatlanok vagyunk. Ezekben az ismétlődő játékokban a sikertünk a másik játékos *benyomásán* múlik, hogy nyomulósok vagy mamlaszok vagyunk, és nem a valódi tulajdonságunkon.

Az ismétlődő játékok teljesen általános tárgyalása meghaladja a könyv kereteit, de számos helyzetben megjelennek. Például konstruálhatunk egy szekvenciális játékot, ha a 17.1. ábra  $4 \times 3$ -as világába két ágenst behelyezünk. Ha kikötjük, hogy nem következik be mozgás, amikor a két ágens ugyanarra a mezőre próbál lépni (ugyanaz a probléma számos forgalmi útkereszteződésben), akkor bizonyos tiszta stratégiák örökre beragadhatnak. Megoldás, ha mindegyik ágens véletlenszerűen választ az előrefelé lépés és a helyben maradás között; a hol pont gyorsan feloldódik, és minden ágens boldog lesz. Pontosan ezt teszik az Ethernet-hálózatokban a csomagküszöb feloldására.

Az ismétlődő játékok jelenleg ismert megoldási módszerei a 6. fejezetből ismert többfordulós játékok megoldási módszereihez hasonlítanak abban, hogy egy játékfa építhető a gyökértől lefelé, és megoldható a levelektől felfelé. A fő különbség az, hogy egy gyerekcsomópont értékei felettesegével egyszerű maximum- vagy minimumképzés helyett az algoritmusnak minden szinten meg kell oldania egy kevert stratégiájú játékot, feltételezve, hogy a gyerekcsomópontok már megoldottak, és jól definiált értékük felhasználható.

Az ismétlődő játékokat *részlegesen megfigyelhető* környezetekben *részleges információjú* (*partial information*) játékoknak nevezik. A példák között találhatunk olyan kártyajátékokat, mint a póker és a bridzs, amelyekben mindegyik játékos a kártyáknak csak egy részét láthatja, de olyan komolyabb „játékokat” is, mint a nukleáris háborús modellek, ahol egyik fél sem ismeri az ellenfél összes fegyverének a helyzetét. Részleges információjú játékokat a hiedelmi állapotok fájának a vizsgálatával oldanak meg úgy, mint az RMMDF-ekben (lásd 17.4. alfejezet). Fontos különbség az, hogy míg a saját hiedelmi állapotok megfigyelhetők, addig az ellenfél hiedelmi állapotai nem. Ilyen játékokra csak mostanában fejlesztettek ki a gyakorlatban használható algoritmusokat. Megoldások születtek a póker bizonyos egyszerűbb változataira, bebizonyítva, hogy a blöffölés valóban racionális választás egy jól kiegyensúlyozott kevert stratégia részeként. Az ilyen tanulmányokból alakult ki az a lényeges meglátás, hogy a kevert stratégiák nemcsak azért hasznosak, mert valakinek a cselekvését megjósolhatatlanná teszik, hanem azért is, mert minimalizálják annak az információnak a mennyiségett, amit az ellenfél a cselekvések megfigyelésével tanulni képes. Érdekes, hogy annak ellenére, hogy a bridzs játszásához

való programok tervezői nagyon is tisztában vannak az információ gyűjtésének és elrejtésének a fontosságával, senki sem javasolta véletlenszerű stratégiák használatát.

A játékelmélet felhasználásának az ágenstervezés szélesebb területén eddig számos akadálya volt. Elsőként vegyük észre, hogy egy Nash-egyensúlyban egy játékos teljes mértékben feltételezi azt, hogy az ellenfél egyensúlyi stratégiát játszik. Ez azt jelenti, hogy a játékos semmilyen elvárását sem képes felhasználni a többi játékos valószínű cselekvésével kapcsolatban, és így lehet, hogy olyan fenyegetések ellen való védekezésre vesztegei el az értékeit, amelyek soha nem is lépnek fel. A **Bayes–Nash-egyensúly** (**Bayes–Nash equilibrium**) fogalma részben ezt a hiányosságot pótolja: ez egy olyan egyensúly, ami megfelel egy játékos a priori valószínűség-eloszlásának a többi játékos stratégiái felett – másképpen fogalmazva, kifejezi egy játékos elvárásait a többi játékos valószínű stratégiájáról. Másodszor, jelenleg nincs jó módszer a játékelméleti és az RMMDF irányítási stratégiák összekapcsolására. Ezek és más problémák miatt a játékelméletet elsődlegesen inkább olyan környezetek *elemzésére* használják, amelyek egyensúlyban vannak, a környezetben lévő ágensek *irányítása* helyett. Hamarosan látni fogjuk, hogy a játékelmélet hogyan segítheti a környezetek tervezését.

## 17.7. MŰKÖDÉSI MÓD TERVEZÉS

Az előző alfejezetben azt a kérdést próbáltuk megválaszolni, hogy „Ha adott egy játék, mi a racionális stratégia?” Ebben az alfejezetben azt kérdezzük, hogy „Ha az ágensek racionálisak, milyen játékot tervezünk?” Pontosabban, egy olyan játékot szeretnénk tervezni, aminek a megoldása az egyes ágensek által követett saját racionális stratégiáik együttese, és ez egy globális hasznosságfüggvény maximálását eredményezi. Ezt a problémát **működési mód tervezésnek** (**mechanism design**) nevezik, vagy néha **inverz játékelméletnek** (**inverse game theory**). A működési mód tervezés a közgazdaság-tudományok és a politikai tudományok a lényege. Ágensek együttesénél ez annak lehetőségét hordozza magában, hogy játékelméleti működési módokat használva ügYES rendszereket hozunk létre korlátoltabb rendszerek együtteséből – még nem együttműködő rendszerekből is –, nagyon hasonlóan ahhoz, ahogY emberek csoportjai olyan célokat tudnak elérni, amik messze túl vannak az egyéni lehetőségeken.

A működési mód tervezés példái között olcsó repülőjegyek elárverezése, TCP-csomagok számítógépek közötti továbbításához megfelelő útvonal keresése, szigorló orvosok kórházakhoz rendelése, illetve annak eldöntése szerepel, hogy robotfocisták hogyan működjenek együtt a csapattársaikkal. A működési mód tervezés az 1990-es évektől lépett ki az az egyetemek falai közül, amikor számos ország, a műsorszóró frekvenciasávok elárverezésének a problémájával szembesülve, dollárszázmilliókat vesztett a lehetséges bevételeiből a gyenge működési mód tervezés eredményeképpen. Formálisan a **működési mód** (**mechanism**) tartalmaz (1) egy nyelvet az ágensek által választható megengedett stratégiák (potenciálisan végtelen) halmazának a leírására és (2) egy  $G$  kimeneteli szabályt, ami a jutalmakat határozza meg az ágenseknek megengedett stratégiák egy adott stratégiaprofilja esetén.

Első ránézésre a működési mód tervezés problémája triviálisnak tűnik. Tegyük fel, hogy az  $U$  globális hasznosságfüggvény dekomponálódik egyéni  $U_i$  ágens hasznosságfüggvények valamely halmazára úgy, hogy  $U = \sum_i U_i$ . Ekkor mondhatnánk azt, hogy

ha mindenki ágens maximálja a saját hasznosságát, ez biztosan a globális hasznosság automatikusan maximálásához fog vezetni. (Például a kapitalizmus egyszeregye szerint, ha egy társadalomban mindenki megpróbál gazdagodni, a társadalom összgazdasága növekedni fog.) Sajnos ez nem működik. Az egyes ágensek cselekvései befolyásolhatják a többi ágens jólétét oly módon, hogy a globális hasznosság csökken. Erre példa a **közlegelő tragediája** (**tragedy of commons**), amely helyzetben az egyéni gazdálkodók az összes lábasjószágukat ingyen legelhetik a város közlegelőjén, ezért a közlegelő tönkremegy, és az összes gazdálkodó nagy veszteséget szenved el. Mindegyik gazdálkodó egyénileg racionálisan cselekedett, azzal érvelve, hogy a közlegelő ingyenes, és azzal, hogy bár a közlegelő használata a tönkremeneteléhez vezet, a használatának a mellőzése ezt nem befolyásolja (hiszen mások úgy is használnák). Hasonló érvelések alkalmazhatók a szennyező anyagok kibocsátása esetén a légkör és az óceánok használatára is.

Az ilyen problémák szabványos megközelítése a működési mód tervezésben az, hogy a köztulajdon használatáért mindenki ágensnél költséget számítunk fel. Általánosabban, azt kell biztosítanunk, hogy minden **külsőség** (**externalities**) – olyan globális hasznosságot befolyásoló tényező, amely az egyéni ágensek tranzakcióiban nincsenek elismerve – különállóan nevesítve megjelenjen. Ebben a nehéz a helyes árak megállapítása. Ennek a megközelítésnek a végletes formája egy olyan működési mód megalkotását jelenti, amelyben mindenki ágenstől valójában a globális hasznosság maximálását követeljük meg. Ez megoldhatatlanul nehéz feladat az ágens számára, aki sem megbecsülni nem tudja a világ jelenlegi állapotát, sem megfigyelni a cselekvéseinek az összes többi ágensre gyakorolt hatását. A működési mód tervezés ezért olyan működésmódot megtalálására összpontosít, amelyeknél az egyes ágensek döntési problémája könnyen megoldható.

Gondoljuk át először az árveréseket. Az árverés a legáltalánosabb formájában egy működésmód bizonyos áruknak egy ajánlattevő csoport tagjai számára történő eladására. Az árajánlatok, a stratégiák és a kimenetel meghatározza, hogy ki kapja az árakat és mennyit fizet. A mesterséges intelligencában például az árverések akkor jelennek meg, amikor mindenki egy csoportja eldönti, hogy együttműködjene-e egy közös terven. Hunsberger és Grosz megmutatta, hogy ez hatékonyan elérhető árvéréssel, amiben az ágensek ajánlatokat tesznek a közös tervbeli szerepükre (Hunsberger és Grosz, 2000). Egyelőre olyan árveréseket tekintünk át, amelyekben (1) egyetlen áru van, (2) mindenki ajánlattevőnek van egy  $v_i$  hasznosságértéke az árura, és (3) ezek az értékek csak az ajánlattevő számára ismertek. Az ajánlattevők megteszik a  $b_i$  ajánlataikat, és a legmagasabb ajánlat nyeri el az árakat, de a működésmód határozza meg, hogyan tehetők meg az ajánlatok, és mi a győztes által fizetendő ár (ami nem szükségszerűen  $b_1$ ). Az árverések legismertebb típusa az **angol árverés** (**English auction**), amelyben az árverező mindenkor növeli az áruk árát, amíg csak egyetlen ajánlattevő marad, ellenőrizve közben, hogy vajon az ajánlattevők érdekeltek-e még. Ennek a működésmódnak az a tulajdonsága, hogy a legnagyobb  $v_i$  értékkal bíró ajánlattevő nyeri el az árakat  $b_m + d$  áron, ahol  $b_m$  a legmagasabb ajánlat az összes többi játékos között, és  $d$  az árverező növekménye az ajánlatok között.<sup>9</sup> Az angol árverésnél az ajánlattevőknek egyszerű domináns stratégiájuk van: addig tegyünk árajánlatokat, ameddig a jelenlegi költség a személyes érték alatt van. Emlékezzünk arra, hogy a „domináns” azt jelenti, hogy a stratégia

minden más stratégia ellen működik. Ez viszont azt jelenti, hogy egy játékos a többi stratégiától függetlenül választhatja ezt. Ezért a játékosoknak nem kell időt és energiát vesztegegniük a többi játékos stratégiáján való elmélkedéssel. Egy működésmódot **stratégiamentesnek (strategy-proof)** nevezünk, ha a játékosoknak van domináns stratégiája, ami magában foglalja a valódi indítékok felfedését is.

Az angol árverés hátrányos tulajdonsága a nagy kommunikációs költség, így vagy egy szobában kell az árverésnek lezajlania, vagy az összes ajánlattevőnek nagysebességű, biztonságos kommunikációs csatornával kell rendelkeznie. Egy kevesebb kommunikációt igényő alternatív működésmód a **zárt ajánlatú árverés (sealed bid auction)**. Ebben minden ajánlattevő egyetlen ajánlatot tesz, és ezt közli az árverezővel, és a legnagyobb ajánlat győz. Ennek a működésmódnak az esetében az a stratégia, aminek az árajánlata a valódi érték, már nem domináns. Ha valakinek az értéke  $v_i$ , és az elvárása szerint az összes többi játékos maximális ajánlata  $b_m$  lesz, akkor az árajánlatának a  $v_i$  és  $b_m + \epsilon$  közül az alacsonyabbnak kell lennie. A zárt ajánlatú árverés két hátránya, hogy előfordulhat, hogy a legnagyobb  $v_i$  értékkel bíró játékos nem kapja meg az árakat, illetve az, hogy a játékosoknak fáradozniuk kell a többi játékos stratégiáján való elmélkedéssel.

A zárt ajánlatú árverés szabályainak kis változtatásával adódik a **zárt ajánlatú második áras árverés (sealed bid second-price auction)**, ami Vickrey-árverés (**Vickrey auction**) néven is ismert.<sup>10</sup> Az ilyen árveréseknel a győztes a második legmagasabb ajánlati árat fizeti a saját árajánlatának a kifizetése helyett. Ez az egyszerű módosítás teljesen kiküszöböli a szabványos (avagy első áras, **first-price**) zárt ajánlatú árverés-nél szükséges összetett mérlegeléseket, mivel ekkor a domináns stratégia saját értékű árajánlat. Ennek belátásához vegyük észre, hogy bármely játékos tekintheti az árverést egy kétszemélyes játéknak, figyelmen kívül hagyva az összes játékest, kivéve önmagát és a többi játékos közül a legmagasabb árajánlatot tevőt. Az  $i$ -játékos hasznossága a saját  $b_i$  ajánlata, a  $v_i$  értéke és a többi játékos legjobb  $b_m$  ajánlata szempontjából

$$u_i(b_i, b_m) = \begin{cases} (v_i - b_m) & \text{ha } b_i > b_m \\ 0 & \text{egyébként} \end{cases}$$

Annak belátásához, hogy a  $b_i = v_i$  domináns stratégia, vegyük észre, hogy amikor  $(v_i - b_m)$  pozitív, akkor bármelyik ajánlat optimális, ami megnyeri az árverést, és  $v_i$  megtétele nevezetesen megnyeri az árverést. Másrészt, amikor  $(v_i - b_m)$  negatív, akkor bármelyik ajánlat optimális, ami elveszti az árverést, és  $v_i$  megtétele nevezetesen elveszti az árverést. Így a  $v_i$  árajánlat optimális az összes lehetséges  $b_m$  értékre, és valójában ez csak a  $v_i$  árajánatra teljesül. Az egyszerűsége miatt, továbbá mind az árverező, mind az ajánlattevő számára jelentkező minimális számítási igénye miatt a Vickrey-árverést széles körben használják elosztott MI-rendszerek létrehozásában.

Most gondoljuk át az internet forgalomirányítási problémáját. A játékosok a kapcsolódási háló gráfjában az éléhez tartoznak. minden játékos ismeri egy üzenet küldésének a költségét a saját élén át; ha nincs küldendő üzenet, akkor a költség 0. A cél

<sup>9</sup> Valójában van egy kis esélye, hogy a legnagyobb  $v_i$  értékkel bíró játékosnak nem sikerül megszereznie az árakat abban az esetben, amikor  $b_m < v_i < b_m + d$ . Annak az esélyét, hogy ez megtörténjen, tettszölegesen kicsivé lehet tenni a  $d$  növekmény csökkentésével.

<sup>10</sup> William Vickreyről elnevezve, aki az 1996-os közgazdasági Nobel-díj kitüntetettje.

az, hogy egy üzenet számára megtaláljuk a legolcsóbb utat a kiindulástól a célig, ahol a teljes út költsége az egyes élek költségének az összege. A 4. fejezet számos algoritmust ad az élköltségek ismeretében a legrövidebb út kiszámítására, így minden, amit tennünk kell, az, hogy az egyes ágenseket rávegyük, hogy jelentsék a valódi  $c_i$  költségeket. Sajnos, ha csupán megkérdezzük az ágenseket, akkor magas költséget fog jelenteni, hogy arra bátortíson, küldjük át az üzenetet másol. Ki kell fejlesztenünk egy stratégiamentes működési módot. Egy ilyen működési mód, ha minden játékosnak fizetünk egy  $p_i$  jutalmat, ami a legrövidebb, az  $i$ -edik élet nem tartalmazó út hossza minusz a legrövidebb út (keresési algoritmus által kiszámolt) hossza, ahol az  $i$ -edik él költségét 0-nak tételezzük fel.

$$p_i = \text{Hossz}(\text{út } c_i = \infty\text{-nél}) - \text{Hossz}(\text{út } c_i = 0\text{-nál})$$

Megmutatható, hogy e mellett a működési mód mellett a domináns stratégia mindegyik játékosnak a  $c_i$  összinté jelentése, illetve, hogy ennek megtétele a legolcsóbb utat fogja eredményezni. E kívánatos tulajdonság ellenére az itt vázolt működésmódot a gyakorlatban nem használják a nagy kommunikációs és központi számítási költség miatt. A működési mód tervezőnek kommunikálnia kell az összes  $n$  játékossal, és aztán egy optimalizációs problémát kell megoldania. Ez megérheti, ha a költségek eloszlanának sok üzenet felett, de valós hálózatokban a  $c_i$  költségek folyamatosan változnának a forgalom torlódása, a gépek üzemképtelenné válása és belépése miatt. Eddig teljesen kielégítő megoldást még nem fejlesztettek ki.

## 17.8. ÖSSZEFOGLALÁS

A fejezet megmutatta, hogyan használjuk a világról meglévő tudásunkat döntések meg-hozatalára, még akkor is, amikor egy cselekvés kimenetelei bizonytalanok, és a cselekvésért járó jutalmat nem arathatjuk le csak több cselekvés után. A legfőbb pontok:

- A szekvenciális döntési problémákat bizonytalan környezet esetén, más néven **Markov döntési folyamatokat** vagy **MDF-ket** (**Markov decision process, MDP**), teljesen definiálja egy cselekvések valószínűségi kimenetelét meghatározó **állapot-átmenet-modell** (**transition model**) és az egyes állapotokbeli jutalmakat meghatározó **jutalomfüggvény** (**reward function**).
- Egy állapot-sorozat hasznossága a sorozathoz tartozó összes jutalom összege, ami lehet az idővel leszámított. Egy MDF megoldása egy **eljárás** (**policy**), ami mindegyik ágens által elérhető állapothoz hozzárendel egy döntést. Egy optimális eljárás maximálja a végrehajtása során fellépő állapot-sorozatok hasznosságát.
- Egy állapot hasznossága az azon állapot-sorozatok várható hasznossága, amelyek ebből az állapotból kezdődnek, és egy optimális eljárás végrehajtása esetén lépnek fel. Az MDF-k megoldására szolgáló **értékiteráció** (**value iteration**) algoritmus a működése során az egyenleteket iteratívan oldja meg, az egyes állapotok hasznosságait a szomszédai hasznosságához kapcsolva.
- Az **eljárás-mód-iteráció** (**policy iteration**) felváltva kiszámolja az állapotoknak az aktuális eljárás-hoz tartozó hasznosságát, majd az aktuális hasznosságokhoz képest javít az aktuális eljárásban.

- A részlegesen megfigyelhető MDF-ek vagy RMMDF-ek sokkal nehezebben oldhatók meg, mint az MDF-ek. Megoldásuk során konvertáljuk őket egy olyan MDF-be, amit hiedelmi állapotok folytonos terében definiálunk. Az optimális viselkedés az RMMDF-ekben magában foglalja az információgyűjtést a bizonytalanság csökkenése céljából, és így a jövőbeli jobb döntések meghozatala miatt is.
- Az RMMDF-környezetekre egy döntéselméleti ágens hozható létre. Az ágens egy **dinamikus döntési hálót (dynamic decision network)** használ az állapotátmenet-modell és megfigyelési modell ábrázolására, a hiedelmi állapotának frissítésére, illetve lehetséges cselekvési sorozatok előre elgondolásához.
- A **játékelmélét (game theory)** olyan szituációkban írja le az ágensek racionális viselkedését, amelyekben több ágens egyidejű interakciója van jelen. A játékok megoldásai Nash-egyenstílyok – olyan stratégiaprofilok, amelyekben egyik ágensnek sincs indítéka a megadott stratégiától eltérni.
- A **működési mód tervezés (mechanism design)** használható fel az ágensek interakcióját megadó olyan szabályok megállapítására, amelyek egy globális hasznosságot az egyéni racionális ágensek tevékenységén keresztül maximálnak. Néha léteznek olyan működésmódok, amelyek ezt a célt úgy érik el, hogy az egyes ágenseknek nem kell meggondolniuk a többi ágens által hozott döntéseket.

Az MDF-ek és az RMMDF-ek világára vissza fogunk térti a 21. fejezetben, amikor a megerősítéses tanulás módszerét tanulmányozzuk, ami lehetővé teszi az ágens számára, hogy a tapasztalatai alapján javítsa a viselkedését szekvenciális, bizonytalan környezetekben.

## Irodalmi és történeti megjegyzések

Richard Bellman kezdeményezte a szekvenciális döntési problémák modern megközelítési módját, és javasolta a dinamikus programozás módszerét általában és külön az értékiterációval kapcsolatban is (Bellman, 1957). Egy PhD-disszertációban Ron Howard vezette be az eljárásmód-iterációt és az átlagjutalom fogalmát végtelen horizontú problémák megoldására (Howard, 1960). Számos további eredményt Bellman és Dreyfus vezettek be (Bellman és Dreyfus, 1962). A módosított eljárásmód-iteráció van Nunentől, illetve Putermanról és Shintől származik (van Nunen, 1976; Puterman és Shin, 1978). Az aszinkron eljárásmód-iterációt Williams és Baird elemezték, tőlük származik a (17.9) egyenletbeli korlát is az eljárásmód-veszteségre (Williams és Baird, 1993). A leszámítolás technikáját stacionárius preferenciák segítségével megfogalmazva Koopmans dolgozta ki (Koopmans, 1972). A (Bertsekas, 1987; Puterman, 1994; Bertsekas és Tsitsiklis, 1996) könyvek precíz bevezetést adnak a szekvenciális döntési problémákhoz. Papadimitriou és Tsitsiklis az MDF-k számítási komplexitásáról közölnek eredményeket (Papadimitriou és Tsitsiklis, 1987).

Sutton és Watkins nagy hatású munkái, amelyek az MDF-ek megoldására szolgáló megerősítéses tanulási módszerekre irányultak, jelentős szerepet játszottak az MDF-ek megismertetésében az MI-kutatás területén (Sutton és Watkins, 1988), hasonlóan Barto és társainak későbbi áttekintéséhez (Barto és társai, 1995). (Werbos [1977] korábbi munkája számos hasonló ötletet tartalmazott, de nem vált ilyen széles körben elfogadottá.)

Az első kapcsolatot az MDF-ek és a mesterséges intelligencia tervezési problémái között Sven Koenig teremtette meg (Koenig, 1991), aki megmutatta hogyan nyújtanak a valószínűségi STRIPS operátorok egy tömör ábrázolást az állapotátmenet-modellek számára. (Lásd még Wellman, 1990b.) Dean és munkatársai, illetve Tash és Russell munkáikban a nagy állapotterek miatti kombinatorikus problémákon kíséreltek meg úrrá lenni korlátoszt keresési horizont és absztrakt állapotok felhasználásával (Dean és társai, 1993); Tash és Russell, 1994). Az információ értékén alapuló heurisztikák felhasználhatók olyan állapottérrelkész kiválasztására, ahol a horizont lokális kiterjesztése a döntés minőségének jelentős javulását eredményezi. Ezt a megközelítést használva az ágensek az időkényszerekhez igazíthatják az erőfeszítéseket, és érdekes viselkedést alakíthatnak ki, mint például a jól ismert „kitaposott utak” használatát az állapottérben való gyors útkeresésnél, elkerülve az optimális döntések újból kiszámítását az egyes pontokban. Boutilier és munkatársai mostani munkái propozicionális és elsőrendű kifejezésekben alapuló *szimbolikus* reprezentációk dinamikus programozásbeli felhasználására összpontosultak, mind az állapotátmenet-modellek, mind az értékfüggvények esetében (Boutilier és társai, 2000; 2001).

Azt a megfigyelést, hogy a részlegesen megfigyelhető Markov döntési folyamatok átalakíthatók hagyományos Markov döntési folyamatokká hiedelmi állapotok felhasználásával, Astrom közli (Astrom, 1965). Az első teljes algoritmust a részlegesen megfigyelhető Markov döntési folyamatok egzakt megoldására Edward Sondik javasolta a PhD-disszertációjában (Sondik, 1971). (Egy későbbi folyóiratcikk [Smallwood és Sondik, 1973] tartalmaz néhány hibát, de jobban elérhető.) Az RMMDF-el kapcsolatos aktuális eredmények áttekintését adja Lovejoy (Lovejoy, 1991). Az első jelentős eredmény az MI területén a Witness-algoritmus volt (Cassandra és társai, 1994; Kaelbling és társai, 1998), ami egy javított változata az RMMDF értékiterációjának. Hamarosan következtek más algoritmusok, közöttük egy Hansentől származó megközelítés, ami egy eljárásmódot fokozatosan hoz létre egy végesállapotú automata formájában (Hansen, 1998). Az eljárásmód ezen ábrázolásában a hiedelmi állapotok közvetlenül az automata konkrét állapotaihoz tartoznak. Közelítőleg optimális eljárásmódok az RMMDF-ekhez olyan eljárással hozhatók létre, ahol az előrefelé keresést a lehetséges megfigyelések és cselekvések kimeneteleinek mintavételezésével kombináljuk (Kearns és társai, 2000; Ng és Jordan, 2000). További, az RMMDF-ekre vonatkozó munkákat a 21. fejezetben tárgyalunk.

A dinamikus döntési hálókat használó ágensarchitektúra alapötleteit Dean és Kanazawa javasolták (Dean és Kanazawa, 1989a). Dean és Wellman könyve, a *Planning and Control*, sokkal részletesebb kapcsolatot teremt a DBH/DDH-modellek és a szűrés klasszikus szabályozáselméletbeli szakirodalma között (Dean és Wellman, 1991). Batman és Shachter mutatta meg, hogy hogyan alkalmazhatók dinamikus programozási algoritmusok DDH-modellekre (Batman és Shachter, 1990). Russell különböző módszereket közül ilyen ágensek felskálázására, és számos nyitott kutatási kérdést fogalmaz meg (Russell, 1998).

A játékelmélet kezdetei a 17. századra nyúlnak vissza, Christiaan Huygens és Gottfried Leibniz javaslataihoz, a versengő és együttműködő emberi interakciók tudományos és matematikai vizsgálatára. A 19. század folyamán számos vezető közgazdász alkotott egyszerű matematikai példákat, hogy a versengési helyzetek konkrét esetet elemezze. Az első formális eredmények a játékelméletben Zermelótól származnak

(Zermelo, 1913) (aki egy évvel korábban a minimax keresés egy – bár hibás – változatát javasolta játékokra). Emil Borel vezette be a kevert stratégia fogalmát (Borel, 1921). Neumann János bizonyította be, hogy minden kétszemélyes, zérusösszegű játéknak van egy maximin egyensúlya a kevert stratégiák körében és egy jól definiált értéke (Neumann, 1928). Neumann együttműködése a közgazdász Oskar Morgensternnel vezetett a *Theory of Games and Economic Behavior* c. könyv megjelenéséhez 1944-ben, ami a játékelmélet meghatározó könyve. A könyv megjelenése a háború miatti papírhiány miatti késsett, míg végül a Rockefeller család egy tagja személyesen támogatta a könyv kiadását.

John Nash, 1950-ben, 21 évesen jelentette meg elgondolásait az általános játékbeli egyensúlyi helyzetekkel kapcsolatban. Az egyensúlyi megoldás általa megadott definíciója, bár Cournot munkáján alapult (Cournot, 1838), úgy vált ismertté, mint Nash-egyenlőségek. Az 1959-től fellépő skizofréniája miatti hosszú szünet után Nash 1994-ben közgazdasági Nobel-díjat kapott (megosztva Reinhart Seltennel és Harsányi Jánossal).

A Bayes–Nash-egyenlőséget Harsányi közli (Harsányi, 1967) és Kadane és Larkey elemzi (Kadane és Larkey, 1982). A játékelmélet ágensek vezérlésében történő felhasználásának bizonyos kérdéseit Binmore tárgyalja (Binmore, 1982).

A fogolydilemmát mint tantermi példát Albert W. Tucker alkotta meg és Axelrod tárgyalja részletesen (Axelrod, 1985). Az ismétlődő játékokat Luce és Raiffa vezette be (Luce és Raiffa, 1957) mint részleges információjú játékokat (Kuhn, 1953).

Az első gyakorlati algoritmust részleges információjú játékokra az MI területén Koller és munkatársai fejlesztették ki (Koller és társai, 1996); Koller és Pfeffer cikke olvasmányos bevezetést ad a terület általános részeihez és egy működő rendszert ír le szekvenciális játékok ábrázolására és megoldására (Koller és Pfeffer, 1997). A játékelméletet és az MDF-eket a Markov-játékok elméletében egységesítő Littman (Littman, 1994). Shapley valójában már Bellman előtt közölte az értékiteráció algoritmusát (Shapley, 1953), de az eredményei nem váltak ismertté, talán amiatt, hogy nem a Markov-játékok keretei között voltak megfogalmazva. A játékelmélet általános tárgyalását tartalmazzák a (Myerson, 1991; Fudenberg és Tirole, 1991; Osborne és Rubinstein, 1994) művek.

A köztulajdon tragédiáját, ami működési mód tervezés egy motiváló problémája, Hardin mutatta be (Hardin, 1968). Hurwicz alkotta meg a működési mód tervezés matematikai alapjait (Hurwicz, 1973). Milgrom egy általa tervezett, akár több milliárd dolláros tételekre is szánt árverés működésmódot közöl (Milgrom, 1997). Az árverések a tervezésben (Hunsberger és Grosz, 2000) és az ütemezésben is felhasználhatók (Rassenti és társai, 1982). Varian rövid áttekintést ad a számításelméleti szakirodalomhoz kapcsolódó hivatkozásokkal együtt (Varian, 1995), a Rosenschein és Zlotkin szerzőpárostól pedig egy könyv terjedelmű tárgyalás jelent meg elosztott MI-alkalmazásokról (Rosenschein és Zlotkin, 1994). Az elosztott MI-ben más nevek alatt folynak kapcsolódó munkák, ideértve az együttes intelligenciát (Collective Intelligence), (Tumer és Wolpert, 2000) és a piacalapú szabályozást (market-based control) (Clearwater, 1996). Az árverések számítási vonatkozásairól gyakran az ACM Conferences on Electronic Commerce-en jelennek meg cikkek.

## Feladatok

- 17.1.** A 17.1. ábra által mutatott  $4 \times 3$ -as világ esetén számítsa ki mely négyzetek és milyen valószínűséggel érhetők el az  $[1, 1]$ -ből a  $[Fel, Fel, Jobbra, Jobbra, Jobbra]$  cselekvéssorozattal. Magyarázza el, hogyan kapcsolódik ez a számítás egy rejtegett Markov-modell előre elgondolásához („projecting”).
- 17.2.** Tegyük fel, hogy egy állapot sorozat hasznosságát a sorozat állapotaiban kapott jutalmak maximumaként definiáljuk. Mutassuk meg, hogy ez a hasznosságfüggvény nem eredményez stacionárius preferenciákat az állapotszekvenciák között. Lehetséges-e mégis olyan hasznosságfüggvényt definiálni az állapotokon, hogy a MVH-döntéshozatal optimális viselkedést adjon?
- 17.3.** Pontosan konvertálható-e egy véges keresési probléma Markov döntési problémává úgy, hogy az utóbbi optimális megoldása az előzőnek is optimális megoldása legyen? Ha igen, magyarázza el pontosan, hogyan konvertálja a problémát, és hogyan konvertálja a megoldást visszafelé; ha nem, akkor magyarázza el, hogy miért nem (azaz adjon ellenpéldát).
- 17.4.** Képzeljünk el egy leszámítolatlan MDF-et három állapottal,  $(1, 2, 3)$  és sorrendben a következő jutalmakkal  $-1, -2, 0$ . A 3-as állapot egy végállapot. Az 1-es és 2-es állapotokban két lehetséges cselekvés van:  $a$  és  $b$ . Az állapotátmenetmodell a következő:
- Az 1-es állapotban az  $a$  cselekvés az ágenst a 2-es állapotba mozgatja 0,8 valószínűsséggel, míg 0,2 valószínűsséggel helyben hagyja.
  - Az 2-es állapotban az  $a$  cselekvés az ágenst az 1-es állapotba mozgatja 0,8 valószínűsséggel, és 0,2 valószínűsséggel helyben hagyja.
  - Mind az 1-es, mind a 2-es állapotban a  $b$  cselekvés az ágenst a 3-as állapotba mozgatja 0,1 valószínűsséggel, és 0,9 valószínűsséggel helyben hagyja.
- Válaszolja meg a következő kérdéseket:
- (a) Mi határozható meg *kvalitatívan* az optimális eljárásmódról az 1-es és a 2-es állapotokban?
  - (b) Alkalmazzon eljárásmód-iterációt az optimális eljárásmód, illetve az 1-es és a 2-es állapotok értékének a meghatározására. minden lépést teljesen illusztráljon. Tegyük fel, hogy a kezdeti eljárásmód minden állapotban a  $b$  cselekvés.
  - (c) Mi történik az eljárásmód-iterációval, ha a kezdeti eljárásmód minden állapotban az  $a$  cselekvés? Segít a leszámítás? Függ-e az optimális eljárásmód a leszámítási tényezőktől?
- 17.5.** Alkalmanként az MDF-ek megadása tartalmaz egy  $R(s, a)$  jutalomfüggvényt, ami a véghezvitt cselekvéstől függ, vagy egy  $R(s, a, s')$  jutalomfüggvényt, ami még a kimeneteli állapottól is függ.
- (a) Írja fel a Bellman-egyenleteket ezt a formalizmust használva.
  - (b) Mutassa meg, hogyan lehet egy  $R(s, a, s')$  jutalomfüggvényt tartalmazó MDF-et átalakítani egy másik,  $R(s, a)$  jutalomfüggvényt tartalmazó MDF-be

úgy, hogy az optimális eljárásmódok az új MDF-ben pontosan megfeleljenek az eredeti MDF-beli optimális eljárásmódoknak.

- (c) Most ugyanígy konvertálja az  $R(s, a)$ -t tartalmazó MDF-eket  $R(s)$ -t tartalmazó MDF-ekbe.

**17.6.** Tekintsük a 17.1. ábrán látható  $4 \times 3$ -as világot.

- (a) Valósítson meg egy olyan környezetszimulátort erre a környezetre, amelyben a környezet földrajzi sajátosságai könnyen megváltoztathatók. Ehhez egy bizonyos forráskód már elérhető a könyv forráskódjában a világhálón.
- (b) Hozzon létre egy eljárásmód-iterációt használó ágenst, és mérje meg a teljesítményét a környezetszimulátorban különböző kiindulási állapotokból. Végezzen nagyszámú kísérletet mindenki kiindulási állapotból, és hasonlítsa össze a futásunként kapott összjutalmak átlagát az állapot hasznosságával (amit a létrehozott algoritmus határozott meg).
- (c) Kísérletezzen a környezet méretének a megnövelésével. Hogyan változik az eljárásmód-iteráció futási ideje a környezet méretének a változásával?

**17.7.** A 17.1. ábrán mutatott környezetre keresse meg az  $R(s)$  függvényre az összes

- küszöbértéket, azaz azokat az értékeket, amelyeket átlépésekor az optimális eljárásmód megváltozik. Ehhez egy módszer szükséges az optimális eljárásmód meghatározásához és egy rögzített  $R(s)$  esetén az értékének a kiszámításához. (Segítség: bizonyítsa be, hogy az érték minden rögzített eljárásmódnál  $R(s)$ -sel lineárisan változik.)

**17.8.** Ebben a feladatban kétszereplős MDF-ekkel foglalkozunk, amelyek a zérus-összegű, többfordulós játékokhoz tartoznak, mint amilyenek a 6. fejezetben szerepelnek. Legyen a két játékos  $A$  és  $B$ , és legyen  $R(s)$  az  $A$  játékos jutalma az  $s$  állapotban. ( $B$  jutalma minden rögzített eljárásmódnál  $R(s)$ -sel előjelű.)

- (a) Legyen  $U_A(s)$  az  $s$  állapot hasznossága, amikor  $A$  lépése következik  $s$ -ben, és legyen  $U_B(s)$  az  $s$  állapot hasznossága, amikor  $B$  következik  $s$ -ben. Az összes jutalom és hasznosság  $A$  szempontjából van megfogalmazva (Pontosan úgy, ahogyan a minimax játékfában). Írja fel az  $U_A(s)$ -t és az  $U_B(s)$ -t definiáló Bellman-egyenleteket.
- (b) Magyarázza el, hogyan végezhet kétszemélyes értékiterációt ezekkel az egyenletekkal, és adjon meg egy alkalmas leállási kritériumot.
- (c) Tekintsük a 6.14. ábrán látható játéket a 241. oldalról. Rajzolja fel az állapotteret (a játékfa helyett), folytonos vonallal mutatva  $A$  mozgásait és szaggatott vonallal  $B$  mozgásait. Jegyezze fel minden állapothoz az  $R(s)$  értékét. Hasznos lehet az  $(s_A, s_B)$  állapotok elrendezése egy kétdimenziós rácson,  $s_A$ -t és  $s_B$ -t „koordináteként” használva.
- (d) Most alkalmazza a kétszemélyes értékiterációt a játék megoldására, és származtassa az optimális eljárásmódot.

**17.9.** Mutassuk meg, hogy egy domináns stratégiai egyensúly egy Nash-egyensúly, de ez visszafelé nem igaz.

**17.10.** A kő-papír-olló gyerekjátékban mindenki játékos egyszerre felmutatja a választását, ami kő, papír vagy olló. A papír becsomagolja a követ, a kő kicsorbítja az ollót, az olló elvágja a papírt. A kiterjesztett kő-papír-olló-tűz-víz változatban a tűz üti a követ, papírt és ollót, a kő, papír és olló üti a vizet, és a víz üti a tüzet. Írja fel a jutalmazási mátrixot, és keressen egy kevert stratégiájú megoldást a játékhöz.

**17.11.** Oldja meg a háromujjas snóblijátékot.

**17.12.** A Nemzeti Jégkorong Szövetségben (National Hockey League) 1999 előtt a csapatok 2 pontot kaptak a győzelemért, 1-et a döntetlenért és 0-t, ha veszítettek. Ez zérusösszegű játék? 1999-ben a szabályokat úgy módosították, hogy egy csapat 1 pontot kap, ha hosszabbításban kap ki. A győztes csapat továbbra is 2 pontot kap. Hogyan változtatja meg ez a módosítás a fenti kérdésekre a válaszokat? Ha törvényes lenne, mikor volna racionális a két csapatnak titokban meggyezni, hogy a rendes játékidőben döntetlenül érnek el, és majd a hosszabbításban döntik el a küzdelmet? Tegyük fel, hogy mindenki csapat számára a kapott pontszámok jelentik a hasznosságot, és hogy létezik egy kölcsönösen ismert  $p$  a priori valószínűség, hogy az első csapat hosszabbításban fog nyerni. Milyen  $p$  értékekre értene egyet mindenki csapat ezzel az egyezséggel?

**17.13.** A következő jutalommatrix Blindertől származik, amit Bernstein közöl (Blinder, 1983; Bernstein, 1996), a politikusok és a jegybank (Federal Reserve) közötti játéket ábrázolja.

	Bank: szigorít	Bank: semmi	Bank: lazít
Pol: szigorít	$B = 7, P = 1$	$B = 9, P = 4$	$B = 6, P = 6$
Pol: semmi	$B = 8, P = 2$	$B = 5, P = 5$	$B = 4, P = 9$
Pol: lazít	$B = 3, P = 3$	$B = 2, P = 7$	$B = 1, P = 8$

A politikusok lazíthatják vagy szigoríthatják a pénzügyi politikát, míg a jegybank lazíthatja vagy szigoríthatja a monetáris politikát. (És természetesen mindenki oldal választhatja azt, hogy nem tesz semmit.) Mindegyik oldalnak vannak preferenciái, hogy kinek mit kell tennie – egyik oldal sem szeretne rossznak látszani. A mutatott jutalmak egyszerűen sorrendi rangok: 9 az első választásért és végül 1 az utolsóért. Keresse meg a játék Nash-egyezsűlyét a tiszta stratégiák között. Ez Pareto-optimális megoldás? Az olvasó esetleg érdemesnek tartja elemezni a jelenlegi kormány politikáját ennek fényében.

# **VI. RÉSZ**

# **TANULÁS**

---

# 18. MEGFIGYELÉSEK ALAPJÁN TÖRTÉNŐ TANULÁS

*Ebben a fejezetben olyan ágensekkel foglalkozunk, amelyek saját tapasztalataik szoros tanulmányozásával javítják teljesítményüket.*

A tanulás alapgondolata az, hogy a megfigyeléseket ne csak az ágens jelenlegi cselekvéseinak kialakítására használjuk, hanem arra is, hogy javítsuk a cselekvésre való jövőbeli képességeit. A tanulás aközben zajlik, míg az ágens megfigyeli a környezettel való kölcsönhatásait és saját döntéshozó folyamatait. A tanulás széles skálán értelmezhető: egyik végén a tapasztalatok egyszerű memorizálása található, amit a 10. fejezetben megismert wumpus ágens példáz, a skála másik végén pedig a tapasztalatok alapján kialakított komplett tudományos elmélet áll, mint például Albert Einstein elmélete. Ebben a fejezetben a megfigyelések alapuló **induktív tanulással (inductive learning)** foglalkozunk. Speciálisan azzal, hogy hogyan tanulhatók meg a propozíciós logikában megfogalmazott egyszerű elméletek. Annak elemzésére, elméleti magyarázatára is sor kerül, hogy miért működik az induktív tanulás.

## 18.1. TANULÁSI FORMÁK

A második fejezetben látuk, hogy a tanuló ágens felfogható úgy, mint aminek van egy **cselekvő komponense (performance element)**, amellyel eldönti, hogy milyen cselekvést válasszon, és egy **tanuló komponense (learning element)**, amellyel módosítja ezt a cselekvő elemet annak érdekében, hogy a későbbiekben jobb döntéseket hozzon (lásd 2.15. ábra). A gépi tanulással foglalkozó kutatók a tanuló komponensek széles választékával álltak elő. Ezek megértését segíti, ha megnézzük, hogy a leendő működési területük hogyan befolyásolja tervezésüket. A tanuló komponens tervezését három dolog befolyásolja alapvetően. Ezek:

- A cselekvő elem mely *komponenseit* akarjuk tanítani.
- Milyen *visszacsatolás* áll rendelkezésre ezen komponensek tanítására.
- Hogyan *reprezentáljuk* a komponenseket.

Ezeket a szempontokat rendre elemezni fogjuk. Látuk, hogy számos lehetőség van a cselekvő elemek építésére, a 2. fejezetben számos ágenskialakítást megismertünk (lásd 2.9., 2.11., 2.13. és 2.14. ábra). Ezeknek az ágenseknek a komponensei a következőket tartalmazzák:

1. Az aktuális állapot feltételeinek közvetlen leképezése cselekvésekre.
2. Olyan lehetőség, amely egy megfigyelési szekvenciából a világ releváns tulajdonságaira képes következtetni.

3. A világ alakulására vonatkozó, valamint az ágens lehetséges cselekvéseinek következményeit leíró információ.
4. A világ lehetséges állapotainak számunkra kívánatos voltát megadó hasznosság-információ.
5. Cselekvés-hasznosság információ, amely az egyes cselekvések kívánatosságát jelzi számunkra.
6. Célok: ezek olyan állapotsztályokat adnak meg, amelyek elérése az ágens hasznosságát maximálja.

Megfelelő visszacsatolás alkalmazása esetén ezen komponensek mindegyike tanulható. Vegyük például egy ágenst, amely taxivezetést tanul. minden egyes alkalommal, amikor a felügyelő elkiáltja magát: „Fék!”, az ágens megtanulhat egy feltétel-cselekvés fékezési szabályt (1. komponens). Amennyiben számos olyan képet mutatunk az ágensnek, amelyről közöljük, hogy busz van rajta, akkor megtanulhatja felismerni azokat (2). Ha egy cselekvést próbálga, és megfigyeli az eredményt – például nedves úton erősen fékez –, megtanulhatja cselekvéseinek hatását (3). Ha nem kap boravalót utasától, akit jól összerázott az út, akkor értékes elemekkel gazdagítja az általános hasznosságfüggvényét (4).

Az ágens előtt álló tanulási folyamat meghatározásában rendszerint a visszacsatolás *jellege* a legfontosabb faktor. A gépi tanulás területén általában három esetet különböztetnek meg: **ellenőrzött (supervised)**, **nem ellenőrzött (unsupervised)** és **megerősítéses (reinforcement)** tanulást.

Az **ellenőrzött tanulás (supervised learning)** egy *leképezésnek* a bemeneti és kimeneti minták alapján történő megtanulását jelenti. Az (1), (2) és (3) esetek mindegyike az ellenőrzött tanulás példája. Az (1) esetben az ágens a fékezés feltétel-cselekvés szabályát tanulja – ez az állapotterről egy Boole-típusú kimenetre való leképezés (fékezzen vagy ne fékezzen). A (2) esetben az ágens képekről egy Boole-típusú kimenetre való leképezést tanul meg (tartalmaz-e a kép buszt, vagy sem). A (3) esetben a tanult fékezési elmélet egy, az állapotok és a fékezési cselekvések teréről a – mondjuk méterben mért – megállási útra történő leképezés. Vegyük észre, hogy az (1) és (2) esetekben a tanító szolgáltatta a mintákhoz tartozó helyes választ; a harmadikban a kimeneti értékeket közvetlenül az ágens észlelései szolgáltatták. Teljesen megfigyelhető környezet esetén minden fennáll a lehetőség, hogy az ágens megfigyeli a cselekvésének következményeit, így használhat ellenőrzött tanulási módszereket annak érdekében, hogy megjósolja azokat. Részlegesen megfigyelhető környezetben nehezebb a probléma, mert a közvetlen hatások láthatatlanok maradhatnak.

Nem **ellenőrzött tanulási (unsupervised learning)** probléma esetén bemeneti minták tanulása történik, de a kimeneti kívánt minták nem biztosítottak. Például egy taxivezező ágens apránként kialakíthatja a „jó közlekedési napok” és a „rossz közlekedési napok” koncepcióját, anélkül hogy bármikor címkézett példákat kapott volna bármeiyikről. Egy tisztán nem ellenőrzött tanulást végző ágens nem képes megtanulni, hogy mit cselekedjék, mivel nincs olyan információja, amely egy cselekvést helyesnek vagy egy állapotot kívánatosnak minősítene. A nem ellenőrzött tanulást elsősorban a valósígnúségi következető rendszerek kapcsán fogjuk tárgyalni (lásd 20. fejezet).

A **megerősítéses tanulás (reinforcement learning)** problémája, amelyet a 21. fejezetben tárgyalunk, a legáltalánosabb a három közül. Ahelyett hogy egy tanító útmutatását

követhetné, egy megerősítéses tanulást végző ágensnek **megerősítési információ** alapján kell tanulnia.<sup>1</sup> Például a borravaló hiánya az út végén (vagy egy hatalmas számla, ha beleszáll az előtte haladó kocsiba) némi információt nyújt az ágensnek arról, hogy viselkedése nem volt megfelelő. A megerősítéses tanulás tipikusan magában foglalja azt a részproblémát, hogy az ágensnek meg kell tanulnia azt is, hogyan működik a világ.

A *megtanult információ reprezentációja* szintén nagyon fontos szerepet játszik abban, hogy meghatározhassuk azt, hogyan is kell működnie a tanulási algoritmusnak. Egy ágens bármelyik komponense reprezentálható az ebben a könyvben található sémák bármelyikével. Számos példát látunk: súlyozott polinomokat, amelyeket a játékokat játszó ágensek használtak hasznosságfüggvényként; ítéletkalkulus-beli és elsőrendű logikai állításokat, amelyeket logikai ágensek használtak; valamint valószínűségi leírásokat, mint például a Bayes-hálókat, amelyeket döntéselméleti ágensek használtak következtető komponensként. Ezek mindegyikére alkottak hatékony tanulási eljárásokat. Ebben a fejezetben a propozíciós logikát alkalmazó módszereket tárgyaljuk. A 19. fejezet az elsőrendű logikát alkalmazó módszereket mutatja be, míg a 20. fejezet a Bayes-hálókat és a neurális hálókat tárgyalja (amelyek speciális esetként tartalmazzák a lineáris polinomokat).

A tanuló rendszerek tervezésének utolsó fontos tényezője az *a priori információ rendelkezésre állásának* kérdése. Az MI, a számítástudomány és a pszichológia területén a tanulással foglalkozó kutatás nagy része azzal az esettel foglalkozott, amikor kezdetben az ágensnek semmi információja nincs arról, amit megpróbál megtanulni. Bár ez egy fontos speciális eset, de egyáltalán nem ez az általános. A legtöbb emberi tanulás egy jó adag kiinduló háttérutánra épül. Néhány pszichológus és nyelvész szerint még az újszülötteknek is van ismeretük a világról. Bármilyen legyen az igazság ezt illetően, az kétségtelen, hogy a kiinduló tudás rengeteget segíthet a tanulásban. Buborék-kamra fényképek alapján egy fizikus egy új – meghatározott tömegű és töltésű – részecske létezésére mutató elméletre következtethet. Ugyanakkor egy kritikus ugyanazon képhalmaz vizsgálata alapján csupán arra következtethet, hogy a „művész” bizonyára valamelyen absztrakt vagy expresszionista irányzat követője. A 19. fejezet számos módszert mutat be arra, hogy a tanulást hogyan segíti a meglévő tudás; azt is megmutatjuk ebben a fejezetben, hogy a tudást hogyan lehet úgy összeállítani, hogy felgyorsítsuk a döntéshozatalt. A 20. fejezet azt mutatja be, hogy a kiinduló tudás hogyan segít a valószínűségi elméletek tanulásában.

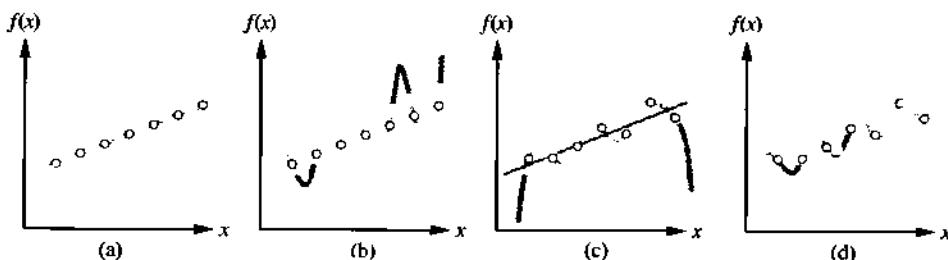
## 18.2. INDUKTÍV TANULÁS

Egy determinisztikus, feliigyejtett tanulást végző algoritmus bemenetként megkapja az ismeretlen függvény bizonyos bemeneti értékekhez tartozó válaszait, feladata az ismeretlen függvény – vagy ahhoz nagyon közel álló leképezés – előállítása. Formálisan azt mondjuk, hogy egy **minta** (*example*) nem más, mint egy  $(x, f(x))$  értékpár, ahol  $x$  a bemeneti érték, míg  $f(x)$  az  $x$ -re alkalmazott függvény kimeneti értéke. A **tisztán induktív következtetés** (*pure inductive inference*) (vagy *indukció* [*induction*]) feladata a következő:

Adott az  $f$ -re vonatkozó minták egy halmaza, ennek alapján határozzunk meg egy  $h$  függvényt, amely közelíti  $f$ -et.

<sup>1</sup> A 17. fejezetben használt **jutalom** (*reward*) kifejezés a **megerősítés** (*reinforcement*) szinonimája.

A  $h$  függvényt **hipotézisnek (hypothesis)** nevezzük. A tanulás nehézsége elvi szempontból abban áll, hogy nem könnyű egy adott  $h$  függvényről megmondani, hogy jól approximálja-e  $f$ -et. Egy jó hipotézis jól általánosít (generalize) – azaz jó becslést kapunk a még nem látott mintákra is. Ez az **induktív következtetés alapproblémája (problem of induction)**. A problémát már évszázadok óta kutatják, a 18.5. alfejezet egy részmegoldást szolgáltat.



**18.1. ábra.** (a) Példa  $(x, f(x))$  értékpárok és egy velük konzisztens lineáris hipotézis. (b) Egy – ugyanakkal az adatokkal konzisztens – hetedfokú polinom. (c) Egy másik adathalmaz egy pontosan illeszkedő hatodfokú polinommal, illetve egy közelítő lineáris illesztéssel. (d) Egy egyszerű színuszos illesztés ugyanazokra az adatokra.

A 18.1. ábra ismert példát mutat be: egy adathalmazra egyváltozós függvényt illesztünk. A minták  $(x, f(x))$  értékpárok, ahol minden  $x$ , minden  $f(x)$  valós értékek. Legyen a **H hipotézistér (hypothesis space H)** – azon hipotézisek halmaza, amelyeket meg fogunk vizsgálni – a legfeljebb  $k$ -ad fokú polinomok tere. (Például  $3x^2 + 2$ ,  $x^{17} - 4x^3$  stb.) A 18.1. (a) ábra valamelyen adatokat és egy az adatokra pontosan illeszkedő egyenes vonalat (elsőfokú polinom) mutat. Ez a vonal itt egy **konzisztens (consistent)** hipotézis, mivel valamennyi adatra illeszkedik. A 18.1. (b) ábra egy nagy fokszámú polinomot mutat, amely szintén konzisztens ugyanazokra az adatokra nézve. Ez a két példa illusztrálja az induktív tanulás első komoly megoldandó kérdését: *hogyan válasszunk több konzisztens hipotézis közül?* Egy lehetséges válasz az ún. **Ockham borotvája<sup>2</sup>** (Ockham's razor) elv: részesítük előnyben a *legegyszerűbb* olyan hipotézist, amely konzisztens az adatokkal. Ez józan intuícióink tűnik, mivel azok a hipotézisek, amelyek nem egyszerűbbek, mint maguk az adatok, nem nyernek ki semmilyen *mintázatot* az adatokból. Az egyszerűség definíciója nem könnyű, de értelmesnek tűnik azt mondani, hogy egy elsőfokú polinom egyszerűbb, mint egy 12-ed fokú.



A 18.1. (c) ábra egy másik adathalmazt mutat. Ezekre az adatokra nem lehet konzisztens módon egyenest illeszteni, valójában egy hatodfokú polinomra (amely 7 szabad paraméterrel rendelkezik) van szükség a pontos illesztéshez. A halmazban csak 7 adat van, így a polinomnak épp annyi paramétere van, ahány adatpont. Ennek megfelelően úgy tűnik, hogy nem sikerül valamelyen mintázatot találnunk az adathalmazban, így nem várunk jó általánosítóképességet. Jobban járhatunk, ha inkább egy egyszerű egyenest illesztünk, amely ugyan nem konzisztens, de ésszerű becsléseket adhat. Ez azt valószí-

<sup>2</sup> A 14. századi angol filozófus, William of Ockham után. Ockham nevét gyakran hibásan Occamnak írják, ami talán a francia „Guillaume d'Occam” fordításából ered.

nősíti, hogy a keresett függvény nem determinisztikus (vagy ami ezzel közelítőleg ekvivalens: a valódi bemeneti minták nem figyelhetők meg teljes pontossággal). *Nem-determinisztikus függvények esetén elkerülhetetlen, hogy kompromisszumot kössük a hipotézis komplexitása és az adatokhoz való illeszkedés pontossága között.* A 20. fejezetben bemutatjuk, hogy ezt a kompromisszumot hogyan lehet a valószínűség-számítás elméletének felhasználásával kialakítani.

Ésben kell tartanunk, hogy egy egyszerű, konzisztens hipotézis megtalálásának esélye erősen függ a választott hipotézistértől. A 18.1. (d) ábra bemutatja, hogy a (c) alatti adatokra pontosan illeszthető egy egyszerű  $ax + b + c \sin(x)$  alakú függvény. Ez a példa jól mutatja a hipotézistér jó megválasztásának fontosságát. Ha olyan hipotézisteret választunk, amely véges fokszámú polinomokat tartalmaz, akkor szinuszos függvényeket nem tudunk pontosan reprezentálni. Így egy olyan tanuló, amelyik ezt a hipotézisteret használja, nem fog tudni szinuszos adatokból tanulni. Egy tanulási problémát **realizálhatónak** (**realizable**) nevezünk, ha a hipotézistérnek eleme az igazi függvény, különben a problémát **nem realizálhatónak** (**unrealizable**) nevezzük. Sajnos nem minden tudjuk megmondani, hogy egy tanulási probléma realizálható-e, mivel nem ismerjük a valódi függvényt. Egy lehetőség e korlát túllépésére az a priori tudás felhasználása. Ennek segítségével olyan hipotézisteret alkotunk, amelyről tudjuk, hogy tartalmazza az igazi függvényt. Ezt a témát a 19. fejezetben tárgyaljuk.

Egy másik lehetséges megközelítés, hogy a lehető legnagyobb hipotézisteret használjuk. Például miért ne legyen a **H** hipotézisterünk az összes Turing-gépek osztálya? Végül is minden kiszámítható függvény reprezentálható valamilyen Turing-géppel, tehát ez a legjobb, amit tehetünk. Ennek a megközelítésnek az a problémája, hogy nem veszi figyelembe a tanulás számítási komplexitását. *Kompromisszumot kell kötnünk a hipotézistér kifejezőképessége és egyszerű konzisztens hipotézisek megtalálásának komplexitása között.* Például: egyenesek illesztése az adatokra könnyen megoldható, nagy fokszámú polinomok illesztése nehezebb, Turing-gépek illesztése pedig nagyon nehéz. Ugyanis annak eldöntése, hogy egy adott Turing-gép konzisztens-e az adatokkal, általánosságban még csak nem is lehetséges. Egy másik ok, ami miatt előnyben részesítjük az egyszerű hipotézistereket, hogy az eredményképp kapott hipotézisek használata is egyszerűbb lehet – könnyebb kiszámítani  $h(x)$ -et, ha  $h$  egy lineáris függvény, mint ha egy tetszőleges Turing-gép programja lenne.

Ezen okokból a legtöbb tanulással foglalkozó eddigi kutatás a viszonylag egyszerű hipotézisreprezentációkra helyezte a fő hangsúlyt. Ebben a fejezetben az ítéletlogikára és az ehhez kapcsolódó nyelvekre koncentrálunk. Az elsőrendű logikai reprezentációt használó tanulás elméletével a 19. fejezet foglalkozik. Látni fogjuk, hogy a kifejezőképesség-komplexitás kompromisszum nem is olyan egyszerű, mint először gondolnánk. Gyakran az a helyzet – mint azt a 8. fejezetben is látta –, hogy egy kifejezőnyelv lehetővé teszi az adatokra illesztett egyszerű elméletet, míg a nyelv kifejezőképességenek korlátozása azt is jelenti, hogy bármely konzisztens elmélet szükségszerűen bonyolult lesz. A sakk szabályait például egy-két oldalon le tudjuk írni elsőrendű logika segítségével, míg az ítéletlogikával történő leírás több ezer oldalt igényelne. Ilyen esetekben kell arra lehetőségnak lennie, hogy a kifejezőbb nyelven sokkal gyorsabb legyen a tanulás.

## 18.3. DÖNTÉSI FÁK MEGALKOTÁSA TANULÁSSAL

A döntési fa tanulás egyike a legegyszerűbb, mégis az eddigiekben az egyik legsikeresebbnek bizonyult tanulási algoritmusnak. Jó bevezetésként szolgál az induktív tanulás területén, és ráadásul könnyen implementálható. Először bemutatjuk a cselekvő alrendszert, majd megmutatjuk, hogy miképpen lehet tanítani. Mindeközben olyan elveket mutatunk be, amelyek az induktív tanulás minden területére jellemzők.

### Cselekvő komponensként használt döntési fák

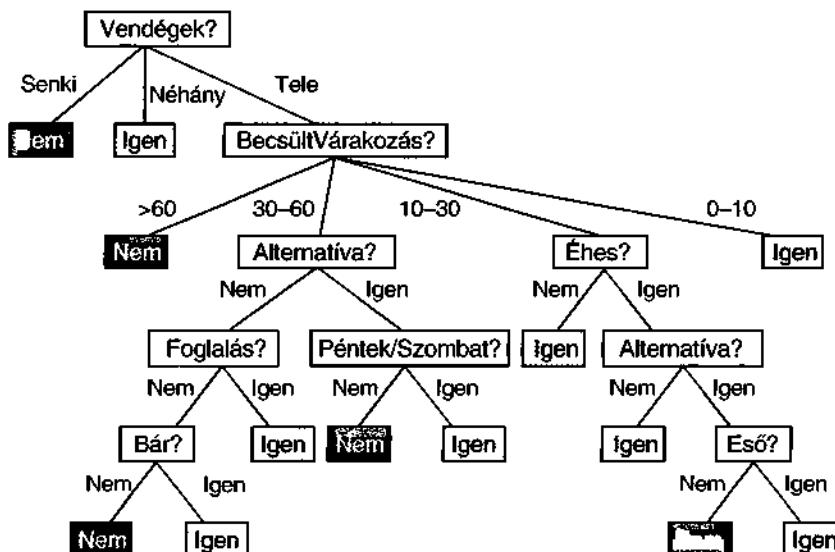
Egy **döntési fa** (**decision tree**) bemenetként egy **attribútumokkal** (**attributes**) leírt objektumot vagy szituációt kap, és egy „**döntést**” ad vissza eredményként – a bemenetre adott válasz jóolt értékét. A bemeneti attribútumok lehetnek diszkrétek vagy folytonosak. Jelen tárgyalásban diszkrét bemeneteket tételezünk fel. A kimeneti érték szintén lehet diszkrét vagy folytonos; egy diszkrét értékkészletű függvény tanulását **osztályozás** (**classification**) tanulásnak, míg a folytonos függvény tanulását **regresszió**nak (**regression**) nevezzük. *Bináris* (*Boolean*) osztályozásra fogunk koncentrálni, ahol minden példát vagy igaznak (**pozitív**), vagy hamisnak (**negatív**) sorolunk be.

A döntési fa egy tesztsorozat elvégzése során jut el a döntéshez. A fa minden egyes belső csomópontja valamely tulajdonság értékére vonatkozó tesztnek felel meg, a csomópontból kilépő ágakat pedig a teszt lehetséges kimeneteivel címkezzük. minden egyes levélcsomópont megadja azt az értéket, amelyet vissza kell adnunk, ha ezt a levelet elérünk. Úgy tűnik, hogy a döntési fa reprezentáció az emberek számára rendkívül természetes; valójában számos „Hogyan csináljuk?” kézikönyvet (például az autójavítási kézikönyveket) írtak meg egyetlen hatalmas, több száz oldalra kiterjedő döntési faként.

Valamelyest egyszerűbb példa lehet az a probléma, hogy várunk-e egy étteremben egy asztal felszabadulására. Az a célunk, hogy tanulással kialakítsuk a **Várunk-e** cél-predikátum (**goal predicate**) definícióját. Ahhoz, hogy ezt tanulási feladatként kezelhessük, először meg kell határozzuk, hogy milyen attribútumok állnak rendelkezésre ahhoz, hogy ezen a problématerületen leírjuk a példákat. A 19. fejezetben megmutatjuk, hogyan lehet automatizálni ezt a feladatot; most egyszerűen tegyük fel, hogy a következő attribútumlista mellett döntöttünk:

1. *Alternatíva*: van-e a közelben megfelelő alternatíváként kínálkozó étterem.
2. *Bár*: van-e az étteremnek kényelmes bár része, ahol várakozhatunk.
3. *Péntek/Szombat*: igaz értéket vesz fel pénteken és szombaton.
4. *Éhes*: éhesek vagyunk-e.
5. *Vendégek*: hány ember van az étteremben (értékkészlet *Senki, Néhány és Tele*).
6. *Drága*: az étterem menyire drága (\$, \$\$, \$\$\$).
7. *Eső*: esik-e odakint az eső.
8. *Foglalás*: foglaltunk-e asztalt.
9. *Konyha*: az étterem típusa (francia, olasz, thai vagy burger).
10. *BecsültVárakozás*: a pincér becsültére várakozási idő (0–10, 10–30, 30–60, >60 perc)

A szerzők egyike (SR) által erre a problémára rendszerint használt döntési fa a 18.2. ábrán látható. Vegyük észre, hogy a fa nem használja a *Drága* és a *Konyha* attribútumokat,



18.2. ábra. Egy döntési fa annak eldöntésére, hogy várunk-e asztalra

valójában irrelevánsnak tekinti azokat. A példákat a döntési fa a gyökérnél kezdi fel-dolgozni, követi a megfelelő ágakat, amíg egy levélhez el nem ér. Például egy *Vendégek = Tele* és *BecsültVárakozás = 0–10* attribútumokkal jellemzhető példa pozitív kimenet fog eredményezni (azaz várni fogunk egy asztalra).

## A döntési fák kifejezőképessége

Logikai felírást használva minden egyes döntési fa, amely a *VárunkE* célpredikátum egy hipotézise, a következő formában felírt állításnak felel meg:

$$\forall s \quad \text{VárunkE}(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \dots \vee P_n(s))$$

ahol minden egyik  $P_i(s)$  feltétel azon tesztek konjunkciójának felel meg, amelyeket a gyökértől egy pozitív kimenetet jelentő levélig megtett út során végeztünk. Bár ez egy elsőrendű logikában felírt mondatnak tűnik, valójában bizonyos értelemben ítéletlogikai kifejezés, mivel csak egyetlen változót tartalmaz és az összes predikátum unáris. Valójában a döntési fa a *VárunkE* és az attribútumértékek bizonyos logikai kapcsolatát írja le. A döntési fa nem használható olyan tesztek reprezentálására, amelyek kettő vagy több különböző objektumra vonatkoznak – például:

$$\exists r_2 \quad \text{Közeli}(r_2, r) \wedge \text{Drága}(r, p) \wedge \text{Drága}(r_2, p_2) \wedge \text{Olcsóbb}(p_2, p)$$

(van a közelben egy olcsóbb étterem?). Nyilvánvalóan felvehetnénk egy *OlcsóbbÉtteremKözeli* nevű logikai attribútumot, de az összes ilyen attribútum hozzáadása kezelhetetlenné teszi a problémát. A 19. fejezet mélyebbre hatol az elsőrendű logikában történő tanulás területén.

Az ítéletlogikai nyelvek területén a döntési fák teljes kifejezőképességgel bírnak, ami azt jelenti, hogy tetszőleges logikai (Boole) függvény felírható döntési faként. Ezt triviálisan megvalósíthatjuk, ha a függvény igazságtáblájának minden sorát megfelelőt jük a döntési fa egy útjának. Ez exponenciálisan növekvő döntési fára vezet, mivel az igazságtábla exponenciálisan növekvő számú sort tartalmaz. Nyilvánvalóan a döntési fák sok függvényt jóval kisebb fával képesek reprezentálni.

Ugyanakkor némely függvényfajtánál ez valóban problémát jelent. Például ha a függvényünk a **paritásfüggvény (parity function)**, amely akkor és csak akkor ad 1-et, ha páros számú bemenet 1 értékű, akkor egy exponenciálisan nagy döntési fára lesz szükség. Hasonlóan nehéz a **többségsfüggvény (majority function)** reprezentálása döntési fával, amely függvényt akkor ad 1-et, ha bemeneteinek több mint fele 1 értékű.

Más szavakkal, a döntési fák bizonyos függvények esetén jók, mások esetén rosszak. Van *bármilyen* olyan reprezentáció, amely mindenfajta függvény esetén hatékony? A válasz sajnos az, hogy nincs. Ezt általában is meg tudjuk mutatni. Vizsgáljuk az összes  $n$  bemeneti attribútummal rendelkező logikai (Boole) függvényt. Hány különböző függvény van ebben a halmazban? Ez éppen a lehetséges felírható igazságtáblák számával egyezik meg, hiszen egy függvényt az igazságtáblája ad meg. Az igazságtáblának  $2^n$  sora van, mivel minden bemeneti esetet  $n$  attribútummal adunk meg. Úgy tekinthetjük a tábla „válasz” oszlopát, mint egy  $2^n$  bites számot, amely definiálja a függvényt. Mindegy, hogy a függvények milyen reprezentációját választjuk, néhánynak (valójában szinte mindegyiknek) a reprezentálásához tényleg szükség lesz ennyi bitre.

Ha  $2^n$  bitre van szükség a függvény megadásához, akkor  $n$  attribútum esetén  $2^n$  a lehetséges függvények száma. Ez megrázóan nagy szám. Például csupán hat Boolean változó esetén is  $2^6 = 18\,446\,744\,073\,709\,551\,616$  különböző logikai (Boole) függvény állítható elő. Szükségünk lesz néhány szellemes algoritmusra, hogy egy ilyen hatalmas téren konzisztens hipotézist tudjunk találni.

## A döntési fák példák alapján történő felépítése

Egy logikai döntési fa által kezelhető példa a bemeneti attribútumok  $X$  vektorából és egyetlen logikai kimeneti értékből,  $y$ -ból áll. A 18.3. ábra egy  $(X_1, y_1) \dots (X_{12}, y_{12})$  példa-halmazt mutat. Azokat nevezzük pozitív példáknak  $(X_1, X_3, \dots)$ , amelyekben a *VárunkE* értéke igaz, és azok a negatív példák, amelyekben az értéke hamis  $(X_2, X_5, \dots)$ . A példák teljes halmazát **tanító halmaznak (training set)** nevezzük.

Az a probléma, hogy a tanító halmaznak megfelelő döntési fát találunk, bonyolultnak tűnik ugyan, de valójában van egy triviális megoldása. Egyszerűen egy olyan fát konstruálhatunk, amelyben minden példához egy külön saját utat hozunk létre egy – a példához tartozó – levélcsomóponthoz. A levélhez vezető út mentén sorra teszteljük az attribútumokat, és a példához tartozó tesztértéket követjük, a levél pedig a példa besorolását adja. Ha ismét ugyanazt a példát<sup>3</sup> vesszük, akkor a döntési fa a helyes osztályozási eredményt adja. Szerencsétlen módon nemigen ad információt egyetlen más esetről sem!

<sup>3</sup> Nagyon fontos, hogy megkülönböztesse az azonos példát egy *azonos leírással rendelkező példától*. Erre a fontos különbségre még visszatérünk a 19. fejezetben.

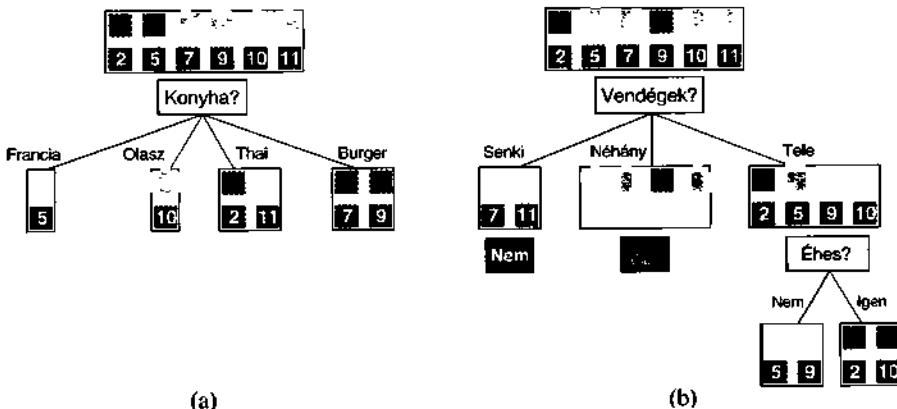
Példa	Attribútumok											Cél Vár- junkE
	Alte- rnatíva	Bár	Péntek/ Szombat	Éhes	Vendégek	Drága	Eső	Foglalás	Konyha	Becsült Várakozás		
X <sub>1</sub>	Igen	Nem	Nem	Igen	Néhány	\$\$\$	Nem	Igen	Francia	0–10	Igen	
X <sub>2</sub>	Igen	Nem	Nem	Igen	Tele	\$	Nem	Nem	Thai	30–60	Nem	
X <sub>3</sub>	Nem	Igen	Nem	Nem	Néhány	\$	Nem	Nem	Burger	0–10	Igen	
X <sub>4</sub>	Igen	Nem	Igen	Igen	Tele	\$	Igen	Nem	Thai	10–30	Igen	
X <sub>5</sub>	Igen	Nem	Igen	Nem	Tele	\$\$\$	Nem	Igen	Francia	>60	Nem	
X <sub>6</sub>	Nem	Igen	Nem	Igen	Néhány	\$\$	Igen	Igen	Olasz	0–10	Igen	
X <sub>7</sub>	Nem	Igen	Nem	Nem	Senki	\$	Igen	Nem	Burger	0–10	Nem	
X <sub>8</sub>	Nem	Nem	Nem	Igen	Néhány	\$\$	Igen	Igen	Thai	0–10	Igen	
X <sub>9</sub>	Nem	Igen	Igen	Nem	Tele	\$	Igen	Nem	Burger	>60	Nem	
X <sub>10</sub>	Igen	Igen	Igen	Igen	Tele	\$\$\$	Nem	Igen	Olasz	10–30	Nem	
X <sub>11</sub>	Nem	Nem	Nem	Nem	Senki	\$	Nem	Nem	Thai	0–10	Nem	
X <sub>12</sub>	Igen	Igen	Igen	Igen	Tele	\$	Nem	Nem	Burger	30–60	Igen	

18.3. ábra. Példák az étterem problématerületre

Ezzel a triviális megoldással az a baj, hogy egyszerűen memorizálja a példákat. Mivel nem nyer ki semmilyen mintázatot a példákból, ezért nem várhatjuk, hogy extrapolálni tudjon azokra a mintákra, amelyeket még sohasem látott. Az Ockham borotvája elvét alkalmazva nekünk azt a *legkisebb* döntési fát kell megtalálnunk, amely konzisztenst a példákkal. Sajnálatos módon a „legkisebb” összes értelmes definíciója esetén a legkisebb döntési fa megtalálása kezelhetetlen problémát jelent. Viszont némi egyszerű heurisztika bevetésével jó eredményt érhetünk el egy „kicsike” fa megtalálásában. A DÖNTÉSI-FA-TANULÁS algoritmus alapötlete az, hogy teszteljük először a legfontosabb attribútumot. A „legfontosabb” alatt azt az attribútumot értjük, amelyik a legnagyobb változást okozza a példák besorolásában. Ezen a módon – reményünk szerint – kisszámu tesztel helyes osztályozáshoz jutunk, ami azt jelenti, hogy a fában minden út rövid lesz, tehát az egész fa kicsi lesz.

A 18.4. ábrán látható, hogy hogyan indul az algoritmus. 12 példánk van, amelyeket pozitív és negatív példahalmazokba sorolunk. Ezek után eldöntjük, hogy melyik attribútumot teszteljük először a fában. A 18.4. (a) ábrán megmutattuk, hogy a *Konyha* attribútum rossz választás lenne, mert tesztjének 4 kimenetele van, és mindegyik esetén ugyannyi pozitív és negatív példánk lesz az eredményül kapott halmazokban. Másrészt viszont a 18.4. (b) ábrán látható, hogy a *Vendégek* egy meglehetősen fontos attribútum, hiszen ha értéke *Senki* vagy *Néhány*, akkor határozott választ tudunk adni (az első esetben *Nem*, a másodikban *Igen* a válaszunk). Ha az attribútum értéke *Tele*, akkor a példák vegyes halmazát kapjuk. Általánosságban megállapítható, hogy miután az első attribútum tesztje csoportokra bontotta a példákat, mindegyik teszteredmény egy újabb döntési fa tanulási problémát eredményez, kevesebb példával és egygyel kevesebb attribútummal. Négy esetet kell áttekintenünk ezekben a rekurzív problémákban:

1. Ha van néhány pozitív és néhány negatív példánk, akkor válasszuk a legjobb attribútumot a szétosztásukra. A 18.4. (b) ábrán bemutattuk, hogy az *Éhes* attribútum alkalmas a megmaradó példák osztályozására.



**18.4. ábra.** A példahalmaz attribútumteszteléssel történő szétosztása. (a) A *Konyha* alapján történő szétosztás nem visz közelebb ahhoz, hogy a pozitív és negatív példákat megkülönböztessük. (b) A *Vendégek* attribútum tesztje jól szeparálja a pozitív és negatív példákat. Miután a *Vendégek* tesztjét elvégeztük, az *Éhes* egy elég jó második tesztelhetőséget ad.

2. Ha valamennyi megmaradt példánk pozitív (vagy minden negatív), akkor készen vagyunk: válaszolhatunk *Igen*-t vagy *Nem*-et. A 18.4. (b) ábra bemutatja ezt a *Senki*, illetve a *Néhány* esetekben.
3. Ha nem marad példa a teszt egyik kimenetele esetén, akkor ez azt jelenti, hogy nem figyeltünk meg ilyen esetet, és a szülöcsomópontban többségben levő választ adjuk.
4. Ha nem maradt attribútumunk, amelyet tesztelhetnénk, de minden pozitív, minden negatív példáink maradtak, akkor bajban vagyunk. Ez azt jelenti, hogy ezeknek a példáknak pontosan azonos jellemzőik vannak, de különböző osztályokba tartoznak. Ez egrészt akkor fordulhat elő, ha néhány adat nem megfelelő, azt mondjuk, hogy zajosak (**noise**) az adatok. Másrészt akkor is előállhat ez a helyzet, ha az attribútumok nem adnak elég információt a szituáció teljes leírására, vagy a problémater valójában nemdeterminisztikus. Egyszerű megoldása lehet ennek a problémának a többségi szavazás használata.

A DÖNTÉSI-FA-TANULÁS algoritmust a 18.5. ábrán mutatjuk be. Az ATTRIBÚTUM-VÁLASZTÁS módszer bemutatása a következő alfejezetben található.

Az algoritmus által a 12 mintából álló adathalmaz alapján létrehozott fa a 18.6. ábrán látható. Az eredményként kapott fa nyilvánvalóan különbözik a 18.2. ábrán látható eredetitől, annak ellenére, hogy az adatokat egy, az eredeti fát használó ágens generálta. Arra gondolhatnánk, hogy az algoritmus nem ért el különösebben jó eredményt a helyes függvény megtanulásában. Mindamellett ez téves következtetés lenne. A tanuló algoritmus a példákat látja, nem az eredeti függvényt. Valójában a létrehozott hipotézis (lásd 18.6. ábra) nem csupán megfelel az összes példának, de lényegesen egyszerűbb is, mint az eredeti fa. A tanuló algoritmusnak semmi oka sincs az *Eső* és *Foglalás* attribútumokra vonatkozó tesztekkel felvenni, mivel ezek nélkül is tudja osztályozni az összes példát. Mellesleg, felfedett egy érdekes és váratlan viselkedést: az első szerző hétvégeken várni fog thai ételekre.

```

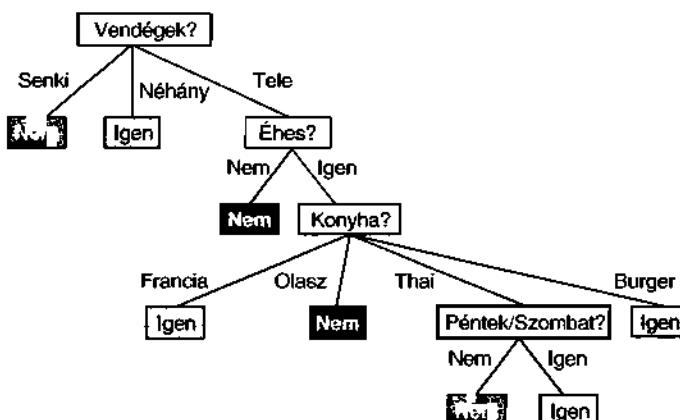
function DÖNTÉSI-FA-TANULÁS(példák, attribútumok, alapérték) returns egy döntési fa
  inputs: példák, a példák halmaza
           attribútumok, az attribútumok halmaza
           alapérték, a célpredikátum alapértéke

  if példák üres then return alapérték
  else if példák minden elemének azonos a besorolása then return a besorolás
  else if attribútumok üres halmaz then return TÖBBSÉGI-ÉRTÉK(példák)
  else
    legjobb  $\leftarrow$  ATTRIBÚTUM-VÁLASZTÁS(attribútumok, példák)
    fa  $\leftarrow$  egy új döntési fa, a gyökér a legjobb attribútum teszije
    m  $\leftarrow$  TÖBBSÉGI-ÉRTÉK(példák)
    for each legjobb minden  $v_i$  értékére do
      példáki  $\leftarrow$  [a példák azon elemei, amelyekre legjobb =  $v_i$ ]
      részfa  $\leftarrow$  DÖNTÉSI-FA-TANULÁS(példáki, attribútumok-legjobb, m)
      a fa döntési fához adjunk egy  $v_i$  címkéjű ágat és a részfa részfát
    return fa

```

18.5. ábra. A döntési fa tanulásra szolgáló algoritmus

Természetesen ha több mintát gyűjtöttünk volna, akkor egy, az eredetihez hasonlóbb fát hozhattunk volna létre az induktív tanulással. A 18.6. ábrán látható fa elkerülhetetlenül hibákat is el fog követni; például sohasem látott olyan helyzetet, amelyben a váratkozási idő 0–10 perc, de az étterem tele van. Egy olyan esetben, amikor az Éhes attribútum hamis értékű, a fa azt választja, hogy ne várunk, pedig én (S. Russell) bizonyára várnék. Felmerül tehát az a kézenfekvő kérdés, hogy ha az algoritmus egy konzisztnens, de inkorrekt fát indukál, akkor mennyire lesz inkorrekt ez a fa? Ezt azután fogjuk kísérletileg elemezni, hogy először elmagyaráztuk az attribútumteszt-választási lépés részleteit.



18.6. ábra. A 12 példás tanító minta halmzból származtatott döntési fa

## Attribútumteszt-választás

A döntési fa tanulás során az attribútumok kiválasztására szolgáló eljárás arra irányul, hogy minimalizáljuk az eredményül kapott fa mélységét. Az alapötlet az, hogy azt az attribútumot válasszuk, amellyel a lehető legmesszebbre jutunk a példák pontos osztályozásában. Egy tökéletes attribútum a példákat egy csupa pozitív és egy csupa negatív példát tartalmazó halmazra osztja. A Vendégek nem tökéletes attribútum, de meglehetősen jó. Egy valójában haszontalan attribútum, mint például a Konyha, tesztjének eredményeként a kapott halmazokban nagyjából ugyanolyan arányban lesz pozitív és negatív példa, mint az eredeti halmazban.

Mindössze arra van szükségünk, hogy formális mértéket találunk arra, hogy mit jelent a „melegelhetősen jó” és a „valójában haszontalan”, ezek után implementálni tudjuk a 18.5. ábrán látható ATTRIBÚTUM-VÁLASZTÁS függvényt. A mérték akkor érje el maximumát, amikor az attribútum tökéletes, és akkor legyen minimális, amikor az attribútumnak egyáltalán nincs semmi haszna. Egy megfelelő mérték az attribútum által szolgáltatott **információ (information)** várható értéke, ahol az információt abban a matematikai értelmezésben használjuk, ahogy először Shannon és Weaver definiálta (Shannon és Weaver, 1949). Az információ definíciójának megértéséhez gondoljunk például annak a kérdésnek a megválaszolására, hogy egy feldobott pénzérme fej oldala lesz-e felül. Az, hogy a válaszban mennyi információ rejlik, az az előzetes a priori tudás-tól függ. Minél kevesebbet tudunk, annál több a szolgáltatott információ. Az információ-elmélet **bitekben** méri az információtartalmat. Egy bit információ ahhoz elég, hogy egy olyan kérdésre, amelyről semmilyen előzetes elképzelésünk sem volt (például egy szabályos érme feldobásának eredménye) igen/nem választ megadjunk. Általánosságban, ha a lehetséges  $v_i$  válaszok valószínűsége  $P(v_i)$ , akkor a válasz  $I$  információtartalmát a következő összefüggés adja meg:

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

A pénzfeldobás esetére ellenőrizve az összefüggést, a következőt kapjuk:

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit}$$

Ha az érme hamisított oly módon, hogy a dobás 99%-ban fejre jön ki, akkor  $I(1/100, 99/100) = 0,08$  bitre van szükség, és ha a fej kimenetel valószínűsége 1-hez tart, akkor az aktuális kimenetel megjósolásához szükséges információ 0-hoz tart.

Döntési fa tanulás esetén a megválaszolásra váró kérdés az, hogy egy adott példának mi a helyes besorolása. Egy jó döntési fa választ ad erre a kérdésre. Még mielőtt egyetlen attribútumot teszteltünk volna, a válaszok valószínűségét becsülhetjük a tanító halmazban található pozitív és negatív minták arányával. Tegyük fel, hogy a tanító halmazban  $p$  pozitív és  $n$  negatív példa található. Ez esetben a helyes válasz információtartalmának becslése:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

A 18.3. ábrán bemutatott étterem tanító halmaz  $p = n = 6$  mintát tartalmaz, tehát 1 bit információra van szükségünk.

Egyetlen  $A$  attribútum tesztje nem fogja megadni mindezt az információt, de valamennyit megad belőle. Pontosan mérni tudjuk, hogy mennyit, ha megnézzük, hogy mennyi információra van még szükségünk az attribútumteszt után. Bármely  $A$  attribútum, amely  $v$  különböző értéket vehet fel,  $E_1, \dots, E_v$  részhalmazokra bontja az  $E$  tanító halmazt, az  $A$  lehetséges értékei szerint. Mindegyik  $E_i$  részhalmaz  $p_i$  pozitív és  $n_i$  negatív példát tartalmaz, így ezen az ágon továbbhaladva  $I(p_i/(p_i + n_i), n_i/(p_i + n_i))$  bit információra van szükségünk a válasz megadásához. A tanító halmazból véletlen mintavétellel nyert minta esetén annak valószínűsége, hogy az  $A$  attribútum ezen a mintán az  $i$ -edik értéket veszi fel:  $(p_i + n_i)/(p + n)$ . Ennek megfelelően az  $A$  attribútum tesztje után az átlagos információszükséglet, amely a példa osztályozásához kell:

$$\text{MaradékInfoSzükséglet}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

Az attribútum tesztjéből származó **információnyereség** (information gain) az eredeti ( $A$  tesztje előtti) információszükséglet és a teszt utáni új információszükséglet különbségeként kapható meg:

$$\text{InfoNyeréség}(A) = I\left(\frac{P}{p+n}, \frac{n}{p+n}\right) - \text{MaradékInfoSzükséglet}(A)$$

Az ATTRIBÚTUM-VÁLASZTÁS függvényben használt heurisztika csupán azt takarja, hogy válasszuk a legnagyobb nyereséget biztosító attribútumot. Visszatérve a 18.4. ábrán vizsgált attribútumokhoz:

$$\text{InfoNyeréség(Vendégek)} = 1 - \left[ \frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] \approx 0,541 \text{ bit}$$

$$\text{InfoNyeréség(Konyha)} = 1 - \left[ \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0$$

Ez megerősíti azt az intuíciót, hogy a *Vendégek* jobb attribútum a tanítóhalmaz részhalmazokra szabdalására, mint a *Konyha*. Az helyzet, hogy a *Vendégek* attribútumnak van a legnagyobb információnyeresége, ezért a döntési fa tanuló algoritmus ennek tesztjét választaná a fa gyökércsomópontjába.

## A tanuló algoritmus teljesítményének becslése

Egy tanuló algoritmus akkor jó, ha olyan hipotéziseket hoz létre, amelyek jól jósolják meg az általuk előzetesen nem látott példák osztályba sorolását. A 18.5. fejezetben azt mutatjuk be, hogy mi módon lehet előre megbecsülni a jóslás minőséget. Ebben a fejezetben egy olyan módszertant vizsgálunk meg, amellyel az osztályba soroló képességet méréssel lehet becsülni.

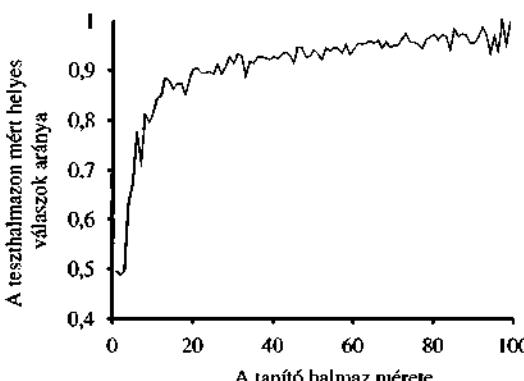
Nyilvánvalóan akkor jó egy jóslás, ha igaznak bizonyul, így a hipotézis minőségét megbecsülhetjük az ismertté vált tényleges osztálybasorolások alapján. Ezt egy **teszt-halmaznak** (test set) nevezett mintahalmaz segítségével végezhetjük el. Ha az összes

rendelkezésünkre álló példát tanításra használjuk, akkor továbbiakat kell gyűjtenünk a teszteléshez. Ezért gyakran kényelmesebb a következő módszert alkalmazni:

1. Gyűjtsük egy nagy példahalmazt.
2. Osszuk két diszjunkt részre: a **tanító halmazra** (**training set**) és a **teszthalmazra** (**test set**).
3. Alkalmazzuk a tanító algoritmust a tanító halmazon, és így generálunk egy  $h$  hipotézist.
4. Mérjük meg a teszthalmazon, hogy a  $h$  hipotézis a halmaz hány százalékára ad helyes osztálybasorolást.
5. Ismételjük meg az 1–4 lépéseket különböző tanító halmaz méretekre, és minden egyik mérethez különböző véletlenszerűen kiválasztott tanító halmazokra.

Ennek az eljárásnak az eredményeként egy adathalmazt kapunk, amelynek feldolgozásával megkaphatjuk az átlagos jóslási képességet a tanító halmaz méretének függvényében. Ezt a függvényt ábrázolva kapjuk az adott algoritmusnak egy adott témaerőre vonatkozó **tanulási görbéjét** (**learning curve**). A DÖNTÉSI-FA-TANULÁS algoritmus érterem példáinkkal felvett tanulási görbéje a 18.7. ábrán látható. Vegyük észre, hogy a tanító halmaz méretével javul a predikció minősége! (Ez okból az ilyen görbéköt ún. **boldog görbéknek** (**happy graphs**) is nevezik.) Ez annak a jele ugyanis, hogy valóban van valami mintázat az adatokban, és az algoritmus felfedezi ezt a mintázatot.

A tanuló algoritmusnak nyilvánvalóan nem szabad az előtt „látnia” a tesztadatokat, mielőtt a megtanított hipotézist teszteltük velük. Szerencsétlen módon nagyon könnyű beleesni abba a hibába, hogy tanítás közben a tesztadatokra **kukucsálunk** (**peeking**). Ez tipikusan a következő módon zajlik: egy tanuló algoritmus általában számos hangsúlyos lehetőséggel rendelkezik, amellyel az algoritmus viselkedését változtatni tudjuk. Például számos különböző kritérium lehet, amelyek alapján a következő attribútumot kiválasztjuk a döntési fa tanulásnál. Különböző beállítások mentén számos különböző hipotézist generálunk, majd mindenket leteszteljük a teszthalmazon, és a legjobb hipotézis predikciós eredményét tekintjük eredménynek. Fájdalmas bár, de kukucsálás történt! Ennek oka, hogy a hipotézist a *teszthalmazon elérte eredménye alapján választottuk ki*, így a teszthalmaz által hordozott információ beszivárgott a tanuló algo-



18.7. ábra. Tanulási görbe az érterem problémájára véletlenszerűen generált 100 példa esetén. A görbe 20 kísérlet eredményét összegzi.

ritmusba. Ennek a mesének az a tanulsága, hogy bármely eljárásnak, amelynek része a hipotéziseknek a teszthalmazon nyújtott teljesítményük alapján való összehasonlítása, egy új teszthalmazt kell használnia, amelyen a végül kiválasztott hipotézis teljesítményét méri. A gyakorlatban ez túl körülményes, így az emberek továbbra is ily módon szennyezett adathalmazaikon folytatják kísérleteiket.

## Zaj és túlilleszkedés

Korábban már láttuk, hogy ha van kettő vagy több olyan példa, amelyeknek azonos a leírásuk (az attribútumokra nézve), de eltérő az osztálybasorolásuk, akkor a DÖNTÉSI-FA-TANULÁS algoritmus nem lehet képes olyan döntési fát találni, amely minden példával konzisztens. Az általunk korábbiakban említett megoldás a következő. Ha determinisztikus osztályozásra van igény, akkor minden levélcsomópont adjon vissza a hozzá tartozó halmaz többségi osztályát. Más esetekben pedig adjon vissza a relatív gyakoriságok alapján becsült osztályba tartozási valószínűségeket. Sajnos, ez még távol van a történet végétől. Könnyen elképzelhető, sőt valójában elég valószínű, hogy a tanuló algoritmus akkor is talál egy olyan döntési fát, amely az összes példával konzisztens, ha nagyon fontos információ hiányzik. Ennek oka, hogy az algoritmus felhasználhat *irrelevant* attribútumokat, ha vannak ilyenek, hogy hamis megkülönböztetést tegyen a példák között.

Vizsgáljuk meg azt a problémát, amikor egy kockadobás eredményét akarjuk megjósolni. Tegyük fel, hogy több napon át kísérleteket végezünk számos kockával. Mind-egyik tanító példát a következő attribútumokkal írunk le:

- Nap*: az a nap, amelyen a kockadobást végeztük (hétfő, kedd, szerda, csütörtök).
- Hónap*: a hónap, amelyben a kockadobást végeztük (január vagy február).
- Szín*: a kocka színe (vörös vagy kék).

Mindaddig, amíg két dobásnak nem lesz pontosan azonos a leírása, a DÖNTÉSI-FA-TANULÁS képes lesz egzakt hipotézist találni. Minél több attribútum van, annál valószínűbb, hogy egzakt hipotézist kapunk. Viszont az összes ilyen hipotézis teljesen hamis lesz! Amit szeretnénk, az az, hogy a DÖNTÉSI-FA-TANULÁS – miután elég példát látott már – egyetlen levélcsomópontot adjon vissza, minden egyes dobási eredményhez közelítőleg 1/6 valószínűsséggel.

Amikor a hipotézisek nagy halmaza lehetséges, akkor óvatosnak kell lennünk, nehogy arra használjuk az ebből eredő nagy szabadságot, hogy értelmetlen „szabályosságot” találjunk az adatokban. Ezt a problémát **túlilleszkedésnek (overfitting)** nevezzük. Ez rendkívül általános jelenség, akkor is jelentkezhet, amikor a keresett függvénynek egyáltalán nincs valószínűségi jellege. Mindamellett ez a probléma az összes tanulási algoritmust sújtja, nem csak a döntési fákat.

A túlilleszkedés kezelésének kimerítő matematikai tárgyalása meghaladja e könyv kereteit. A probléma megoldására itt csupán egy egyszerű – **döntési fa metszés (decision tree pruning)** nevű – módszert mutatunk be. A metszés (vagy nyesés) azon az alapon működik, hogy megakadályozzuk a nem nyilvánvalóan releváns attribútumok tesztje mentén a mintahalmaz ismételt (rekurzív) felosztását, még akkor is, ha az adatok osztálybasorolása az adott csomópontban nem egyforma. Az a kérdés, hogyan vesszük észre egy attribútumról, hogy irrelevant?

Tegyük fel, hogy egy irreleváns attribútumra alapozva osztottuk ketté a mintahalmazunkat. Általánosságban szólva azt várjuk, hogy ez esetben az eredményül kapott részhalmazokban nagyjából ugyanabban az arányban fognak szerepelni az egyes osztályokba tartozó minták, mint az eredeti halmazban. Ekkor az információnyereség közel nulla.<sup>4</sup> Tehát az információnyereség (annak hiánya) jó jelzés lehet az irrelevanciára. Most az a kérdés merül fel, hogy mekkorának kell lennie az információnyereségnak ahhoz, hogy ezen attribútum mentén szétosszuk a mintahalmazt?

Ezt a kérdést a statiszkai szignifikanciaszűrő (significance test) használva válaszolhatjuk meg. Ez a teszt azzal a feltételezéssel indul, hogy a példákban egyáltalán nincs közös mintázat (ez az úgynevezett nullhipotézis). Ezek után az aktuális adathalmazt vizsgáljuk: meg akarjuk határozni annak mértékét, hogy az adathalmaz mennyire tér el a tökéletesen mintázat nélküli helyzettől. Ha az eltérés mértéke statisztikailag már valószínűleg (rendszerint 5% vagy ennél kisebb valószínűséget értünk ezalatt), akkor ezt annak bizonyítékoként vesszük, hogy az adatokban jelen van egy alapvető mintázat. A valószínűséget a véletlen mintavételezés esetén várható eltérések standard eloszlását feltételezve számítjuk ki.

Ebben az esetben a nullhipotézis az, hogy az attribútum irreleváns, ennek megfelelően egy végtelen nagy mintahalmazra vett információnyereség nulla lenne. Azt kell kiszámítanunk, hogy a nullhipotézis feltéve egy  $v$  méretű mintahalmazban a várt pozitív és negatív eseteloszlástól a megfigyelt eloszlás eltérése milyen valószínűséggel léphet fel. Az eltérés mértékét megadhatjuk a részhalmazok tényleges pozitív és negatív esetszámanak ( $p_i, n_i$ ) a nullhipotézis fennállása esetén várt esetszámokkal ( $\hat{p}_i, \hat{n}_i$ ) való összehasonlításával:

$$\hat{p}_i = p \times \frac{p_i + n_i}{p + n} \quad \hat{n}_i = n \times \frac{p_i + n_i}{p + n}$$

A teljes eltérés kényelmesen számítható mértékét a következő összefüggés adja:

$$D = \sum_{i=1}^v \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{n_i}$$

A nullhipotézist feltételezve a  $D$  eltérés mérték  $v - 1$  szabadságfokú  $\chi^2$  (khí-négyzet) eloszlást követ. Annak valószínűségét, hogy az attribútum valóban irreleváns, standard  $\chi^2$  táblázatok vagy statiszkai szoftver segítségével számíthatjuk ki. A 18.11. feladatban tüztük ki célul, hogy a DÖNTÉSI-FA-TANULÁS algoritmusban elvégezzük azokat a változtatásokat, amelyek a metszés ezen –  $\chi^2$  metszés ( $\chi^2$  pruning) néven ismert – formájához vezetnek.

A zaj metszés segítségével kezelhető: az osztályozási hibák lineáris növekedést okoznak a prediktív hibában, míg az esetleírásban fellépő hibák aszimptotikus hatást gyakorolnak, amely egyre rosszabb, ahogy a döntési fa egyre kisebb halmazok kezelésére zsugorodik. Ha az adatok nagy zajjal terheltek, akkor a metszéssel készült döntési fák lényegesen jobb eredményt adnak, mint a metszés nélkül készültek. Ráadásul a metszéssel készült fák gyakran jóval kisebbek, ezért könnyebben érhetők.

<sup>4</sup> Valójában a nyereség pozitív lesz, kivéve azt az esetet, amikor az arányok minden pontosan egyformák (lásd 18.10. feladat).

Egy másik túlilleszkedést csökkentő technika a **keresztvalidáció (cross-validation)**. Bármely tanulási eljárásnál alkalmazható, nem csupán a döntési fa tanulásnál. Az alapvető ötlet annak megbecsülése, hogy az egyes hipotézisek mennyire jól fogják megjósolni a még nem látott esetekre adandó válaszokat. Ez így becsülhetjük, hogy az ismert adatok egy részét félretestessük, és ezekkel teszteljük a megmaradt adatok alapján tanulással létrehozott fa predikciós képességét.  $K$ -szoros kereszтvalidációban nevezünk, ha  $k$  kísérletet végzünk, és minden esetben az adatok más és más  $1/k$ -ad részét tesszük felre validációs tesztcélra, majd a végén átlagoljuk az eredményeket. Az 5 és a 10 elterjedten használt  $k$  értékek. Szélső esetként használják a  $k = n$  választást, amelyet hagyj-kí-egyet kereszтvalidációs módszerként ismerünk. A kereszтvalidációt tetszőleges döntési fa tanulási módszerrel együtt alkalmazhatjuk (beleértve a metszést is), célja minden az, hogy olyan döntési fa kiválasztását segítsük elő, amely jó predikciós képességgel rendelkezik. A kukucsálási jelenség elkerülése végett ezek után ezt a predikciós képességet egy új teszthalazon kell mérnünk.

## A döntési fák alkalmazhatóságának kiterjesztése

Egy sor problémát meg kell oldanunk ahhoz, hogy a döntési fa indukciót a problémák szélesebb körére kiterjeszthessük. Nagyjából felvázoljuk mindegyiket, azt javasolta, hogy a teljesebb megértés érdekében a kapcsolódó feladatokat oldja meg az olvasó.

- Hiányzó adatok (missing data):** Számos területen nem ismerhető meg minden példa összes attribútuma. Lehet, hogy az értékek nem kerültek tárolásra, vagy túl drága lenne a mérésük. Ez két problémát vet fel: először, ha van is egy teljes döntési fánk, akkor hogyan tudunk egy olyan mintát osztályozni, amelynek egy tesztelendő attribútuma hiányzik? Másodszor, hogyan módosítsuk az információnyereségre vonatkozó formulát, ha néhány példa ezen attribútumának értéke nem ismert? Ezeket a kérdéseket a 18.2. feladatban vizsgáljuk.
- Sokértékű attribútumok (multivalued attributes):** Ha egy attribútum nagyszámú értéket vehet fel, akkor az információnyereség nagysága nem megfelelő mértéke az attribútum hasznosságának. Szélsőséges esetben egy olyan attribútumot használnánk, mint például az ÉtteremNév attribútum, amely minden példára más és más értéket ad. Ez esetben példák minden részhalmaza egyelemű lesz, egyedi osztálybansorolással, így az információnyereség erre az attribútumra veszi fel maximális értékét. Mindamellett ez az attribútum irreleváns, értéktelen. Egy lehetséges megoldás a nyereségarány (gain ratio) használata (lásd 18.13. feladat).
- Folytonos és egész értékű bemeneti attribútumok (continuous and integer-valued input attributes):** A folytonos vagy az egész értékű bemeneti attribútumok – mint például a Magasság és a Súly – által felvethető értékek halmaza végtelen. A döntési fa tanuló algoritmusok az ilyen esetekre nem generálnak végtelen elágazási csomópontokat, inkább megkeresik azt a küszöbpontot (split point), amely a legnagyobb információnyereséget eredményezi. Például egy adott csomópontnál az lehet a helyzet, hogy a Súly  $> 160$  teszt adja a legnagyobb információnyereséget. Léteznek hatékony dinamikus programozási módszerek jó küszöbpontok megtalálására, de még

mindig messze ez a legtöbb erőforrást igénylő része a döntési fa tanulási eljárások valós problémákra való alkalmazásának.

- **Folytonos értékkészletű kimeneti attribútumok (continuous-valued output attributes):** Ha egy numerikus értéket akarunk megjósolni, például egy műtárgy árát, akkor nem osztályozásra van szükségünk, hanem egy regressziós fára (**regression tree**). Egy ilyen fa nem egyetlen értéket (osztályt) ad vissza az egyes levél csomópontjai- ban, hanem egy attribútumhalmaz lineáris függvényét. Például a műtárgyak regreszsiós fájának kézzel festett metszetekre vonatkozó ága a területnek, a kornak és a színek számának lineáris függvényéhez vezethet. A tanuló algoritmusnak kell azt megoldania, hogy mikor álljon le a minták csomópontokban történő szétosztásával, áttérve a maradék attribútumokat (vagy azok bizonyos részhalmazát) felhasználó lineáris regresszió megalkotására.

Egy döntési fa tanulásra szolgáló, valós problémák megoldását célzó rendszereknek képesnek kell lennie mindenben problémák kezelésére. A folytonos értékű változók kezelése különösen fontos, mivel mind a fizikai, mind a gazdasági folyamatok numerikus értékekkel jellemzőek. Számos, üzleti forgalomban kapható programcsomag készült, amelyek megfelelnek mindenben a kritériumoknak, és segítségükkel több száz – valamilyen konkrét területen – alkalmazott rendszert fejlesztettek ki. Az ipar és kereskedelem sok területén, ha minták alapján kialakított osztályozásra van szükség, akkor elsősorban döntési fákkal próbálkoznak. A döntési fák egyik fontos tulajdonsága, hogy az ember számára jól érthető a tanuló algoritmus által előállított eredmény. (Valójában olyan gazdasági döntések esetén, amelyeknek megkülönböztetés elleni jogszabályoknak kell megfelelniük, ez *jogi követelmény*.) A neurális hálózatoknál hiányzik ez a fontos tulajdonság (lásd 20. fejezet).

## 18.4. HIPOTÉZISHALMAZ EGYÜTTES TANULÁSA

Az eddigiekben olyan tanulási algoritmusokat vizsgáltunk, amelyek eredményeként a hipotézistérből kiválasztott egyetlen hipotézist használunk arra, hogy predikciót végezzünk. Az **együttes tanulás (ensemble learning)** alapötlete, hogy válasszunk ki a hipotézistérből egy teljes hipotéziskollekción vagy hipotézisegyüttést (**ensemble**), és kombináljuk az általuk adott predikciókat. Generálhatunk például ugyanazon tanító halmaz alapján száz különböző döntési fát, és egy új példa esetén szavazással alakíthatjuk ki a legjobb predikciós osztályozási eredményt.

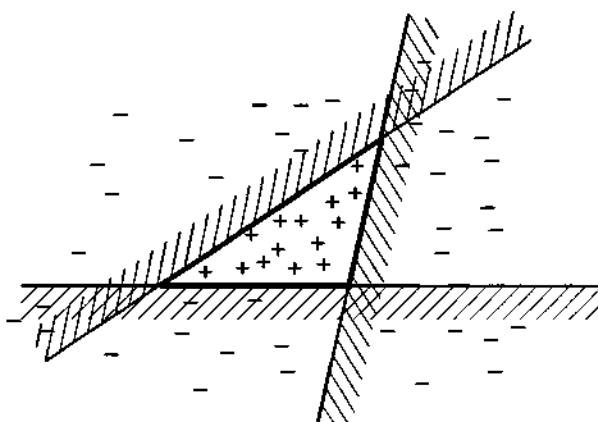
Az együttes tanulás motivációja kézenfekvő. Vizsgálunk egy  $M = 5$  hipotézisból álló együttest, és tegyük fel, hogy a predikciós eredményeket egyszerű többségi szavazással kombináljuk össze. Ahhoz, hogy az együttes rosszul osztályozzon egy új példát, *legalább háromnak az öt hipotézisból rossz osztályozási eredményt kell adnia!* Reményünk szerint ez sokkal kevésbé valószínű, mint az, hogy egyetlen hipotézis rossz osztályozásra jusson. Tegyük fel, hogy az együttes minden  $h_i$  hipotézisének  $p$  a hibavalószínűsége, azaz  $p$  annak valószínűsége, hogy  $h_i$  rosszul osztályoz egy véletlen módon kiválasztott példát. Továbbá tegyük fel, hogy a hipotézisek által elkövetett hibák *függetlenek*. Ez esetben – ha  $p$  kicsi – rendkívül kicsi a valószínűsége, hogy nagy számban történék téves osztályozás. Egyszerű számítással (lásd 18.14. feladat) megmutatható például, hogy öt hipotézisból

tézisből álló együttest használva a hibaarány 0,1-ről kevesebb mint 0,01-ra csökken. Valójában a függetlenség feltételezése nem racionális, mert a hipotéziseket a tanító halmaz félrevezető tulajdonságai hasonló módon vezetik félre. Mindamellett, ha a hipotézisek legalább kismértékben eltérőek, lecsökkentve a hibáik közti korrelációt, akkor az együttes tanulás nagyon hasznos lehet.

Az együttes tanulás tárgyalható más aspektusból is: gondolhatunk rá úgy, mint a hipotézistér egy általános kiterjesztésére. Tekintsünk úgy magára az együttesre, mint egy hipotézisre, és az új hipotézistér legyen minden mintához minden lehetséges együttesek halmaza, amelyek az eredeti hipotézistérből konstruálhatók. A 18.8. ábra mutatja, hogyan eredményezhet ez egy sokkal kifejezőbb hipotézisteret. Ha az eredeti hipotézistér egyszerű és hatékony tanulási eljárást tett lehetővé, akkor az együttes tanulás módszere egy sokkal kifejezőbb hipotézisosztály tanulását biztosítja. Ennek általában nem túl nagy számítási és algoritmikus komplexitás növekedés az ára.

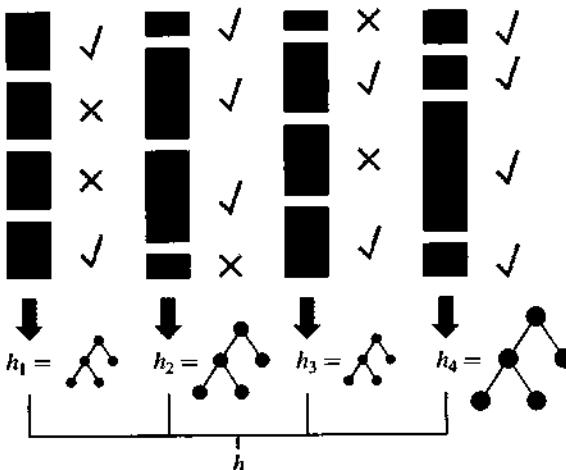
A legelterjedtebben használt együttes tanulási módszert **turbózásnak (boosting)** nevezzük. Ahhoz, hogy működését megértsük, be kell vezetnünk a **súlyozott tanító halmaz** (**weighted training set**) fogalmát. Egy ilyen tanító halmazban minden mintához hozzárendelünk egy  $w_j \geq 0$  súlyt. Minél magasabb ez a súly, annál nagyobb jelentőséget tulajdonítunk az adott mintának a hipotézis tanulása során. Az eddigiekben áttekinntett tanuló algoritmusaink kézenfekvő módon módosíthatók úgy, hogy súlyozott tanító halmazokkal is tudjanak működni.<sup>5</sup>

A turbózás az összes mintára  $w_j = 1$  értékkel (azaz egy normál tanító halmazzal) indul. Ebből a halmazból generálja az első hipotézist,  $h_1$ -et. Ez a hipotézis egyes tanító-



**18.8. ábra.** A együttes tanulás eredményeképpen nyert megnövekedett kifejezőről illusztráló példa. Három lineáris küszöb jellegű hipotézist vizsgálunk, mindenkor a nem vonalkázott félsíkon ad pozitív besorolást. Az együttesben csak arra adunk pozitív osztályozást, amelyre minden pozitívát ad. Az eredményül kapott háromszögterület egy olyan hipotézisnek felel meg, amelyet az eredeti hipotézistér egyetlen eleme sem képes kifejezni.

<sup>5</sup> Azon tanulási eljárásokra, ahol ez nem lehetséges, **ismétléses tanító halmazt (replicated training set)** hozhatunk létre, ahol a  $j$ -edik minta a  $w_j$  súlyjal arányos példányszámban szerepel. Az esetleges nem egész súlyokat úgy kezelhetjük, hogy a mintaszámot valószínűségi változóként fogjuk fel.



**18.9. ábra.** A turbó algoritmus működése. Mindegyik árnyékolt téglalap egy példának felel meg, a téglalap magassága mutatja a példa súlyát. A pipák és keresztek mutatják, hogy a példát jól vagy rosszul osztályoztuk a pillanatnyi hipotézis alapján. A döntési fa mérete mutatja, hogy ennek a hipotézisnek mekkora a súlya a végső együttesben.

mintákat jól osztályoz, másokat rosszul. Azt szeretnénk, ha a következő hipotézis jobb eredményt érne el a rosszul osztályozott mintákon, ezért megnöveljük a rosszul osztályozottak súlyait, míg a jól osztályozottakét csökkentjük. Ebből az új, súlyozott mintahalmazból generáljuk a  $h_2$  hipotézist. Az eljárást addig folytatjuk, míg  $M$  hipotézist nem generáltunk, ahol  $M$  a turbó tanulási algoritmus bemenő paramétere. Az algoritmus eredményeként kapott hipotézis együttes nem más, mint az  $M$  hipotézis súlyozott többségi szavazással kombinált eredménye. A súlyokat aszerint határozzuk meg, hogy az egyes hipotézisek mennyire jól teljesítettek a tanító halmazon. A 18.9. ábrán bemutatjuk az algoritmus koncepcióját. Sok variánsa ismert a turbó tanulásnak, eltérő súlymódosítással, illetve a hipotézisek különböző kombinálási lehetőségeivel. Egy kiválasztott – AdaBoost nevű – algoritmust mutatunk be a 18.10. ábrán. Az alkalmazott súlymódosítási eljárás jelen esetben nem túl lényeges, viszont az AdaBoost algoritmusnak van egy nagyon lényeges jellemzője. Tegyük fel, hogy a használt  $L$  tanulási algoritmus egy **gyenge tanulási (weak learning)** algoritmus, ami azt jelenti, hogy  $L$  minden visszaad egy olyan hipotézist, amelyik legalább egy csöppet jobban teljesít a tanító halmaz súlyozott hibájára nézve, mint a véletlen találgatás (azaz eredménye jobb mint 50% bináris osztályozás esetén). Ez esetben, kellően nagy  $M$ -re az AdaBoost minden előállít egy olyan hipotézist, amely a tanító mintákat tökéletesen osztályozza! Tehát az algoritmus **fokozza** az eredeti tanuló eljárásnak a tanító mintán mért pontosságát. Ez minden fennáll, attól függetlenül, hogy esetleg mennyire kevésbé kifejező az eredeti hipotézis-tér, és mennyire bonyolult a keresett függvény.

Vizsgáljuk meg, hogy mennyire jól működik a turbózás az éttermi adatainkon. Kiinduló hipotézistérként az úgynevezett **döntési tönkök (decision stumps)** terét választjuk, amelyek olyan döntési fák, amelyek csupán egy – a gyökérkocsomópontban elhelyezkedő – tesztből állnak. A 18.11. (a) ábra alsó görbéje mutatja, hogy turbózás nélkül

```

function ADABoost(példák, L, M) returns egy súlyozott többségi hipotézis
  inputs: példák, minták N elemű címkézett halmaza ( $x_1, y_1$ ), ..., ( $x_N, y_N$ )
    L, egy tanuló algoritmus
    M, az együttesben szereplő hipotézisek száma
  local variables: w, az N mintához rendelt súlyok vektora, indulásnál  $1/N$ 
    h, az M darab hipotézisből képzett vektor
    z, az M hipotézishez rendelt súlyok vektora
  for m = 1 to M do
    h[m]  $\leftarrow L(\text{példák}, \mathbf{w})$ 
    error  $\leftarrow 0$ 
    for j = 1 to N do
      if h[m]( $x_j$ )  $\neq y_j$  then error  $\leftarrow error + w[j]$ 
    for j = 1 to N do
      if h[m]( $x_j$ )  $\neq y_j$  then w[j]  $\leftarrow w[j] \cdot error/(1 - error)$ 
    w  $\leftarrow \text{NORMALIZÁL}(\mathbf{w})$ 
    z[m]  $\leftarrow \log(1 - error)/error$ 
  return SÚLYOZOTT-TÖBBSÉG(h, z)

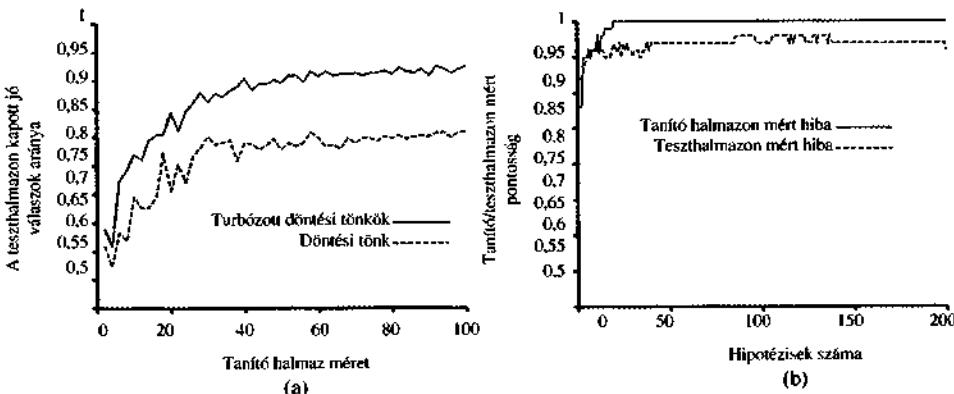
```

**18.10. ábra.** Az együttes tanulás turbó módszereinek ADABoost nevű variánsa. Az algoritmus hipotézisek kombinációját hozza létre. A SÚLYOZOTT-TÖBBSÉG egy hipotézist generál. Ez az új hipotézis azt a ki-meneti értéket adja vissza, amely a *h*-ban található hipotézisek közül, a *z*-ben számon tartott súlyozással a legtöbb szavazatot kapta.

ezek a döntési tönök nem adnak valami jó eredményt, csupán 81%-os a 100 mintán mért predikciós teljesítményük. Amikor turbózást alkalmazunk (*M* = 5), akkor a teljesítmény javul, eléri a 93%-ot 100 példa esetén.

Érdekes dolog történik, amikor az együttes *M* mérete növekszik. A 18.11. (b) ábra mutatja a tanító halmazon mért teljesítményt *M* függvényében. Figyeljük meg, hogy *M* = 20-nál a hiba eléri a nullát (ahogy a turbózással foglalkozó elrnélet megjósolta), azaz 20 döntési tönk súlyozott többségi szavazással összefogott kombinációja elégseges a 100 mintára való pontos illeszkedéshez. Ha további tönköket adunk az együtteshez, akkor a hiba nulla marad. A görbe mutatja ugyanakkor, hogy a teszthalmazon mért teljesítmény még hosszan nő azután is, hogy a tanító halmazon mért hiba elérte a nullát. *M* = 20 esetén a teszthalmazon mért teljesítmény 0,95 (a hiba 0,05), *M* = 137-ig növelve az együttes résztvevőinek számát a teljesítmény 0,98-ra nő, ezután leesik 0,95-re.

Ez a megfigyelés, amely különböző mintahalmazokra és hipotézisterekre nézve meglehetősen robusztusnak bizonyult, nagy meglepetést keltett a felfedezések. Az Ockham borotvája elv azt sugallja, hogy ne tegyük a hipotéziseket komplexebbé, mint szükséges, de a görbe azt mutatja, hogy a predikciók *javulnak*, ahogy az együttes hipotézis összetettebbé válik! Számos magyarázatot javasoltak erre a jelenségre. Az egyik megközelítés szerint a turbózás nem más, mint a Bayes-tanulás közelítése (lásd 20. fejezet), amely egy bizonyíthatóan optimális tanulási eljárás. Ez a közelítés javul, ahogy több és több hipotézist adunk az együtteshez. Másik lehetséges magyarázat, hogy további hipotéziseknek az együtteshez adása lehetővé teszi, hogy *határozottabban* legyen a megkülönböztetés a pozitív és negatív példák között, ami segít az új minták osztályozásában.



**18.11. ábra.** (a) A döntési tönkök és a turbózott döntési tönkök teljesítményének az éterem példán törtenő összehasonlítása  $M = 5$  esetén. (b) A tanító halmazon és a teszthalmazon kapott jó válaszok aránya  $M$ -nek – az együttesben található hipotézisek számának – a függvényében. Figyeljük meg, hogy a teszthalmazon mért pontosság még azután is javul egy kicsit, amikor a tanító halmazon elérte az 1-et, azaz miután az együttes pontosan illeszkedett az adatokra.

## 18.5. MIÉRT MŰKÖDIK A TANULÁS: A TANULÁSI SZÁMÍTÁSI ELMÉLETE

A legfontosabb megválaszolatlan kérdés, amelyet a 18.2. alfejezetben tettünk fel, a következő: hogyan bizonyosodhat meg valaki arról, hogy a tanulási algoritmusa által létrehozott elmélet helyesen fogja megjósolni a jövőt? Formálisan: honnan tudjuk, hogy a  $h$  hipotézis jól közelíti az  $f$  keresett függvényt, ha nem ismerjük  $f$ -et? Évszázadok óta foglalkoznak ezekkel a kérdésekkel. Amíg nem találjuk meg a válaszokat, addig a gépi tanulást – a legjobb esetben is – csak zavarba hozza saját sikereit.

Az ebben az alfejezetben tárgyalt megközelítés a **tanulás számítási elméletén** (**computational learning theory**) alapul, amely az MI, a statisztika és az elméleti számítástudomány közös határterülete. Ennek az elméletnek az alapvető elve a következő: *bármely súlyosan hibás elmélet szinte bizonyosan nagy valószínűséggel felismerhető kisszámú példa vizsgálata alapján, mivel helytelen predikciót fog adni. Tehát minden hipotézis, amely egy kielégítően nagy tanító példahalmazzal konziszens választ ad, nem valószínű, hogy súlyosan hibás lenne: azaz valószínűleg közelítőleg helyesnek (*probably approximately correct*) kell lennie.* Minden olyan tanuló algoritmust, amely valószínűleg közelítőleg helyes hipotéziseket ad, **VKH-tanuló (PAC-learning)** algoritmusnak nevezünk.

Van néhány bonyolultabb rész az előbbi gondolatmenetben. A fő kérdés a tanító és a tesztpéldák viszonya: végül is mi azt szeretnénk, ha a hipotézis nem csupán a tanító halmazon lenne közelítőleg helyes, hanem elsősorban a teszthalmazon. Az alapvető feltevés az, hogy a tanító és a teszthalmazt *ugyanolyan valószínűség-eloszlást* használva, véletlenszerűen és függetlenül választottuk ugyanabból a példapopulációból. Ezt **stacionaritási (stationarity)** feltevésnek nevezzük. A stacionaritási feltevés nélkül az elmélet semmilyen, a jövőre vonatkozó igénynyel nem léphet fel, mivel nem lesz szükségszerű kapcsolat a múlt és a jövő között. A stacionaritási feltevés támásztja alá

azt a gondolatot, hogy a példákat kiválasztó eljárás nem rosszindulatú. Persze ha a tanító halmaz kizárolag furcsa példákat tartalmaz – kétfejű kutyákat például –, akkor a tanuló algoritmus nyilvánvalóan nem tehet mászt, mint sikertelen általánosításra jut a felől, hogy mi módon lehet a kutyákat felismerni.

## Hány példára van szükség?

Néhány definícióra szükségünk lesz ahhoz, hogy ezeket a felismeréseket a gyakorlatba át tudjuk ültetni:

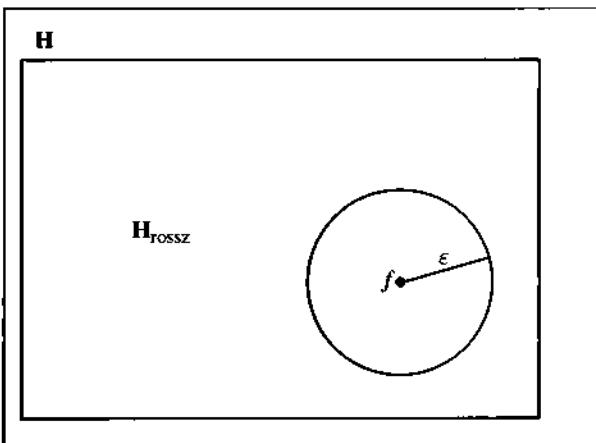
- Jelöljük  $X$ -szel az összes lehetséges példák halmazát.
- Jelölje  $D$  azt a valószínűségi eloszlást, amely alapján a példákat választjuk.
- Legyen  $H$  a lehetséges hipotézisek halmaza.
- Legyen  $N$  a tanító halmaz elemeinek a száma.

Előzetesen feltesszük, hogy a keresett  $f$  függvény eleme  $H$ -nak. Most már definiálhatjuk azt a hibát (**error**), ami egy  $h$  hipotézisnek a keresett  $f$  függvénytől való eltérését jellemzi a mintákra vonatkozó adott eloszlás mellett, mint annak valószínűségét, hogy  $h$  az  $f$ -től eltérő választ ad egy példára:

$$\text{error}(h) = P(h(x) \neq f(x) | x \text{ a } D \text{ alapján választva})$$

Ez ugyanaz a mennyiség, mint amelyet az előzőkben bemutatott tanulási görbükkel mértünk a gyakorlatban.

Egy  $h$  hipotézist közelítőleg helyesnek (**approximately correct**) nevezünk, ha  $\text{error}(h) \leq \epsilon$ , ahol  $\epsilon$  egy kis konstans. A bizonyítás megközelítése az, hogy megmutatjuk:  $N$  példa vizsgálata után az összes konziszens hipotézis nagy valószínűséggel közelítőleg helyes lesz. Úgy gondolhatunk egy közelítőleg helyes hipotézisre, mint amely közel van a keresett függvényhez a hipotézistérben: egy – a keresett  $f$  körül felvett –  $\epsilon$ -gömbön ( $\epsilon$ -ball) belül van. A 18.12. ábrán bemutatjuk az összes hipotézis  $H$  terét,



18.12. ábra. A hipotézistér sematikus ábrázolása a keresett  $f$  függvény körül felvett „ $\epsilon$ -gömb” ábrázolásával

amelyet két részre osztottunk, egyik az  $\varepsilon$ -gömb  $f$  körül, a másik a  $\mathbf{H}_{\text{rossz}}$ -szal jelölt műradék.

A következők alapján kiszámíthatjuk annak valószínűségét, hogy egy „súlyosan hibás”  $h_r \in \mathbf{H}_{\text{rossz}}$  hipotézis konzisztens lesz az első  $N$  példával. Tudjuk, hogy  $\text{error}(h_r) > \varepsilon$ . Tehát annak valószínűsége, hogy egy adott példára helyes választ ad, legfeljebb  $1 - \varepsilon$ . Ezek után az  $N$  példára vonatkozó összefüggés:

$$P(h_r \text{ konzisztens } N \text{ példával}) \leq (1 - \varepsilon)^N$$

Annak valószínűségét, hogy  $\mathbf{H}_{\text{rossz}}$  legalább egy konzisztens hipotézist tartalmaz, a lehetséges egyedi hipotézisek száma határozza meg:

$$P(\mathbf{H}_{\text{rossz}} \text{ tartalmaz egy konzisztens hipotézist}) \leq |\mathbf{H}_{\text{rossz}}| (1 - \varepsilon)^N \leq |\mathbf{H}| (1 - \varepsilon)^N$$

Kihasználtuk, hogy  $|\mathbf{H}_{\text{rossz}}| \leq |\mathbf{H}|$ . Ezt a valószínűséget valamely kellően kis  $\delta$  érték alá akarjuk csökkenteni:

$$|\mathbf{H}| (1 - \varepsilon)^N \leq \delta$$

Felhasználva, hogy  $(1 - \varepsilon) \leq e^{-\varepsilon}$ , a célunk elérhető, ha lehetővé tesszük, hogy az algoritmus

$$N \geq \frac{1}{\varepsilon} \left( \ln \frac{1}{\delta} + \ln |\mathbf{H}| \right) \quad (18.1)$$

példát megvizsgálhasson. Ennek értelmében, ha egy algoritmus ilyen mennyiségi példával konzisztens hipotézist ad vissza, akkor legalább  $1 - \delta$  valószínűséggel a hibája legfeljebb  $\varepsilon$ . Más szavakkal közelítőleg helyesnek nevezzük. A szükséges példaszámot, ami  $\varepsilon$  és  $\delta$  függvényeként adható meg, a hipotézistér minta komplexitásának (**sample complexity**) nevezzük.

Kiderült, hogy kulcskérdes a hipotézistér mérete. Mint korábban láttuk, ha  $\mathbf{H}$  az  $n$  attribútumon felvethető Boole-függvények halmaza, akkor  $|\mathbf{H}| = 2^n$ . Tehát a tér minta komplexitása  $2^n$  szerint nő. Mivel a lehetséges példák száma szintén  $2^n$ , ebből az következik, hogy a Boole-függvények terében egyetlen tanuló algoritmus sem tud jobb eredményt elérni, mint egy táblázat, ha csupán arra szorítkozik, hogy egy olyan hipotézist adjon vissza, amely az összes ismert példával konzisztens. Ennek meghatározására egy másik lehetőség az, hogy megfigyeljük: egy tetszőleges ismeretlen példára a hipotézistér ugyannyi pozitív kimenetelt jósoló konzisztens hipotézist tartalmaz, mint amennyi negatív kimenetelt jósol.

A következő dilemmával kerültünk szembe: ha nem korlátozzuk a szóba jövő függvények terét, akkor az algoritmus nem lesz képes tanulni, viszont ha korlátozzuk a függvények terét, akkor fennáll a veszély, hogy kihagyjuk a keresett függvényt. Két lehetőségünk van, hogy „kimeneküljünk” ebből a dilemmából. Az első, hogy nem csupán ahhoz ragaszkodunk, hogy az algoritmus egy tetszőleges konzisztens hipotézist adjon vissza, hanem ahhoz is, hogy részesítse előnyben az egyszerűeket (mint ahogy a döntési fa tanulásnál tettek). Ezen algoritmusok elméleti tárgyalása túlmegy ennek a könyvnek a keretein, de megemlíjtük, hogy azon esetekben, amikor az egyszerű konzisztens hipotézisek keresése kezelhető probléma, a mintakomplexitás-eredmények jobbak, mint a csupán konziszción alapuló vizsgálatoké. A másik menekülési lehetőség, amelyet követni is fogunk, hogy a Boole-függvények teljes halmazából a megtanulható függvények részhalmazára

fogunk koncentrálni. Az ötlet abban áll, hogy legtöbb esetben nincs szükségünk a Boole-függvények teljes kifejezőjére, korlátozottabb nyelvek is kielégítők számunkra. A következőkben egy ilyen korlátozottabb nyelvet vizsgálunk meg részletesebben.

## Döntési listák tanulása

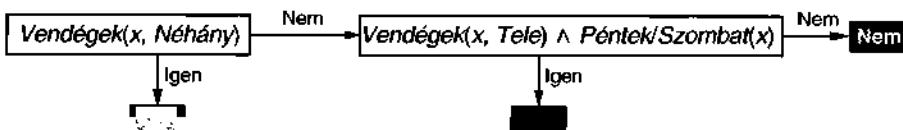
A **döntési lista** (decision list) egy kötött formájú logikai kifejezés. Egy tesztsorozatból áll, amely tesztek mindegyike literálisok konjunkciója. Ha egy teszt valamelyik példa leírására alkalmazva pozitív eredményt ad, akkor a döntési lista határozza meg a példára adandó választ. Ha a teszt eredménye negatív, akkor a lista következő teszttjével folytatódik a feldolgozás.<sup>6</sup> A döntési listák emlékeztetnek a döntési fákra, de a globális struktúrájuk egyszerűbb, ezzel szemben az egyes tesztek bonyolultabbak, mint a döntési fánál. A 18.3. ábra egy döntési listát mutat be, amely a következő hipotézist reprezentálja:

$$\forall x \text{ VárunkE}(x) \Leftrightarrow \text{Vendégek}(x, \text{Néhány}) \vee (\text{Vendégek}(x, \text{Tele}) \wedge \text{Péntek/Szombat}(x))$$

Ha megengedünk tetszőleges méretű teszteket, akkor a döntési listák bármely Boole-függvény reprezentálására képesek (lásd 18.15. feladat). Másrészről, ha az egyes tesztek méretét  $k$  literálisra korlátozzuk, akkor lehetővé válik, hogy a tanuló algoritmus kisszámú példa alapján sikeresen általánosítson. Ezt  $k$ -DL nyelvnek nevezzük. Könnyen megmutatható, hogy a  $k$ -DL nyelvek részhalmazként tartalmazzák a  $k$ -DF nyelveket, amelyek a legfeljebb  $k$  mélységű döntési fák halmazát alkotják. Fontos, hogy egy adott nyelv, amelyre  $k$ -DL jelöléssel hivatkozunk, függ azoktól az attribútumoktól, amelyekkel a példákat leírjuk. A továbbiakban  $k$ -DL( $n$ ) lesz a jelöléstünk azokra a  $k$ -DL nyelvekre, amelyek  $n$  logikai attribútumot használnak.

Első feladatunk annak megmutatása, hogy a  $k$ -DL nyelvek megtanulhatók, azaz bármely  $k$ -DL függvény pontosan közelíthető, ha kellő számú tanító mintán tanultunk. Ehhez ki kell számítanunk a nyelvben felvehető hipotézisek számát. Legyen a tesztek nyelve –  $n$  attribútumot használva, legfeljebb  $k$  literális konjunkciója –  $\text{Conj}(n, k)$ . A döntési listákat tesztekből építjük fel, és minden teszthez egy *Igen* vagy *Nem* választ rendelhetünk, illetve harmadik lehetőséggént a teszt hiányozhat a listáról, ezért legfeljebb  $3^{|\text{Conj}(n, k)|}$  különböző teszthalmaz létezhet. Ezek a tesztek tetszőleges sorrendben alkalmazhatók, így

$$|k\text{-DL}(n)| \leq 3^{|\text{Conj}(n, k)|} |\text{Conj}(n, k)|!$$



18.13. ábra. Az éttermi problémára alkotott döntési lista

<sup>6</sup> Egy döntési lista tehát szerkezetében megegyezik a Lisp nyelv cond állításával.

Az  $n$  attribútumot használó  $k$  literális lehetséges konjunkcióinak száma:

$$|\text{Conj}(n, k)| = \sum_{i=0}^k \binom{2n}{i} = O(n^k)$$

Így néhány lépés után:

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}$$

Ezt a (18.1) egyenletbe helyettesítve megmutathatjuk, hogy a  $k$ -DL függvények VKH-tanulásához szükséges minták száma polinomiálisan nő  $n$ -nel:

$$N \geq \frac{1}{\varepsilon} \left( \ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right)$$

Ennek értelmében bármely algoritmus, amely konzisztens döntési listát ad vissza, kis  $k$  értékek esetén elfogadható példaszám alapján VKH-tanulással képes előállítani egy  $k$ -DL függvényt.

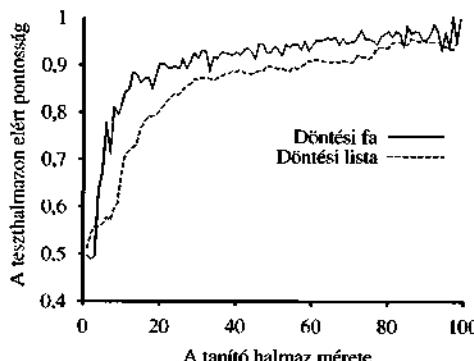
A következő feladatunk, hogy hatékony algoritmust találunk, amely konzisztens döntési listát eredményez. A DÖNTÉSI-LISTA-TANULÁS nevű mohó algoritmust fogjuk használni. Ez ismételten talál egy olyan tesztet, amely pontosan a tanító halmaz valamely részhalmazának felel meg. Amikor talál egy ilyen tesztet, akkor azt hozzáadja a döntési listához, egyben eltávolítva a tesztnak megfelelő példákat a tanító halmazból. Ezek után létrehozza a döntési lista hátralevő részét pusztán a maradék példákra alapozva. Ezt addig ismétli, amíg nem marad több példa. Az algoritmust a 18.14. ábra mutatja.

Ez az algoritmus nem specifikálja, hogy milyen módszert használunk a következő – a döntési listához adni kívánt – teszt kiválasztásához. Bár a bemutatott formálisan elért eredmények nem függnek a kiválasztás módszerétől, mégis az látszik józannak, ha előnyben részesítjük azokat az egyszerű teszteket, amelyek az osztályozott példák nagy részhalmazainak felelnek meg. Ennek eredményeként a döntési lista a lehető legtömörebb lesz. A legegyszerűbb stratégia, ha azt a legrövidebb  $t$  tesztet keressük meg, amely egy tetszőleges, egyforma osztálybasorolással rendelkező részhalmaznak megfelel, figyelmen kívül hagyva a részhalmaz méretét. Még ez a megközelítés is egész jól működik, mint az a 18.15. ábrán látható.

```

function DÖNTÉSI-LISTA-TANULÁS(példák) returns egy döntési lista vagy kudarc
  if példák üres then return a triviális döntési lista, Nem
  t  $\leftarrow$  egy teszt, amely pontosan megfelel a példák egy példák, nemüres részhalmazának, azaz a példák,
    minden elemére a teszt vagy pozitív, vagy negatív értéket ad
  if nincs ilyen t then return kudarc
  if példák, elemeire a teszt pozitív then o  $\leftarrow$  Igen else o  $\leftarrow$  Nem
  return egy döntési lista, amelynek első eleme t, a válasz o, és a maradék teszteket a
    DÖNTÉSI-LISTA-TANULÁS(példák – példák) adja
  
```

18.14. ábra. Egy döntési lista tanulásra használható algoritmus



**18.15. ábra.** A DÖNTÉSI-LISTA-TANULÁS algoritmusnak az étermi adatokon felvett tanulási görbéje. Az összehasonlítás kedvéért feltüntettük a DÖNTÉSI-FA-TANULÁS algoritmus görbékét is.

## Elemzés

A tanulás számítási elmélete új szempontból világította meg a tanulás problémáját. Az 1960-as évek elején a tanulás elmélet az **identifikáció határátmenetben (identification in the limit)** problémájára koncentrált. E szerint egy identifikációs algoritmusnak mindenkorábban vissza kell adnia egy olyan hipotézist, amely pontosan megegyezik a keresett függvényt. Erre a következőkben ismertetett eljárás ad egy lehetséges módszert. Először is rendezzük a H hipotézistér minden elemét valamelyen egyszerűség mérték szerint. Ezek után válasszuk azt a legegyszerűbb hipotézist, amely konzisztens az összes már rendelkezésre álló példával. Ahogy újabb példák érkeznek, a módszer fel fogja adni az egyszerűbb hipotézist, ha az új példák érvénytelenítették, és helyette egy bonyolultabbat választ. Amikor eléri a keresett függvényt, akkor azt már soha nem fogja feladni. Sajnálatos, hogy számos hipotézistérben óriási nagy lehet a keresett függvény eléréséhez szükséges idő, valamint a szükséges példák száma. Ezért a tanulás számítási elmélete nem ragaszkodik ahhoz, hogy a tanuló ágens megtalálja a környezetét vezérlő „egyedi helyes törvényt”. Ehelyett inkább egy olyan hipotézist keres, amely egy bizonyos prediktív pontosságot elér. A tanulás számítási elmélete ezen felül erősen hangsúlyozza a hipotézis nyelv kifejezőképessége és a tanulás komplexitása közötti kompromisszum fontosságát. Ez az elmélet vezetett egy fontos tanuló algoritmus osztályhoz, a support vektorgépekhez.

Az általunk bemutatott, VKH-tanulásra vonatkozó eredmények a legrosszabb esetre érvényesek, nem feltétlenül mutatva azt az átlagos esetre vonatkozó mintakomplexitást, amelyeket viszont a bemutatott tanulási görbék tükröznek. Egy, az átlagos esetre vonatkozó analízisnek további feltételezésekkel kell elnie a minták eloszlásáról, valamint a keresett függvények eloszlásáról. Miközben ezeket a kérdéseket egyre jobban megérjük, a tanulás számítási elmélete folyamatosan a gépi tanulással foglalkozó kutatók hasznos útmutatójának bizonyul. Ezek a kutatók algoritmusaiak módosításában vagy algoritmusaiak tanulási képességének meghatározásában érdekeltek. A döntési listák mellett a Boole-függvények szinte minden ismert alosztályára, a neurális hálókra (lásd 20. fejezet), valamint az elsőrendű logikai állítások halmazaira (lásd 19. fejezet) születtek eredmények. Az elérte eredmények azt mutatják, hogy a tisztán induktív tanulás

rendkívül nehéz. Tisztán induktíos tanuláson azt a helyzetet értjük, amelyben az ágens nem rendelkezik semmilyen, a keresett függvényre vonatkozó, előzetes ismerettel. Mint a 19. fejezetben megmutatjuk, az előzetes tudásnak az induktív tanulás vezérlésére való felhasználása lehetővé teszi, hogy meglehetősen nagy állításhalmazokat elfogadható mintaszám alapján megtanulunk, még egy olyan nyelven is, amelynek kifejezőereje megegyezik az elsőrendű logikával.

## 18.6. ÖSSZEFOGLALÁS

Ebben a fejezetben determinisztikus függvények példák alapján való induktív tanulására koncentráltunk. A főbb pontok a következők voltak:

- A tanulásnak számos formája van, amely a cselekvő alrendszer jellegétől, a javítani kívánt komponenstől és a rendelkezésre álló visszacsatolástól függ.
- Ha akár egy tanár, akár a környezet lehetővé teszi a minták helyes értékének vissza-csatolását, akkor **ellenőrzött tanulási** (*supervised learning*) problémáról beszélünk. A feladat, amelyet **induktív tanulásnak** (*inductive learning*) is nevezhetünk, abban áll, hogy egy függvényt kell megtanulnunk annak bemeneti és kimeneti mintáit alapján. Diszkrét értékkészletű függvény tanulását **osztályozásnak** (*classification*), folytonos értékű függvény tanulását pedig **regressziónak** (*regression*) nevezzük.
- Az induktív tanulás magában foglalja, hogy egy, a példákkal **konzisztens** hipotézist kell találnunk. Az **Ockham borotvája** (*Ockham's razor*) elv értelmében célszerű a legegyszerűbb konzisztens hipotézis választása. Ennek a feladatnak a nehézségét a választott reprezentáció döntően befolyásolja.
- A **döntési fák** (*decision trees*) alkalmasak tetszőleges Boole-függvény reprezentálására. Az **információyerésre** (*information gain*) alapozó heurisztika hatékony módszert biztosít egyszerű, konzisztens döntési fák konstruálására.
- A tanulási algoritmus teljesítményét a **tanulási görbén** (*learning curve*) mérhetjük le, amely a **teszthalmazon** (*test set*) mért predikciós pontosságot mutatja a **tanító halma**z (*training set*) méretének függvényében.
- Az együttes tanulási módszerek, például a **turbózás** (*boosting*), gyakran jobb eredményt érnek el, mint az egyedi módszerek.
- A **tanulás számítási elmélete** (*computational learning theory*) az induktív tanulás minta komplexitásának és számítási komplexitásának analízisére koncentrál. Kompromisszumot kell kötnünk a hipotézis nyelv kifejezőképessége és a tanulás nehézsége között.

## Irodalmi és történeti megjegyzések

Az 1. fejezet felvázolta az induktív tanulás filozófiai vizsgálatának történetét. William of Ockham (1280–1349) volt korának legnagyobb hatású filozófusa és a középkori ismeretelmélet, a logika és a metafizika egyik nagy alakja. Neki tulajdonítják azt a mondást, ami „Ockham borotvája” néven rögzült a köztudatban. Latinul: *Entia non sunt multiplicanda praeter necessitatem*, ami azt jelenti magyarul, hogy „A dolgokat nem

kell a szükségesnél jobban bonyolítani". Sajnos ez a dicséretes mondás sehol sem található meg a műveiben pontosan így megfogalmazva.

Az EPAM (Elementary Perceiver And Memorizer, Elemi észlelő és megjegyző) volt az első rendszerek egyike, amely döntési fákat (**diszkriminációs hálókat, discrimination nets**) használt (Feigenbaum, 1961). Az EPAM létrehozásának célja az emberi fogalomtanulás kognitív-szimulációs modelljének megalkotása volt. A CLS-rendszer (Hunt és társai, 1979) heurisztikus előretekintő eljárást használt a döntési fák konstrukciójára. Az ID3 (Quinlan, 1979) hozta azt a kulesfontosságú ötletet, hogy a heurisztikus függvény használja az információtartalmat. Az információelméletet Claude Shannon fejlesztette ki a kommunikáció tanulmányozására (Shannon és Weaver, 1949). (Shannon hozzájárult a gépi tanulás területéhez is, az egyik legkorábbi példát szolgáltatva mechanikus egerével, amely egy útvesztőben navigált a kísérlet-kudarc elvet használva.) A döntési fák  $\chi^2$  metszését Quinlan írta le először (Quinlan, 1986). Az ipari felhasználásra alkalmas C4.5 döntési fa programcsomag is Quinlan munkáiban található meg (Quinlan, 1993). A döntési fa tanulásnak egy, az előzőtől független hagyománya alakult ki a statisztikus körében. A *Classification and Regression Trees* (Breiman és társai, 1984), amelyet röviden „CART könyv”-nek neveznek, az alapvető irodalom e tekintetben.

A tanulás számos másik algoritmikus megközelítését is vizsgálták. A **pillanatnyilag-legjobb-hipotézis (current-best-hypothesis)** megközelítés egyetlen hipotézissel foglalkozik, specializálva, ha túl tágak bizonyul, általánosítva, ha túl szűk. Ez a filozófiában már régóta ismert elv (Mill, 1843). A kognitív pszichológia korai munkájában szintén ezt javasolták, mint az emberi fogalom tanulás természetes formáját (Bruner és társai, 1957). Az ML-területen ez a megközelítés elsősorban Patrick Winston munkájához köthető, aki PhD-disszertációjában (Winston, 1970) a komplex objektumok leírásának tanulásával foglalkozott. A **verziótér (version space)** módszer (Mitchell, 1977; 1982) eltérő megközelítésen alapul, az összes konzisztens hipotézissel egyszerre foglalkozik, elhagyva azokat, amelyek az új példákkal inkonzisztenciát mutatnak. Ezt a megközelítést a **META-DENDRAL** kémiai szakértő rendszerben használták (Buchanan és Mitchell, 1978), majd később Mitchell **LEX** rendszerében (Mitchell, 1983), ami matematikai analízis problémák tanulással történő megoldását célozta. A harmadik nagyhatású irányzat kialakítása Michalski és társai munkájához köthető, akik az AQ algoritmusok sorozatával foglalkoztak, amelyek logikai szabályok halmozat tanulták (Michalski, 1969; Michalski és társai, 1986b).

Az együttes tanulás a tanuló algoritmusok teljesítménynövelésének egyre népszerűbb módszere. A zsákolás (**bagging**)<sup>7</sup> (Breiman, 1966) az első hatékony módszer, amely többféle indító (**bootstrap**) adathalmaz alapján megtanult hipotézisek kombinációját hozza létre. Ezeket az adathalmazokat az eredeti alulmintavételezésével állítják elő. A fejezetben ismertetett **turbózás (boosting)** módszere eredetileg Schapire elméleti munkájából (Schapire, 1990) származik. Az AdaBoost algoritmust Freund és Schapire dolgozta ki (Freund és Schapire, 1996), és Schapire adta meg elméleti analízisét (Schapire, 1999). Friedman és társai statisztikai alapon tárgyalják a turbózás módszerét (Friedman és társai, 2000).

A tanulási algoritmusok elméleti vizsgálata Gold munkájával kezdődött (Gold, 1967) az **identifikáció határátmenetben (identification in the limit)** probléma tanulmányo-

<sup>7</sup> A bagging név eredetileg a „bootstrap aggregating” rövidítéseként jelent meg, de az angol „bag” szó jelentését nagyrészt átvette, ezért fordítottuk „zsákolásnak”. (A ford.)

zásával. Ez a megközelítés ugyan részben tudományfilozófiai felfedezésből eredt (Popper, 1962), de alkalmazásának legfőbb területe nyelvtanok példamondatok alapján történő tanulása volt (Osherson és társai, 1986).

Míg az identifikáció határtalanenben megközelítés a végső konvergenciára koncentrál, addig a **Kolmogorov-komplexitás** (**Kolmogorov complexity**) vagy más néven **algoritmikus komplexitás** (**algorithmic complexity**), amelyet egymástól függetlenül Solomonoff (1964) és Kolmogorov (1965) fejlesztett ki, formális definíciót próbál adni az Ockham borotvája elvben használt egyszerűségnak. Hogy valahogy kimeneküljenek abból a csapdából, hogy az egyszerűség függ az információ reprezentációjának módjától, azt javasolták, hogy az egyszerűséget annak a legrövidebb programnak a hosszával mérjék, amely egy általános Turing-gépen helyesen adjva vissza a megfigyelt adatokat. Bár sok különböző általános Turing-gépet lehet létrehozni, és így több különböző „legrövidebb” program lehetséges, ezek hossza legfeljebb csak egy konstansban tér el, amely független az adatok mennyiségtől. Ezt a gyönyörű eredményt, amely betekintést nyújt abba, hogy *bármely* előzetes reprezentációs eltérést végül maguk az adatok győznek le, csupán az teszi tönkre, hogy a legrövidebb program hosszának kiszámítása eldönthetetlen problémára vezet. Közelítő mértékeket lehet alkalmazni, például a **legrövidebb leíró hosszt** (**minimum description length**) vagy LLH-t (Rissanen, 1984), amivel ki-magasló gyakorlati eredményeket értek el. Li és Vitányi írása a Kolmogorov komplexitás legjobb összefoglalása (Li és Vitányi, 1993).

A tanulás számítási elméletét – azaz a VKH-tanulás elméletét – Lesli Valiant vezette be (Valiant, 1984). Valiant munkája a számítási komplexitásra és a mintakomplexitásra helyezte a fő hangsúlyt. Michael Kearns-szel együtt Valiant megmutatta, hogy számos olyan fogalomosztály van, amely VKH-tanulással nem kezelhető probléma, még akkor sem, ha a példák elegendő információt nyújtanak. Bizonyos problémaosztályokban, például döntési listáknál, elértek néhány pozitív eredményt (Rivest, 1987).

A mintakomplexitás vizsgálatának ettől független tradícióját találhatjuk a statisztikusok körében, amely az **egyenletes konvergencia elméletének** (**uniform convergence theory**) vizsgálatával kezdődött (Vapnik és Cservonenkis, 1971). Az úgynevezett **VC-dimenzió** nagyjából a VKH tanulásból származó mértékhez hasonló – bár annál általánosabb – mértéket ad. A VC-dimenzió, ellentében a VKH-analízissel, folytonos függvényosztályokra is alkalmazható. A VKH-analízis és a VC elmélet között először a „négy német”, Blumer, Ehrenfeucht, Haussler és Warmuth (1989) teremtett kapcsolatot (akik közül valójában egyik sem német). A VC elmélet további fejlődése vezetett a **szupport vektorgép** vagy **SVM**<sup>8</sup> (**support vector machine**) kidolgozásához (Boser és társai, 1992; Vapnik, 1998). A szupport vektorgéppel a 20. fejezetben foglalkozunk.

Nagyon sok, a gépi tanulással foglalkozó fontos publikációt gyűjtöttek egybe a *Readings in Machine Learning* c. kötetben (Shavlik és Dietterich, 1990). A kétkötetes *Machine Learning 1* (Michalski és társai, 1983), illetve *Machine Learning 2* (Michalski és társai, 1986a) is sok fontos közlést tartalmaz, továbbá hatalmas bibliográfiát. Weiss és Kulikowski a függvény tanulás áttekinthető bevezetését adja gépi tanulási, statisztikai és neurális alapokon (Weiss és Kulikowski, 1991). A STATLOG projektben (Michie és társai, 1994) végezték a tanuló algoritmusok teljesítményének messze legkimerítőbb összehasonlítását. A jelenleg is folyó, a gépi tanulással foglalkozó fontos kutatási eredmény-

<sup>8</sup> Magyarul is az SVM rövidítés terjedi el. (A ford.)

mények az International Conference on Machine Learning éves kiadványaiban, a Neural Information Processing Systems konferencián, a *Machine Learning* és a *Journal of Machine Learning Research* folyóiratokban, továbbá a fontosabb MI-újságokban jelennek meg. A tanulás számítási elméletének eredményei megjelennek az éves ACM Workshop on Computational Learning Theory (COLT) kiadványokban, valamint Kearns és Vazirani (1994), továbbá Anthony és Bartlett (1999) publikációiban.

## Feladatok

- 18.1.** Vizsgálja meg a beszélni, illetve egy nyelvet megérteni tanuló gyerek problémáját! Magyarázza meg, hogy ez a folyamat hogyan illeszkedik az általános tanulási modellhez, és megfelelően azonosítsa a modell egyes komponenseit!
- 18.2.** Ismételje meg a 18.1. feladatot a tenisz (vagy valamilyen ön által ismert sport) tanulásának esetére! Ez felügyelt vagy megerősítéses tanulás?
- 18.3.** Rajzoljon fel egy döntési fát arra a problémára, hogy az útkereszteződésben elinduljunk-e előre, ha a lámpa éppen most váltott zöldre!
- 18.4.** Soha nem teszteljük kétszer ugyanazt az attribútumot a döntési fa egy útvonalá mentén. Miért?
- 18.5.** Tegyük fel, hogy egy döntési fa segítségével generálunk egy tanító mintahalmazt, majd döntési fa tanulást alkalmazunk erre a halmazra. A tanulási algoritmus végül is a helyes döntési fát fogja visszaadni, ha a tanító halmaz mérete a végtelenhez tart? Miért, vagy miért nem?
- 18.6.** Egy jó „butácska” tanulási algoritmus a következő: készítsünk egy táblázatot az összes tanítópéldából. Vizsgáljuk meg, hogy milyen kimenet szerepel a legtöbbször a tanítópéldákban: jelöljük ezt  $d$ -vel. Ezek után, ha olyan minta jelenik meg a bemeneten, ami nincs a táblázatban, akkor adjunk vissza  $d$ -t. Olyan bemenetekre, amelyek megtalálhatók a táblában, adjuk vissza a táblázatban a bemenethez asszociált kimeneti értéket (vagy ha egynél több kimeneti érték van azonos bemenethez, akkor a leggyakrabban előfordulót). Implementálja ezt az algoritmust, és vizsgálja meg egy példának választott területen (például az étterem problémán) azt, hogy mennyire működik jól! Ennek alapján fogalmat alkothatunk a tanulás területének alapszintjéről – a minimális elvárható teljesítményről, amelyet minden tanuló algoritmusnak el kell érnie.
- 18.7.** Tegyük fel, hogy egy új algoritmussal tanulási kísérletet folytat. Egy adathalmazt használ, amiben 25-25 példa van az előforduló két osztályra. Azt tervezzi, hogy a hagyj-ki-egyet keresztszűrő módszert fogja használni. Összehasonlítási alapként egy egyszerű többségi szavazót használ. (A többségi szavazónak megadva egy tanító halmazt, a bemenetektől függetlenül minden a tanító halmazban lévő példát fogja elosztani a többségi szavazónak.) Azt várta, hogy

a többségi szavazó, a hagyj-ki-egyet keresztsvalidációban nagyjából 50%-ot fog elérni, de meglepetésre 0% az eredmény. Meg tudja magyarázni, hogy miért?

- 18.8.** A döntési fák rekurzív előállítása során néha előfordul, hogy pozitív és negatív példák vegyes halmaza marad egy levélcsomópontban, még akkor is, ha az összes attribútumot használtuk már. Tegyük fel, hogy  $p$  pozitív és  $n$  negatív példánk van.
- Mutassa meg, hogy a DÖNTÉSI-FA-TANULÁS algoritmusban használt megoldás, amely a többségi osztályozást választja, minimalizálja a példahalmazra vonatkozó abszolút hibát a levélnél!
  - Mutassa meg, hogy a  $p/(p + n)$  osztály-valószínűség (class probability) eredményként való visszaadása minimalizálja négyzetes hiba összegét!
- 18.9.** Tegyük fel, hogy egy tanulási algoritmus konzisztens hipotézist keres, és az osztályozandó példákat véletlen módon generáljuk. A példákat egyenletes eloszlás-sal választjuk az  $n$  Boole-attribútum alapján kialakított  $2n$  példa lehetőségből. Számítsa ki, hány példa kell ahoz, hogy egy ellentmondás felmerülésének a valószínűsége 0,5 legyen!
- 18.10.** Tegyük fel, hogy egy attribútum a példák  $E$  halmazát  $E_i$  részhalmazokra bontja, és minden egyik részhalmazban  $p_i$  pozitív és  $n_i$  negatív példa van. Mutassa meg, hogy ha a  $p_i/(p_i + n_i)$  arány nem egyformá minden  $i$ -re, akkor az attribútum információnyeresége biztosan pozitív!
- 18.11.** Módosítsa a DÖNTÉSI-FA-TANULÁS algoritmust úgy, hogy tartalmazza a  $\chi^2$  metrót! A részleteket megtalálhatja Quinlan írásában (Quinlan, 1986).
- 18.12.** A fejezetben leírt standard DÖNTÉSI-FA-TANULÁS algoritmus nem képes azon esetek kezelésére, amelyekben néhány példa esetén hiányoznak attribútumértékek.
- Először is valamilyen módszerre van szükségünk, amellyel egy adott döntési fa alapján be tudunk sorolni példákat akkor is, ha a fában olyan attribútumokra vonatkozó tesztek szerepelnek, amelyek értéke hiányozhat egyes példáknál. Tegyük fel, hogy az  $X$  példa  $A$  attribútumra vonatkozó értéke hiányzik, és a döntési fa teszteli  $A$ -t egy olyan csomópontban, amit  $X$  elér. Ennek az esetnek egyik kezelési lehetősége, hogy úgy teszünk, mintha a példa az összes lehetséges attribútumértékkel rendelkezne, de súlyozzuk az egyes értékeket azzal a gyakorisággal, amelyet az összes olyan példán mérünk, amely eléri a döntési fa ezen csomópontját. A besorolási algoritmusnak minden olyan csomópont összes ágát követnie kell, amelyben egy érték hiányzik, és minden egyik út mentén össze kell szoroznia a súlyokat. Írja meg azt a módosított döntési fa algoritmust, amely rendelkezik ezzel a tulajdonsággal!
  - Módosítsa az információnyereség számítást a következő módon: a fa építésének fázisában egy adott csomópontba jutó tetszőleges  $C$  tanító példa-halmaznál azok a példák, amelyeknek hiányzik bármely hátralevő (még nem vizsgált) attribútumhoz tartozó értéke, a  $C$  halmazban ezen attribútumra előforduló értékgyakoriság szerint kapjanak „mint-ha” értéket!

- 18.13.** A 18. fejezetben megmutattuk, hogy azok az attribútumok, amelyeknek nagyon sokféle lehetséges értékük van, problémát jelenthetnek az információnyereség számításánál. Ezek az attribútumok jó eséllyel nagyon sok kis osztályra, akár egyelemű osztályokra bontják a halmazt, ezért úgy tűnik a nyereség számításánál, hogy nagyon fontosak az adott csomópontban. A **nyereségarány** (*gain ratio*) kritérium az attribútumokat az általuk hozott információnyereség és a saját belső információtartalmuk arányával méri. A belső információtartalom arra a kérdésre adott válasz, hogy „Mi az értéke ennek az attribútumnak?”. A nyereségarány kritérium ennek megfelelően azt próbálja mérimi, hogy mennyire hatékony információt biztosít a példa helyes osztályozásához ez az attribútum. Alkossa meg az attribútum információtartalmának matematikai kifejezését, és módosítsa a DÖNTÉSI-FA-TANULÁS algoritmust nyereségarány-alapúra!
- 18.14.** Egy együttes tanulási algoritmust vizsgál, amely  $M$  megtanult hipotézis eredményének egyszerű többségi szavazását használja. Tegyük fel, hogy az összes hipotézisnek  $\varepsilon$  a hibája, és az egyes hipotézisek által elkövetett hibák függetlenek a többiek hibájától. Adja meg  $M$  és  $\varepsilon$  függvényében azt az összefüggést, amelyivel az együttes tanulással előállított eszköz hibája számítható, és számítsa ki  $M = 5, 10$  és  $20$ , illetve  $\varepsilon = 0,1, 0,2$  és  $0,4$  esetén! Ha feladjuk a függetlenségre vonatkozó feltételezést, akkor előfordulhat-e, hogy az együttes hibája rosszabb lesz, mint  $\varepsilon$ ?
- 18.15.** Ez a feladat a döntési listák kifejező erejével foglalkozik (lásd 18.5. alfejezet).
- Mutassa meg, hogy egy döntési lista képes bármely Boole-függvény reprezentálására, ha nem korlátozzuk a tesztek méretét!
  - Mutassa meg, hogy ha egy teszt legfeljebb  $k$  literálist tartalmazhat, akkor a döntési lista minden olyan függvény reprezentálására képes, amely egy  $k$  mélységű döntési fával reprezentálható!

# 19. A TUDÁS SZEREPE A TANULÁSBAN

*Ebben a fejezetben a tanulással foglalkozunk, de csak akkor, amikor már tudunk is valamit.*

Az előbbi három fejezetben bemutatott tanulási módszerek mindegyikében az volt az alapötlet, hogy egy olyan függvényt konstruálunk, amelynek bemeneti/kimeneti viselkedése megegyezik azzal, amit az adatoknál megfigyeltünk. Bármilyen esetet is nézünk, a tanulási módszer tulajdonképpen nem más, mint egy megfelelő függvénynek a hipotézistérben történő megkeresése, a függvény formájára vonatkozó igen elemi feltételezésekkel kiindulva. Ilyen feltételezés lehet például, hogy a függvény „egy másodrendű polinom” vagy „egy döntési fa”, vagy az az elfogultság, hogy „az egyszerűbb a jobb”. Ennek megvalósítása felért azzal, mintha azt mondánánk, hogy mielőtt valami újat tanulnánk, el kellene felejteni (majdnem) minden, amit tudunk. Ebben a fejezetben olyan tanulási módszereket tanulmányozunk, amelyek a priori tudásból (prior knowledge) merítenek előnyöket. Az esetek többségében az a priori tudást elsőrendű elméletek segítségével reprezentáljuk. Így most először fogjuk összekapcsolni a tudás-reprezentációra és a tanulásra vonatkozó eredményeket.

## 19.1. A TANULÁS LOGIKAI MEGFOGALMAZÁSA

A 18. fejezetben a tisztán induktív tanulást olyan folyamatnak tekintettük, amely egy, az adatokkal konziszens hipotézist keresett. Most ezt a definíciót olyan esetre pontosítjuk, amikor a hipotézist logikai állítások halmazával reprezentáljuk. A példaleírások és besorolások szintén logikai állítások, egy új példát pedig úgy tudunk osztályozni, hogy egy osztályozó állítást a hipotézisből és a példa leírásából kikövetkeztetünk. Ezzel a megközelítéssel a hipotéziseket inkrementálisan, állításokonként építhetjük. A megközelítés lehetővé teszi az a priori tudás használatát is, mivel a már megismert állításokat bevethetjük új példák osztályozásába. A tanulás logikai megfogalmazásában először látszólag sok a munka, ez azonban a tanulás számos aspektusát segíti tisztázni. Ez lehetővé teszi, hogy a 18. fejezetben megismert egyszerű tanulási módszereket jócskán túlsáryaljuk azáltal, hogy a logikai következtetés teljes erejét tanulási célokra használjuk.

### Példák és hipotézisek

Emlékezzünk a 18. fejezetben a tanulási problémára bemutatott étterem példára: egy olyan döntési szabály megtanulására, hogy vajon érdemes-e egy asztalra vájni. A pél-

dákat olyan attribútumokkal (**attributes**) írtuk le, mint az *Alternatíva*, *Bár*, *Péntek/Szombat* stb. Logikai megközelítésben egy példa egy logikai állítással leírt objektum; az attribútumok pedig unáris predikátumok. Nevezzük általánosságban az *i*-edik példát  $X_i$ -nek. A 18.3. ábra első példáját az alábbi állítások írják le:

$$\text{Alternatíva}(X_1) \wedge \neg \text{Bár}(X_1) \wedge \neg \text{Péntek/Szombat}(X_1) \wedge \text{Éhes}(X_1) \wedge \dots$$

A  $D_i(X_i)$  jelöléssel az  $X_i$  leírására fogunk hivatkozni, ahol  $D_i$  tetszőleges, egyargumen-tumú logikai kifejezés lehet. Az objektumok osztályozását a

$$\text{Várjunk}E(X_1)$$

állítás végzi. Pozitív példák esetén a  $Q(X_i)$ , negatív példák esetén a  $\neg Q(X_i)$  általános je-lölést fogjuk használni. A teljes tanító halmaz az összes leíró és besoroló állítás konjunkciója.

Logikai megközelítésben az induktív tanulás célja a  $Q$  célpredikátummal ekvivalens logikai kifejezés megtalálása, amivel korrektül tudjuk a példákat osztályozni. Mind-egyik hipotézis egy-egy javaslat a keresett logikai kifejezésre, amit mi a célpredikátum **definíciójelöltjének** (*candidate definition*) nevezünk. Jelöljük  $C_r$ -vel a definíciójelöltet, ekkor mindegyik  $H_i$  hipotézis egy  $\forall x Q(x) \Leftrightarrow C_i(x)$  formájú kifejezés. Például a dönté-si fa azt fejezi ki, hogy a célpredikátum egy objektum esetében csak akkor igaz, ha az *igaz*-hoz vezető egyik ága teljesül. Így a 18.6. ábrán látható döntési fa a következő logikai definíciót fejezi ki (amelyre a jövőben  $H_r$ -rel fogunk hivatkozni):

$$\begin{aligned} \forall r \text{ Várjunk}E(r) &\Leftrightarrow \text{Vendégek}(r, \text{Néhány}) \\ &\quad \vee \text{Vendégek}(r, \text{Tele}) \wedge \text{Éhes}(r) \wedge \text{Konyha}(r, \text{Francia}) \\ &\quad \vee \text{Vendégek}(r, \text{Tele}) \wedge \text{Éhes}(r) \wedge \text{Konyha}(r, \text{Thai}) \\ &\quad \quad \wedge \text{Péntek/Szombat}(r) \\ &\quad \vee \text{Vendégek}(r, \text{Tele}) \wedge \text{Éhes}(r) \wedge \text{Konyha}(r, \text{Burger}) \end{aligned} \quad (19.1)$$

Mindegyik hipotézis azt jósolja meg, hogy példák egy bizonyos halmaza, nevezetesen azok, amelyek kielégítik a hipotézisdefiníció jelöltjét, a célpredikátumnak megfelelő példák lesznek. Ezt a halmazt a predikátum **kiterjedésének** (*extension*) nevezünk. Két különböző kiterjedéssel rendelkező hipotézis így logikailag ellentmondásban van egy-mással, hiszen legalább egy példa esetén eltérő eredményt jósolnak. Ha viszont azonos a kiterjedésük, akkor logikailag ekvivalensek.

A  $H$  hipotézistér ezek után az összes olyan hipotézis  $\{H_1, \dots, H_n\}$  halmaza, amelyek kezelésére a tanuló algoritmust terveztek. Például DÖNTÉSI-FA-TANULÁS algoritmus az adott attribútumokkal kialakítható összes döntési fa hipotézissel képes foglalkozni; így hipotézistere az összes döntési fát tartalmazza. Feltehetően a tanuló algoritmus hisz abban, hogy az egyik hipotézis korrekt, azaz hisz a

$$H_1 \vee H_2 \vee H_3 \vee \dots \vee H_n \quad (19.2)$$

logikai kifejezésben. Ahogy a példák érkeznek, a példákkal nem **konzisztenzs** (*consistent*) hipotéziseket ki lehet zární. Vizsgáljuk meg alaposabban a konziszencia ezen fogalmát. Nyilvánvaló, hogy ha a  $H_i$  hipotézis a teljes tanító halmazzal konzisztenzs, akkor mindegyik példával konzisztenzsnek kell lennie. Mit jelent viszont az, hogy vala-melyik példával ellentmondásban van? Ez kétféle módon történhet meg:

- A hipotézis szempontjából egy példa **hamis negatív (false negative)**, ha a valóságban pozitív, de a hipotézis szerint negatív. Például a következő leírással rendelkező  $X_{13}$  új példa:

$$\begin{aligned} & \text{Vendégek}(X_{13}, \text{Tele}) \wedge \text{BecsültVárakozásE}(X_{13}, 0\text{-}10) \wedge \neg \text{Éhes}(X_{13}) \wedge \dots \wedge \\ & \wedge \text{VárjunkE}(X_{13}) \end{aligned}$$

a korábban definiált  $H_r$  hipotézis számára hamis negatív példa lesz.  $H_r$ -ből, illetve a példa leírásából kikövetkeztethető minden a  $\text{VárjunkE}(X_{13})$  – ezt állítja a példa, minden a  $\neg \text{VárjunkE}(X_{13})$  – ezt állítja a hipotézis. Tehát a hipotézis és a példa logikailag ellentmondásban van.

- A hipotézis szempontjából egy példa **hamis pozitív (false positive)**, ha a valóságban negatív, de a hipotézis szerint pozitív.<sup>1</sup>

Ha egy példa hamis negatív vagy hamis pozitív egy hipotézis szempontjából, akkor a példa és a hipotézis logikailag ellentmondásban van egymással. Feltéve, hogy a példa a tények korrekt megfigyelésén alapul, a hipotézist kizárhatsuk. Logikailag ez a következetetés a rezolúciós szabállyal pontos analógiában van (lásd 9. fejezet). Hipotézisek diszjunkciója felel meg egy klóznak, a példa pedig megfelel annak a literálisnak, amely a klózban levő egyik literálissal rezolvál. Egy szokásos logikai következetető rendszer tehát elméletileg tud példákból tanulni oly módon, hogy egy vagy több hipotézist kizár. Tegyük fel például, hogy a példát az  $I_1$  állítással jelöljük, és a hipotézistér  $H_1 \vee H_2 \vee H_3 \vee H_4$ . Ha  $I_1$  ellentmondásban van  $H_2$ -vel és  $H_3$ -mal, akkor a logikai következetető rendszer származtatni tudja az új  $H_1 \vee H_4$  hipotézisteret.

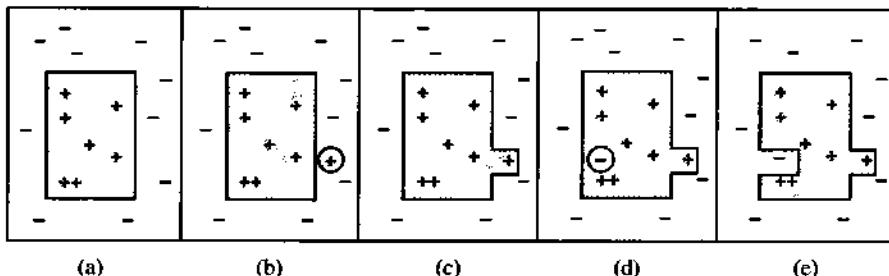
Tehát az induktív tanulást logikai leírással olyan folyamatként jellemezhetjük, mint amelyik fokozatosan kizárja a példáknak ellentmondó hipotéziseket, leszűkítve a lehetőségek körét. Mivel a hipotézistér rendszerint nagy (vagy elsőrendű logika esetén egyenesen végtelen), nem javasoljuk egy rezolúcióalapú tételbizonyító rendszer építését és a hipotézistér teljes számbavételét. Ehelyett két megközelítést tárgyalunk, amelyek sokkal kisebb erőfeszítéssel logikailag ellentmondásmentes hipotéziseket képesek találni.

## A pillanatnyilag legjobb hipotézis keresése

**A pillanatnyilag legjobb hipotézis (current-best-hypothesis)** keresési eljárás alapgondolata az, hogy egyetlen hipotézist vegyük figyelembe, és ha új példa érkezik, akkor ennek figyelembevételével alakítsuk át a hipotézist annak érdekében, hogy az ellentmondás-mentességet fenntartsuk. Az algoritmus alapját John Stuart Mill írta le (Mill, 1843), de könnyen lehet, hogy már korábban megalkották.

Tegyük fel, hogy van valamilyen hipotézünk, például  $H_r$ , amit már meglehetősen megkedvünk. Amíg egy új példával sincs ellentmondásban, semmit sem kell tennünk. Aztán egyszer csak egy hamis negatív példa ( $X_{13}$ ) érkezik. Mit csinálunk? A 19.1. (a) ábra sematikusan úgy mutatja be  $H_r$ -t, mint egy területet, a négy szögön belül minden  $H_r$  kiterjedésébe tartozik. Az eddig látott példákat „+” vagy „–” jelöl, és látható, hogy  $H_r$  jól sorolja be az összes példát, mint  $\text{VárjunkE}$  pozitív vagy negatív példáit. A 19.1. (b)

<sup>1</sup> „Hamis pozitív” és „hamis negatív” kifejezések először a gyógyításban használtak arra, hogy a téves laboratóriumi eredményeket jellemzzék. Egy eredmény akkor hamis pozitív, ha azt jelzi, hogy a páciens az adott betegségen szenved, míg valójában nem ez a helyzet.



**19.1. ábra.** (a) Egy konzisztens hipotézis. (b) Egy hamis negatív példa. (c) A hipotézist általánosítottuk. (d) Egy hamis pozitív példa. (e) A hipotézist szűkítettük.

ábrán egy új – hamis negatív – példa jelenik meg (bekarikázva): a hipotézis azt állítja, hogy negatív, pedig valójában pozitív példa. A hipotézis kiterjedését növelni kell ahhoz, hogy tartalmazza ezt a példát is. Ezt a lépést általánosításnak (*generalization*) nevezzük: egy lehetséges általánosítás a 19.1. (c) ábrán látható. Ezután a 19.1. (d) ábrán egy hamis pozitív példát látunk: a hipotézis azt állítja, hogy pozitív, pedig valójában negatív. A hipotézis kiterjedését csökkenteni kell ahhoz, hogy kizártuk ezt a példát. Ezt szűkítésnek (*specialization*) nevezzük; a 19.1. (e) ábrán a hipotézis egy lehetséges szűkítését látjuk.

Most már specifikálni tudjuk a 19.2. ábrán látható PILLANATNYILAG-LEGJOBB-TANULÁS algoritmusát. Vegyük észre, hogy valahányszor általánosítjuk vagy szűkítjük a hipotézist, ellenőriznünk kell az összes többi példára, hiszen nincs garancia arra, hogy a kiterjedés önkényes növelése/csökkentése elkerüli bármelyik másik negatív/pozitív példa befoglalását/kirekesztését.

Az általánosítást és a szűkítést úgy definiáltuk, mint olyan műveleteket, amelyek a hipotézis *kiterjedését* változtatják meg. Most pontosan meg kell határoznunk azt, hogy ezek a műveletek hogyan implementálhatók olyan szintaktikus műveletekként, amelyek a hipotézishoz rendelt definíciójelöllet úgy változtatják meg, hogy azt egy program végre tudja hajtani. Ehhez először is vegyük észre, hogy az általánosítás és a szűkítés egyben hipotézisek közötti *logikai* relációk. Ha a  $C_1$  definíciónak megfelelő  $H_1$  hipotézis a  $C_2$  definícióknak megfelelő  $H_2$  hipotézis általánosítása, akkor fenn kell állnia annak, hogy:

$$\forall x \ C_2(x) \Rightarrow C_1(x)$$

```
function PILLANATNYILAG-LEGJOBB-TANULÁS(példák) returns egy hipotézis
```

```

 $H \leftarrow$  egy olyan hipotézis, amelyik a példák halmaz első példájával konzisztens
for each a példák további példáira do
  if a H alapján e hamis pozitív then
     $H \leftarrow$  choose H olyan szűkítését, amely konzisztens a példák példáival
  else if a H alapján e hamis negatív then
     $H \leftarrow$  choose H olyan általánosítását, amely konzisztens a példák példáival
    if ha nem található konzisztens szűkítés/általánosítás then fail
  return H
```

**19.2. ábra.** A pillanatnyilag-legjobb-hipotézis tanuló algoritmus. Konzisztens hipotézist keres és viszszalép, ha nem található konzisztens szűkítés/általánosítás.

Tehát ahhoz, hogy a  $H_2$  hipotézis általánosítását létrehozzuk, egyszerűen egy olyan  $C_1$  definíciót kell találnunk, amelyik logikailag következik  $C_2$ -ból. Ezt könnyen megtehetjük. Ha például  $C_2(x) \text{ Alternatíva}(x) \wedge \text{Vendégek}(x, \text{Néhány})$ , akkor egy általánosítási lehetőség a  $C_1(x) \equiv \text{Vendégek}(x, \text{Néhány})$ . Ez a feltételek törlésének (*dropping conditions*) nevezik. Felfoghatjuk úgy, hogy egy gyengébb definíció jön létre, így a pozitív példák nagyobb halmazat teszi lehetővé. Számos egyéb általánosítási eljárás is létezik, azon nyelvtől függően, amelyen a műveleteket végezzük. Hasonlóképpen, egy hipotézist szűkíteni tudunk, ha további feltételeket adunk a hozzá tartozó definíciójelölthöz, illetve ha eltávolítunk tagokat egy diszjunktív definícióból. Lássuk, hogyan működik ez az éttermi feladatban, 18.3. ábra adatait használva:

- Az első példa –  $X_1$  – pozitív.  $\text{Alternatíva}(X_1)$  igaz, tehát vegyük fel a következő kiinduló hipotézist:

$$H_1: \forall x \text{ VárjunkE}(x) \Leftrightarrow \text{Alternatíva}(x)$$

- A második példa –  $X_2$  – negatív.  $H_1$  alapján pozitív lenne, így hamis pozitív. Ezért szűkítenünk kell  $H_1$ -et. Ezt megtehetjük például úgy, hogy egy olyan további feltételt adunk hozzá, amely feltétel kizáraja  $X_2$ -t. Egy lehetőség:

$$H_2: \forall x \text{ VárjunkE}(x) \Leftrightarrow \text{Alternatíva}(x) \wedge \text{Vendégek}(x, \text{Néhány})$$

- A harmadik példa –  $X_3$  – pozitív.  $H_2$  alapján negatív lenne, így hamis negatív. Tehát általánosítanunk kell  $H_2$ -t. Ezt elérhetjük, ha töröljük *Alternatíva* feltételt, amely a következő hipotézisre vezet:

$$H_3: \forall x \text{ VárjunkE}(x) \Leftrightarrow \text{Vendégek}(x, \text{Néhány})$$

- A negyedik példa –  $X_4$  – pozitív.  $H_3$  alapján negatív lenne, így hamis negatív. Így általánosítanunk kell  $H_3$ -at. Nem törölhetjük *Vendégek* feltételt, mert ez egy minden tartalmazó hipotézishez vezetne, ami ellentmondásban van  $X_2$ -vel. Egyik lehetőség a következő diszjunkció bevezetése:

$$H_4: \forall x \text{ VárjunkE}(x) \Leftrightarrow \text{Vendégek}(x, \text{Néhány}) \\ \vee (\text{Vendégek}(x, \text{Tele}) \wedge \text{Péntek/Szombat}(x))$$

A hipotézis ezek után már kezd ésszerűnek tűnni. Nyilvánvaló, hogy más lehetőségek is vannak, amelyek konzisztensek az első négy példával, kettő ezek közül:

$$H'_4: \forall x \text{ VárjunkE}(x) \Leftrightarrow \neg \text{BecsültVárakozás}(x, 30-60)$$

$$H''_4: \forall x \text{ VárjunkE}(x) \Leftrightarrow \text{Vendégek}(x, \text{Néhány}) \\ \vee (\text{Vendégek}(x, \text{Tele}) \wedge \text{BecsültVárakozás}(x, 10-30))$$

A PILLANATNYILAG-LEGJOBB-TANULÁS algoritmusát nem determinisztikusan írjuk le, mivel bármely ponton számos olyan szűkítési vagy általánosítási lehetőség lehet, melyek bármelyikét alkalmazhatjuk. A már meghozott döntések nem szükségszerűen vezetnek a legegyszerűbb hipotézishez, sőt olyan megoldhatatlan szituációhoz is vezethetnek, amelyben nincs a hipotézisnek olyan egyszerű módosítása, amely minden adattal konzisztens hipotézist állít elő. Ilyen esetekben a programnak egy előző választási ponthoz kell visszalépnie.

A PILLANATNYILAG-LEGJOBB-TANULÁS algoritmusát – és különböző változatait – számos gépi tanulást felhasználó rendszerben alkalmazták, először Patrick Winston (Winston, 1970) „ív-tanuló” programjában. Azonban ha a tér nagy, és nagyszámú példával dolgozunk, akkor bizonyos nehézségek merülnek fel:

1. Az összes előző példa újraellenőrzése minden egyes módosításnál nagyon munkaigényes.
2. Nehéz jó keresési heurisztikát találni, és a visszalépések örökké tarthatnak. Mint a 18. fejezetben már láttuk, a hipotézistér kétszeresen exponenciálisan nagy tér lehet.

## Legkisebb megkötés elvű keresés

A visszalépésre azért van szükség, mert a „pillanatnyilag-legjobb-hipotézis” eljárásban **választani** kell egy legjobb becslést, egy partikuláris hipotézist, akkor is, ha még nincs elég adatunk ahhoz, hogy biztosak legyünk a választásunk helyességében. Ehelyett azt tehetjük, hogy az összes olyan és csak olyan hipotézist megtartjuk, amelyek az eddigi adatainkkal konzisztenek. Ezek után mindegyik új eset vagy nincs hatással, vagy kizárt néhányat a hipotéziseink közül. Emlékezzünk arra, hogy a hipotézistér egy diszjunktív állításnak tekinthető:

$$H_1 \vee H_2 \vee H_3 \vee \dots \vee H_n$$

Ahogy egyre több hipotézis bizonyul a példákkal inkonzisztensnek, ez a diszjunktív állítás zsugorodik, csak azokat a hipotéziseket tartva meg, amelyeket nem szűrtünk ki. Feltéve, hogy az eredeti hipotézistér tartalmazza a helyes választ, a redukált diszjunktív állításnak is szükségszerűen tartalmaznia kell azt, hiszen csak inkorrekt hipotéziseket szűrtünk ki. A megmaradt hipotézisek halmazát nevezzük **verziótérnek (version space)**. Az ilyen elven működő tanulási algoritmust (amelyet a 19.3. ábrán vázoltunk fel) verziótér tanulási algoritmusnak (más néven **jelölteltávolítási – candidate elimination** – algoritmusnak) nevezzük.

Fontos tulajdonsága ennek a megközelítésnek, hogy **inkrementális**: soha nem kell visszalépnünk és újravizsgálnunk a régi példákat. Az összes megmaradt hipotézis garantáltan konzisztens az összes példával. Másrészt **legkisebb megkötés elvű (least**

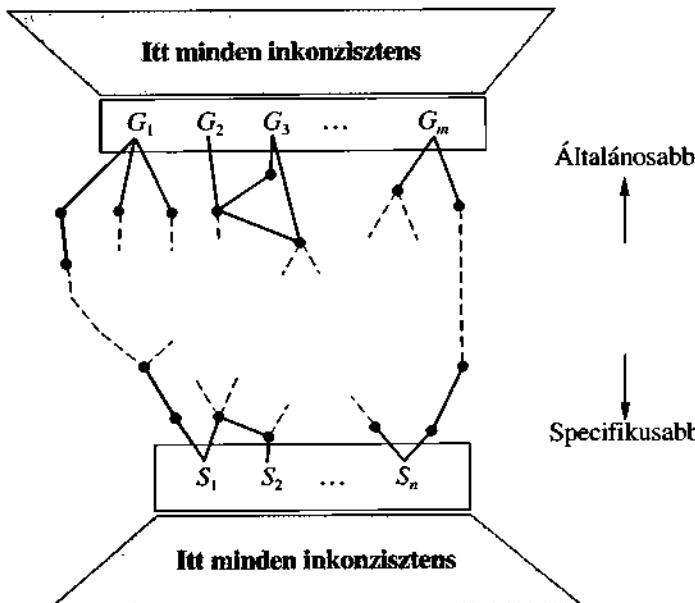
```
function VERZIÓ-TÉR-TANULÁS(példák) returns egy verziótér
  local variables: V, a verziótér: az összes hipotézis tere
```

```
    V  $\leftarrow$  az összes hipotézis halmaza
    for each a példák összes e példájára do
      if V nem üres then VERZIÓ-TÉR-MÓDOSÍTÁS(V, e)
    return V
```

---

```
function VERZIÓ-TÉR-MÓDOSÍTÁS(V, e) returns egy módosított verziótér
  V  $\leftarrow$  {h  $\in$  V : h konzisztens e-vel}
```

**19.3. ábra.** Verziótér tanuló algoritmus. *V*-nek olyan részhalmazát tanulja meg, amelynek elemei a *példákkal* konzisztensek.



19.4. ábra. A verziótér a példákkal konzisztens összes hipotézist tartalmazza

(commitment) algoritmusnak is nevezhetjük, ugyanis nem tesz önkényes választásokat (vö. a 11. fejezetben található részben rendezett tervkészítő algoritmussal). Van viszont egy nyilvánvaló probléma. Már leszögeztük, hogy a hipotézistér óriási, hogyan tudjuk akkor leírni ezt az óriási diszjunktív állást?

A következő egyszerű analógia sokat segít a megértésben. Hogyan ábrázoljuk az összes 1 és 2 közé eső valós számot? Végül is végtelen sok ilyen van! A megoldás az intervallumábrázolás, amelyik csak a halmaz határait specifikálja:  $[1, 2]$ . Ez azért működik, mert a valós számok felett értelmezett egy *rendezés*.

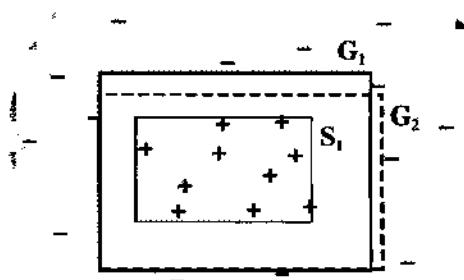
A hipotézistér felett is rendelkezünk rendezéssel, nevezetesen az általánosítás/szűkítés alapján. Ez egy részleges rendezés, ami azt jelenti, hogy a határ nem egy pont, hanem a hipotézisek egy halmaza: a **határhalmaz** (boundary set). A nagyszerű a dologban az, hogy a teljes verziótér reprezentálható csupán két határhalmaz: egy legáltalánosabb határhalmaz (a **G halmaz**) és egy legszűkebb határhalmaz (az **S halmaz**) segítségével. *E kettő közé eső bármely hipotézis garantáltan konzisztens az összes példával*. Mielőtt ezt bizonyítjuk, röviden foglaljuk össze az eddigieket:

- A jelenlegi verziótér azon hipotézisek halmaza, amelyek az összes eddigi példával konzisztensek. Ezt az **S halmaz** és a **G halmaz** segítségével ábrázoljuk, mindenkor a hipotézisek valamelyen halmaza.
- Az **S halmaz** összes eleme konzisztens az összes eddigi megfigyeléssel, és nincs ennél szűkebb konzisztens hipotézis.
- A **G halmaz** összes eleme konzisztens az összes eddigi megfigyeléssel, és nincs ennél általánosabb konzisztens hipotézis.

Azt kívánjuk meg, hogy a kezdeti tér (mielőtt bármelyik példát megvizsgáltuk volna), az összes elképzelhető hipotézist tartalmazza. Ezt úgy biztosítjuk, hogy a  $G$  halmazt egyszerűen az *Igaz* értékre állítjuk (az a hipotézis, amely minden tartalmaz), az  $S$  halmazt pedig egyszerűen a *Hamis* értékre állítjuk (az a hipotézis, amelynek kiterjedése üres).

A 19.4. ábra a verziótér határhalmazokkal való ábrázolásának általános struktúráját mutatja. Ahhoz, hogy megmutassuk, hogy ez az ábrázolás elégéges, két tulajdonságra van szükségünk:

1. Mindegyik konzisztens hipotézis (amelyik nem valamelyik határhalmaz eleme) szűkebb, mint a  $G$  halmaz valamelyik eleme, és általánosabb, mint az  $S$  halmaz valamelyik eleme. (Azaz nincsenek kívül „kóbor” elemek.) Ez közvetlenül következik az  $S$  és  $G$  definíciójából. Ha lenne egy kóbor  $h$ , akkor az vagy nem lehetne szűkebb  $G$  valamely eleménél, tehát  $G$ -hez kellene tartoznia, vagy nem lehetne általánosabb  $S$  valamely eleménél, ebben az esetben viszont  $S$ -hez tartozna.
2. Mindegyik hipotézis, amelyik szűkebb a  $G$  halmaz valamely eleménél, és általánosabb az  $S$  halmaz valamely eleménél, konzisztens hipotézis (azaz a határok között nincsenek „lyukak”). Bármely – az  $S$  és  $G$  közé eső –  $h$ -nak az összes olyan negatív példát vissza kell utasítania, amelyet  $G$  minden egyes tagja visszautasít (mivel szűkebb), ugyanakkor el kell fogadnia az összes pozitív példát, amelyet az  $S$  akármelyik tagja elfogad (mivel általánosabb). Így  $h$  az összes példát jól kezeli, tehát nem lehet inkonzisztens. A 19.5. ábra mutatja be a helyzetet: nincs ismert példa, amely  $S$ -en kívül, de  $G$ -n belül helyezkedik el, így a kettő között elhelyezkedő összes hipotézisnek konzisztensnek kell lennie.



19.5. ábra. A  $G$  és az  $S$  elemeinek kiterjesztései. A kettő között nem található ismert példa.

Bemutattuk tehát, hogy ha  $S$ -t és  $G$ -t definíciójuknak megfelelően kezeljük, akkor a verziótér kielégítő leírását adják. Az egyetlen hátralévő probléma, hogy hogyan módosítsuk  $S$ -t és  $G$ -t egy új példa esetén (a VERZIÓ-TÉR-MÓDOSÍTÁS függvény feladata). Elsőre ez meglehetősen bonyolultnak tűnik, de a definíciók és a 19.4. ábra alapján nem túl nehéz az algoritmus előállítása.

Az  $S$  és  $G$  halmazok  $S_i$  és  $G_i$  elemeivel kell foglalkoznunk. Az új példa bármelyik esetén lehet hamis pozitív vagy hamis negatív.

1.  $S_i$ -re hamis pozitív. Azt jelenti, hogy  $S_i$  túl általános, de mivel  $S_i$ -nek (a definíciója értelmében) nincs konzisztens szűkítése, így  $S_i$ -t eltávolítjuk az S halmazból.
2.  $S_i$ -re hamis negatív. Azt jelenti, hogy  $S_i$  túl szűk, így  $S_i$ -t az összes közvetlen általánosításával helyettesítjük, feltéve, hogy ezek szűkebbek, mint G egyes elemei.
3.  $G_i$ -re hamis pozitív. Azt jelenti, hogy  $G_i$  túl általános, így az összes közvetlen szűkítésével helyettesítjük, feltéve, hogy ezek általánosabbak, mint S egyes elemei.
4.  $G_i$ -re hamis negatív. Azt jelenti, hogy  $G_i$  túl szűk, de mivel nincs konzisztens általánosítása (definíciója értelmében), így  $G_i$ -t eltávolítjuk a G halmazból.

Ezeket a műveleteket minden új példára elvégezzük addig, amíg az alábbi három eset valamelyike fel nem lép:

1. Pontosan egy hipotézis marad a verziótérben, ebben az esetben ezt adjuk vissza mint az egyetlen hipotézist.
2. A verziótér **összeomlik** – vagy S, vagy G üressé válik, azt jelezve, hogy nincs a tanító halmazra nézve konzisztens hipotézis. Ez megegyezik az egyszerű döntési fa algoritmus sikertelen kimenetelével.
3. Kifogyunk a példákból, miközben számos hipotézis maradt a verziótérben. Ez azt jelenti, hogy a verziótér a hipotézisek diszjunkcióját reprezentálja. Ha egy új példa esetén a diszjunkció összes tagjában azonos eredményt kapunk, akkor ezt adhatjuk vissza mint a példa besorolását. Ha az eredmények nem egyeznek meg, akkor egyik lehetőséggé alkalmazhatjuk a többségi szavazást.

Gyakorlás céljára hagyjuk a VERZIÓ-TÉR-TANULÁS algoritmusnak az éttermi adatokra való alkalmazását.

A verziótér megközelítésnek két alapvető hátránya van:

- Ha a problématerület adatai zajt tartalmaznak, vagy ha az attribútumok a pontos besoroláshoz nem elégsgesek, akkor a verziótér minden összeomlik.
- Ha korlátlan méretű diszjunkciót megengedünk a hipotézistérben, akkor az S halmaz minden tartalmazni fog egy egyedi legszűkebb hipotézist, nevezetesen az összes eddig látott pozitív példa diszjunkcióját. Hasonlóképpen a G halmaz tartalmazni fogja a negatív példák diszjunkciójának negáltját.
- Egyes hipotézistereknél az S- és a G-beli elemek száma az attribútumok számában exponenciálisan növekedhet annak ellenére, hogy az ilyen hipotézisterekhez ismertek hatékony tanuló algoritmusok.

A zaj problémájára napjainkig nem ismert teljes, sikeres megoldás. A diszjunktív kapcsolatok problémája kezelhető oly módon, hogy csak korlátozott méretű diszjunktív formákat engedünk meg, vagy az általánosabb predikátumokra egy **általánosítási hierarchiát (generalization hierarchy)** vezetünk be. Például a *BecsültVárakozás(x, 30-60)  $\vee$  BecsültVárakozás(x, >60)* helyett használhatjuk a *HosszúVárakozás(x)* literált. Az általánosítási és a szűkítési műveletek egyszerűen kiterjeszhetők ennek kezelésére.

A verziótér algoritmus tiszta változatát először a Meta-DENDRAL rendszerben alkalmazták. Ezt a rendszert olyan szabályok megtanulására terveztek, amelyek azt írják le, hogy egy tömegspektrométerben a molekulák hogyan esnek szét darabokra (Buchanan és Mitchell, 1978). A Meta-DENDRAL képes volt olyan szabályok generálására, amelyek elég újak voltak egy analitikus kémiával foglalkozó újságban való publikálhatósághoz.

Ezek voltak az első – egy számítógépes program által felfedezett – valós tudományos eredmények. Az algoritmust az elegáns LEX rendszerben (Mitchell és társai, 1983) szintén használták, ez a rendszer saját sikereinek és kudarcainak tanulmányozása alapján szimbolikus integrálási problémák megoldásának megtanulására volt képes. Bár a verzitér-módszerek valószínűleg a legtöbb valós probléma tanulása esetén – főleg a zaj miatt – nem praktikusak, jó betekintést adnak a hipotézistér logikai szerkezetébe.

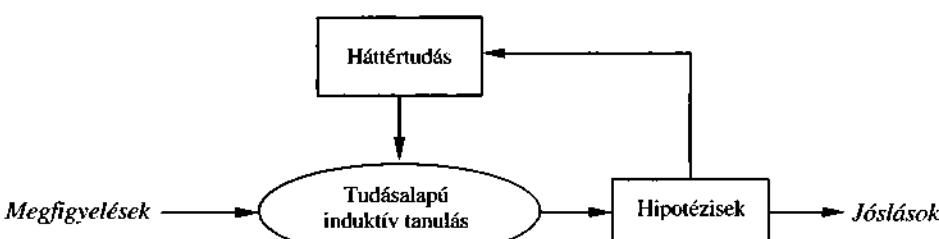
## 19.2. A TUDÁS SZEREPE A TANULÁSBAN

Az előbbi részben az induktív tanulás egyszerű sémájával foglalkoztunk. Ahhoz, hogy a háttértudás szerepét megérthessük, beszélünk kell a hipotézisek, a példaleírások és a besorolások közötti kapcsolatról. Jelölje a *Leírások* az összes példaleírás konjunkcióját és a *Besorolások* a példabesorolások konjunkcióját. A „megfigyelést megmagyarázó” *Hipotézis*-nek akkor a következő tulajdonsággal kell rendelkeznie (emlékezzünk, hogy  $a \vdash$  jelentése a „logikailag maga után vonz”):

$$\text{Hipotézis} \wedge \text{Leírások} \vdash \text{Besorolások} \quad (19.3)$$

Az ilyen kapcsolatot, ahol a *Hipotézis* az „ismeretlen”, **vonzatkényszernek (entailment constraint)** nevezzük. Ennek a kényszernek a megoldása a tiszta induktív tanulás, amikor a *Hipotézis*-t valamelyen előre definiált hipotézistérből vesszük. Ha például a döntési fát logikai formulának tekintjük (lásd 783. oldal (19.1) képlet), akkor egy döntési fa, amely az összes példával konzisztens, kielégíti a (19.3) képleteket. Természetesen abban az esetben, ha a hipotézisek logikai formáját semmiben *nem* korlátozzuk, a *Hipotézis = Besorolások* szintén teljesíti a kényszert. Általában az Ockham borotvája elve azt sugallja, hogy *kis* konzisztens hipotéziseket kellene előnyben részesíteni, valami jobbat kell tehát kitalálnunk annál, minthogy csupán a példákat memorizáljuk.

Az induktív tanulás ezen egyszerű képe az 1980-as évek elejéig uralkodott. A korszerű megközelítés az, hogy olyan ágenseket tervezünk, amelyek már *tudnak valamit*, és amelyek igyekeznek, hogy még többet tudjanak. Lehet, hogy ez nem tűnik félelmetesen mély gondolatnak, azonban igen nagy különbséget jelent, ha az ágensek tervezését tekintjük. Sőt az is lehet, hogy ez a megközelítés valamelyen mértékben hozzájárul annak megértéséhez, hogy a tudomány maga hogyan működik. Az általános séma a 19.6. ábrán látható.



19.6. ábra. A kumulatív tanulási folyamat felhasználja és idővel folyamatosan bővíti a háttértudás készletét

Amennyiben egy háttértudást felhasználó autonóm tanuló ágenst szeretnénk felépíteni, az ágensnek legelőször valamilyen módszerrel meg kell szereznie a háttértudást, hogy utána azt az új tanulási epizódokban felhasználhassa. E módszernek önmagában is egy tanulási folyamatnak kell lennie. Az ágens életútját így *kumulatív* vagy *inkrementális* fejlődés jellemzi. Az ágens feltételezhetően a semmiből kezdhet, egy jó kis induktív program mintájára. Ha egyszer azonban evett már a Tudás Fajából, többé ilyen naiv spekulációra nem adhatja a fejét, és a háttértudást a minél hatásosabb tanulásra kell fordítania. A kérdés most az, hogyan tudjuk ezt megvalósítani.

## Néhány egyszerű példa

Nézzünk néhány „józan ész” példát a háttértudás felhasználásával történő tanulásra. Sok, a következetesen alapuló, látszólag racionális viselkedésről ki fog derülni, hogy ha a megfigyeléseket is figyelembe kell venni, nyilvánvaló lesz, hogy az ágens nem a tiszta indukció egyszerű elvét követi.

- Néha az általános konklúziót már egyetlenegy megfigyelés alapján megfogalmazhatjuk. Gary Larson egy karikatúrájában Zog, a szemüveges ősember egy fanyárs végére tűzött gyíkok süt a tűz felett. Kevésbé intelligens barlangtársainak bámulatba ejtett csoportja figyeli: ők az ennivalót a tűz felett csupasz kezükben tartják. Ez a megvilágosító tapasztalat elegendő, hogy a csoport tagjai a fájdalommentes főzés elvét elsajátítsák.
- Vagy vegyük például a Brazíliába érkező utazó esetét, amikor is az első brazil bennszülötttel találkozik. Hallván, hogy az portugálul beszél, utazónk egyből megállapítja, hogy Brazíliában mindenki beszél portugálul, felfedezvén azonban, hogy a bennszülöttet Fernandónak hívják, ezt a felfedezését az összes brazil férfira egyáltalán nem terjeszti ki. Hasonló esetek a tudományra is jellemzők. Amikor például egy kezdő fizikushallgató egy adott hőmérsékletű rézmintának a sűrűségét és az elektromos vezetőképességét méri, a mérési eredményeket magabiztosan általánosítja a réz minden lehetséges darabkájára. Ha azonban a minta tömegét méri, meg sem fordul a fejében, hogy más rézdarabkáknak is ez lenne a tömege. Ez az általánosító feltevés viszont egészen értelmes lenne, ha például az amerikai 1 centes érméről lenne szó.
- Végül vegyük egy gyógykezelésben járatlan, diagnosztikai szempontból azonban képzett orvostanhallgató esetét, aki egy páciens és egy szakértő belgyógyász konzultációját figyeli. Egy sor kérdés és felelet után a szakértő egy bizonyos antibiotikumos gyógykezelést ír elő a páciens részére. A hallgatónk azt az általános szabályt fogalmazza meg, hogy ez a konkrét antibiotikum hatásos erre a konkrét típusú fertőzésre.

 Ezek mind olyan esetek, amikor a háttértudás felhasználása sokkal gyorsabb tanulást tesz lehetővé ahoz képest, mint amit a tisztán induktív programtól el lehetne várni.

## Néhány általános séma

Az előző példákban azért fordulunk a háttértudáshoz, hogy a megválasztott általánosítást megpróbáljuk igazolni. Most azt nézzük meg, mik azok a vonzatkényszerek, amelyek ott rejtőznek. A kényszerek a *Hipotézis*-en, a megfigyelt *Leírások*-on és a *Besorolások*-on kívül a *Háttértudás*-ra is kiterjednek.

A gyíksütés esetén az ösemberek úgy általánosítottak, hogy *megmagyarázták* a nyárs sikerét: a nyárs megtartja a gyíkot a tűz felett, miközben a kéz sértetlen marad. Ebből a magyarázatból azt az általános szabályt vonhatják le, hogy egy hosszú, merev, hegyes objektum alkalmas a kis, puha testű, ehettő dolgok sütéséhez. Az ilyen általánosító folyamatot **magyarázatalapú tanulásnak** vagy **MAT-nak (explanation-based learning, EBL)** nevezzük. Figyeljük meg, hogy az általános szabály az ösemberek birtokában lévő háttértudásból *logikai módon következik*. A MAT által teljesített vonzatkényszerek tehát a követezők:

$$\text{Hipotézis} \wedge \text{Leírások} \models \text{Besorolások}$$

$$\text{Háttértudás} \models \text{Hipotézis}$$

Az első kényszer a (19.3) képpel megegyezik, azért a MAT-ot eredetileg a példákból való tanulás jobb módszerének tekintették. Azonban amiatt, hogy a háttértudásnak elelegendőnek kell lennie a *Hipotézis* megmagyarázásához is, ami viszont a megfigyelést magyarázza meg, a megfigyelt esetből az ágens igazából semmi tény szerűen újat nem tanul. Az ágens a példát akár ki is következtethette volna az ismert tudásanyag alapján, bár elközelhető, hogy ez elfogadhatatlan mennyiségű számítással járt volna együtt. Újabban a MAT-ot úgy tekintik, mint az egyik olyan módszert, amely elsődleges elméleteknek speciális célú tudásba való konvertálását végzi. A MAT algoritmussal a 19.3. alfejezetben foglalkozunk.

A Brazília utazónak egészén más a helyzete. Az utas szükségszerűen nem is képes megmagyarázni, hogy Fernando miért beszél úgy, ahogy beszél, amíg az idevágó pápai bullákat nem ismeri. Az általánosításra azonban képes lesz a történelemben teljesen járatlan utas is. Ebben az esetben a releváns háttértudás az, hogy minden egyes országban az emberek többsége ugyanazt a nyelvet beszéli. Azt azonban nem tételezzük fel, hogy minden brazil embert Fernandónak hívnak, mert az effajta szabályosság a nevekre nem vonatkozik. Hasonlóképpen, az első éves fizikushallgató is nehezen tud magyarázatot adni a réznél felfedezett vezetési és sűrűségi értékekre. Azt azonban tudja, hogy a vezetést az objektum anyaga és annak hőmérséklete határozza meg együttesen. Mind-egyik esetben az *a priori* *Háttértudás* szerepe az, hogy a célpredikátum szempontjából **releváns tulajdonságalmazok körülhatárolhatók** legyenek. Ez a tudás *megfigyelésekkel együtt* lehetővé teszi, hogy az ágens új, általános szabályt hozzon létre megfigyeléseinek megmagyarázására:

$$\text{Hipotézis} \wedge \text{Leírások} \models \text{Besorolások}$$

$$\text{Háttértudás} \wedge \text{Leírások} \wedge \text{Besorolások} \models \text{Hipotézis}$$

(19.4)

Ezt a fajta általánosítást **relevanciaalapú tanulásnak** vagy **RAT-nak (relevance-based learning, RBL)** nevezzük (bár ez az elnevezés még nem mondható teljesen elfogadott-nak). Vegyük észre, hogy annak ellenére, hogy a RAT a megfigyelések tartalmát felhasználja, a megfigyelések és a háttértudás logikai tartalmán túlmutató hipotéziseket

nem eredményez. Ez a tanulás egy *deduktív* formája, és egymagában nem lehet felelős új tudásanyagnak az alapokból való létrehozásáért.

A szakértőt figyelő orvostanhallgató esetében feltételezzük, hogy a hallgató előzetes tudása elegendő ahhoz, hogy a páciens  $D$  betegségét a tünetekből megállapítsa. Ez azonban nem elegendő annak indoklására, hogy az orvos miért éppen az  $M$  orvosságot írja fel. A hallgatónak egy másik szabályt kell javasolnia, azaz, hogy az  $M$  orvosság a  $D$ -vel szemben hatásos. Ezzel a szabállyal és a hallgató előzetes tudásával most már képes magyarázatot adni, hogy az orvos ebben a konkrét esetben miért az  $M$  gyógyszert írja fel. A példát általánosíthatjuk egy vonzatkényszer megadásával:

$$\text{Háttérítudás} \wedge \text{Hipotézis} \wedge \text{Leírások} \models \text{Besorolások} \quad (19.5)$$



*Példák magyarázatát tehát a háttérítudás és az új hipotézis együttesen adja meg.* A tisztán induktív tanuláshoz hasonlóan a tanuló algoritmusnak a lehető legegyszerűbb és a kényszerekkel konzisztens hipotéziseket kellene előállítania. A (19.5) kényszert teljesítő algoritmusokat **tudásalapú induktív tanulásnak** vagy **TIT-nek (knowledge-based inductive learning, KBIL)** nevezzük.

A 19.5. alfejezetben részletesen leírt TIT algoritmusokat főleg az **induktív logikai programozás, ILP (inductive logical programming)** területén tanulmányozták. Az előzetes tudásnak egy ILP-rendszerben két kulcsszerepe van a tanulás komplexitásának csökkentésében:

1. Tekintettel arra, hogy a megfogalmazott hipotézisek mindegyikének konzisztensnek kell lennie a megfigyelésekkel és az előzetes tudással is, ténylegesen egy redukált hipotézistérrel van dolgunk, hiszen ez a hipotézistér csak azokat az elméleteket tartalmazza, amelyek az eddig ismert dolgokkal konzisztensek.
2. Bármilyen adott megfigyeléshalmaz esetén a megfigyelések magyarázatát megadó hipotézis lényegesen redukált méretű lehet, hiszen a megfigyeléseket magyarázó új szabályok meghatározásához az előzetes tudás is rendelkezésünkre áll. Viszont minél kisebb a hipotézis, annál könnyebb azt megtalálni.

Az a priori tudás befogadásán túl, az ILP-rendszerök képesek arra, hogy hipotéziseket általános elsőrendű logikában és nem a 18. fejezetben használt, korlátozott attribútumalapú nyelven fogalmazzák meg. Ez azt is jelenti, hogy az ILP-rendszerök képesek olyan könyezetekben tanulni, amelyek érhetetlenek az egyszerűbb rendszerek számára.

### 19.3. MAGYARÁZATALAPÚ TANULÁS

Ahogy a fejezet bevezetőjében elmagyaráztuk, a magyarázatalapú tanulás olyan módszer, ami megfigyelésekkel általános szabályokat nyer ki. Tekintsük például az algebrai kifejezések egyszerűsítését és differenciálását (lásd 9.15. feladat).  $X^2$ -et  $X$  szerint differenciálva  $2X$ -et kapunk (vegyük észre, hogy az aritmetikai ismeretlent nagy  $X$  betűvel jelöljük az  $x$  logikai változótól megkülönböztetve). Egy logikai következetető rendszerben a célt így a KÉRDEZ( $\text{Derivált}(X^2, X) = d, TB$ ) kifejezéssel fejezhetjük ki, ahol a megoldás a  $d = 2X$ .

Bárki, aki a differenciáliszámítást ismeri, a megoldást „ránézésre” tudja. Az ilyen problémákkal első alkalommal találkozó hallgatónak vagy netán a tapasztalatot nélkü-

lőző programnak sokkal nehezebb lesz a dolga. A differenciálás közismert szabályai segítségével a kifejezést előbb-utóbb az  $1 \times (2 \times (X^{(2-1)}))$  alakra lehet hozni, és ez előbb-utóbb elvezet minket a  $2X$ -hez. A szerzők logikai programjának ez 136 bizonyítási lépésbe került, amiből 99 lépés a bizonyítás zsákutca elágazásaira esett. Ezt tapasztalva azt szeretnénk, ha a program ugyanennek a problémának a megoldását a következő alkalommal lényegesen gyorsabban adná meg.

A memoizálás (memoization) módszerét a számítógép-tudományban régóta alkalmazzák, miszerint a számítások megygyorsíthatók az eredmények eltárolása révén. A memofüggvény alapgondolata, hogy bemenet/kimenet párokat gyűjtünk ki egy külön adatbázisba. Meghíváskor a függvény először megvizsgálja az adatbázist, hátha a probléma teljes újbóli megoldását így meg tudja kerülni. A magyarázatalapú tanulás ezt az ötletet továbbfejleszti úgy, hogy egy általános szabályt fogalmaz meg, amely az esetek egész osztályát képes lefedni. A differenciálás esetében a memoizálás emlékezne ugyan arra, hogy az  $X^2$ -nek  $X$  szerinti deriváltja  $2X$ , a  $Z^2$  Z szerinti deriváltjának kiszámítását azonban teljes egészében az ágensre hagyná. Célszerű lenne, ha egy olyan általános szabályt<sup>2</sup> tudnánk megfogalmazni, amely minden lehetséges  $u$  aritmetikai ismeretlen esetén megadná, hogy az  $u^2$ -nek  $u$  szerinti deriváltja  $2u$ . A logikában ezt az alábbi szabállyal fejezzhetjük ki:

$$\text{AritmetikaiIsmeretlen}(u) \Rightarrow \text{Derivált}(u^2, u) = 2u$$

Ha a tudásbázis rendelkezik egy ilyen szabállyal, akkor minden új eset, ami a szabály egy példánya, azonnal megoldható.

Természetesen ez csupán egy triviális példa egy igen általános jelenségre. Ha egyszer megértettük valamit, azt általánosítva más körülmények között is felhasználhatjuk. Egy „nyilvánvaló” megoldási lépést kapunk, amely építőkockaként felhasználható bonyolultabb problémák megoldásában. Alfred North Whitehead, aki Bertrand Russell-lel együtt a *Principia Mathematica* társszerzője – talán a Zog felfedezése jellegű események megértéséhez saját magára alkalmazva a MAT elvét – azt írta, hogy „*A civilizáció azáltal halad előre, hogy szaporítja azoknak a fontos cselekvéseknek a számát, amelyeket gondolkodás nélkül véghezvihetünk*” (Whitehead, 1911). Ha ön, kedves olvasó, a deriválási példa lényegét megértette, akkor az agya már buzgón azon dolgozik, hogy megkísérítse a magyarázatalapú tanulás általános elvét kinyerni ebből a példából. Figyelje meg, hogy ha csak nem lényegesen okosabb a szerzőknél, a magyarázatalapú példa bemutatása előtt még nem ismerte fel a MAT-ot. A Zogot figyelő ósemberekhez hasonlóan önnel (és nekünk is) egy példát kellett látnunk, mielőtt az alapelvet meg tudtuk fogalmazni. Ez azért van így, mert *megmagyarázni*, hogy *miért* jó egy ötlet, sokkal könnyebb, mint magát az ötletet megfogalmazni.

## Általános szabályok kinyerése példákból

A MAT alapötlete az, hogy először az előzetes tudásra alapozva megkonstruáljuk a megfigyelés magyarázatát, majd meghatározzuk annak az esetosztálynak a definícióját, amelyre a megkonstruált magyarázat alkalmazható. A definíció alapul szolgál az osz-

<sup>2</sup> Természetesen az  $u^n$  általános szabályát is meg tudnánk valósítani, a mondanivalót azonban a közölt példa is jól illusztrálja.

tály eseteit lefedő szabály számára. A „magyarázat” lehet egy logikai bizonyítás, de általánosságban lehet akármilyen következetetési vagy problémamegoldó folyamat, feltéve, hogy a lépései jól definiáltak. Kulcsfontosságú azoknak a szükséges feltételeknek az azonosítása, amelyek révén az egyes lépések más esetre is alkalmazhatók lesznek.

Következtető rendszerként a 9. fejezetben leírt egyszerű, hátrahaladó tételbizonyító rendszert fogjuk használni. A  $\text{Derivált}(X^2, X) = 2X$  bizonyítási fája túlságosan nagy ahhoz, hogy példaként szerepeljen, ezért az általánosítás módszertanát egy valamivel egyszerűbb példával fogjuk illusztrálni. Tegyük fel, hogy az  $1 \times (0 + X)$  kifejezést akarjuk egyszerűsíteni. A tudásbázis az alábbi szabályokat tartalmazza:

$$\text{Átírás}(u, v) \wedge \text{Egyszerűsítés}(v, w) \Rightarrow \text{Egyszerűsítés}(u, w)$$

$$\text{Primitív}(u) \Rightarrow \text{Egyszerűsítés}(u, u)$$

$$\text{AritmetikaiIsmeretlen}(u) \Rightarrow \text{Primitív}(u)$$

$$\text{Szám}(u) \Rightarrow \text{Primitív}(u)$$

$$\text{Átírás}(1 \times u, u)$$

$$\text{Átírás}(0 + u, u)$$

$$\vdots$$

Annak bizonyítása, hogy a válasz  $X$ , a 19.7. ábra felső részében látható. A MAT-módszer egyidőben két bizonyítási fát konstruál. A második bizonyítási fa a *szabad változókkal rendelkező* cél használja, ahol az eredeti cél konstansait változókkal helyettesítettük. Ahogy az eredeti bizonyítás halad előre, úgy halad a szabad változós bizonyítás is, *pontosan ugyanazokat a szabályokat alkalmazva*. Lehetséges, hogy néhány változót közben le kell kötni. Ahhoz, hogy az  $\text{Átírás}(1 \times u, u)$  szabályt használhassuk, az  $\text{Átírás}(x \times (y + z), v)$  részcél  $x$  változóját 1-re kell lekötni. Hasonlóképpen az  $\text{Átírás}(y + z, v')$  részcél  $y$  változóját 0-ra kell lekötni, ha az  $\text{Átírás}(0 + u, u)$  szabályt szeretnénk alkalmazni. Az általánosított bizonyítási fa birtokában a levelekből (a szükséges kötéseket figyelembe véve) a célpredikátum általános szabályát képezzük:

$$\begin{aligned} & \text{Átírás}(1 \times (0 + z), 0 + z) \wedge \text{Átírás}(0 + z, z) \wedge \text{AritmetikaiIsmeretlen}(z) \\ & \Rightarrow \text{Egyszerűsítés}(1 \times (0 + z), z) \end{aligned}$$

Vegyük észre, hogy a bal oldal első két feltétele igaz *függetlenül attól, hogy z-nek mi az értéke*. Így ezeket kihagyhatjuk a szabályból, és eredményül azt kapjuk, hogy:

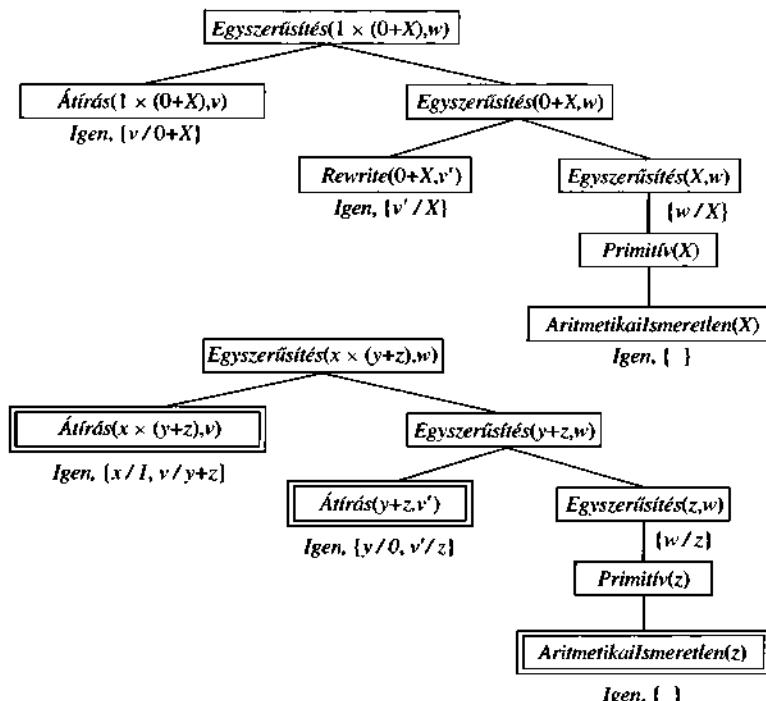
$$\text{AritmetikaiIsmeretlen}(z) \Rightarrow \text{Egyszerűsítés}(1 \times (0 + z), z)$$

Általánosságban elmondható, hogy a végeleges szabályból azok a feltételek hagyhatók ki, amelyek nem jelentenek kényszert a szabály jobb oldalán lévő változókra vonatkozólag. Az eredményül kapott szabály továbbra is igaz, sőt még hatékonyabb is lesz. Vegyük észre azt is, hogy az  $\text{AritmetikaiIsmeretlen}(z)$  feltétel nem hagyható ki, hiszen a  $z$ -nek nem minden lehetséges értéke aritmetikai ismeretlen. A  $z$  más jellegű értékei esetén feltehetően más egyszerűsítési szabályokat kellene alkalmazni – például ha  $z$  netán  $2 \times 3$  lenne, akkor az  $1 \times (0 + (2 \times 3))$  helyes egyszerűsítése 6, és nem  $2 \times 3$  lenne.

Összegezve, az alap MAT-módszer a következőképpen működik:

- Ha adott egy példa, a rendelkezésre álló háttérítést felhasználva konstruálunk egy bizonyítást arra, hogy a célpredikátum érvényes a példára.

2. Az előbbivel egy időben az eredeti bizonyítás következetesi lépéseihez alkalmazva konstruáljuk meg a szabad változós cél általánosított bizonyítási fáját.
3. Konstruáljuk meg az új szabályt úgy, hogy a szabály bal oldala a fa leveleiből áll, a jobb oldala viszont a szabad változós cél (az általánosított bizonyítás változókötései figyelembe véve).
4. Hagyuk ki azokat a feltételeket, amelyek a célban foglalt változók értékeitől nem függnek.



**19.7. ábra.** Az egyszerűsítési probléma bizonyítási fái. Az első fa az eredeti problémápéldány bizonyítása, amiből az  $AritmetikaiIsmeretlen(z) \Rightarrow Egyszerűsítés(1 \times (0 + z), z)$  vezethető le. A második fa az a bizonyítás, amikor az eredeti problémápéldányban szereplő összes konstanst változókkal helyettesítettük, amiből viszont további szabályok sokaságát vezethetjük le.

## A hatékonyság javítása

A 19.7. ábrán látható általánosított bizonyítási fából több általánosított szabályt is kíepesek vagyunk kinyerni. Ha a fa jobb oldali ágának növekedését leállítjuk, vagy az ágat **lemettssük (prune)**, amikor *Primítív* lépéshez jutottunk, az alábbi szabályt kapjuk:

$$Primítív(z) \Rightarrow Egyszerűsítés(1 \times (0 + z), z)$$

Ez a szabály annyira érvényes, mint az *AritmetikaiIsmeretlen*-t felhasználó szabály, bár annál általánosabb, mivel olyan esetekre is vonatkozik, amikor  $z$  numerikus értékű.

Még általánosabb szabályokat is kinyerhetünk, ha az *Egyszerűsítés*( $y + z, w$ ) után metszünk. Ekkor a szabály:

$$\text{Egyszerűsítés}(y + z, w) \Rightarrow \text{Egyszerűsítés}(1 \times (y + z), w)$$

Általánosságban elmondható, hogy szabályokat az általánosított bizonyítási fa *bármely részfájából* nyerhetünk. Szembe kell néznünk azonban azzal a problémával, hogy ezek után melyik szabályt válasszuk.

Az a döntés, hogy melyik szabályt érdemes létrehozni, végső soron a hatékonyságon műlik. A MAT által biztosított hatékonyságnövekedésnek három tényezője van:

1. A tudásbázishoz nagyszámú szabály hozzáadása a következetetői folyamatot lelassíthatja, mivel a következetetői mechanizmusnak ezeket a szabályokat akkor is meg kell vizsgálnia, amikor ezek nem járulnak hozzá a megoldáshoz. Másképpen fogalmazva, a keresési tér **elágazási tényezője (branching factor)** ilyenkor növekszik.
2. Ahhoz, hogy ezt a jelenséget kompenzáljuk, a létrehozott szabályoknak lényeges sebességnövekedést kell garantálniuk az általuk lefedett problémapéldányok esetében. Az ilyen sebességnövekedés főleg abból származik, hogy a származtatott szabályok révén elkerülhetjük azokat a holtágakat, amelyekbe különben a bizonyítás során belemennénk, illetve abból, hogy a bizonyítások rövidebbek lesznek.
3. A származtatott szabályoknak a lehető legáltalánosabbaknak kell lenniük, hogy a lehető legnagyobb esethalmazt fedjék le.

A származtatott szabályok hatékonyságát megsokott módon úgy biztosíthatjuk, hogy megköveteljük a szabály minden részcéljától, hogy **hatásos (operational)** legyen. Durván fogalmazva, egy cél hatásos, ha „könnű” megoldani. A *Primitív(z)* részcélt például könnyű megoldani, a megoldásához két lépés elegendő. Ezzel szemben az *Egyszerűsítés(y + z, w)* részcél tetszőleges számú következetetőhez vezethet az y és a z értékeinek függvényében. Ha az általánosított bizonyítás megkonstruálásakor a hatásossági tesztet minden lépésnél elvéggezzük, az ágat azonnal lemeteszhetjük, amint egy hatásos részcélra rátaláltunk. A hatásos részcélt ilyenkor az új szabály egy konjunktív elemeként megtartjuk.

Sajnos a hatásosság és az általánosság között általában kompromisszumot kell kötni. A konkrétabb részcélokat könnyebb megoldani, viszont azok kevesebb esetet fednek le. A hatásosság tovább fokozható; egy vagy két lépés nyilvánvalóan hatásos, de mi a helyzet 10-zel vagy 100-zal? Egy adott részcél megoldásának a költsége végül attól függ, hogy a tudásbázis milyen egyéb szabályokat tartalmaz. A költség növekedhet vagy csökkenhet, ahogy a tudásbázishoz új szabályokat adunk. Az adott kezdeti tudásbázis hatékonyságának maximalizálásánál a MAT-rendszer valóban igen komplex optimizációs problémával kerülnek szembe. Néha megalkotható annak matematikai modellje, hogy egy adott szabály hozzáadása általában milyen hatással van a hatékonyságra, és ennek a modellnek a használatával a hozzáadandó legjobb szabály megválasztható. Az elemzés azonban igen komplikált lehet, különösképpen ha rekurzív szabályokkal van dolgunk. Ígéretes megközelítés a hatékonyság empirikus vizsgálata, amikor néhány szabály hozzáadásával meggyőződünk arról, hogy a szabályok közül melyik az, amelyik ténylegesen hasznos, és amelyik az eljárásokat felgyorsítja.

A hatékonyság empirikus elemzésének gondolata a MAT lényegét érinti. Amit eddig informálisan „az adott tudásbázis hatékonyságának” nevezünk, az nem más, mint egy

átlagos esetkomplexitás a megoldandó problémák egy eloszlásán mérve. A régi példák általánosításával a MAT növeli a tudásbázis hatékonyságát a jövőben ésszerűen várható problémák szempontjából. Az ötlet addig jó, amíg a régi példák eloszlása durván megegyezik a jövőbeli példák eloszlásával. Ez a feltételezés azonos azzal, amit a 18.5. alfejezetben a VKH-tanulás kapcsán megtettünk. Ha a MAT-rendszert gondosan kiviteleztek, a későbbi problémáknál lényeges javulás érhető el. Egy nagyon nagy méretű, svéd és angol nyelv közötti beszédfordításra kifejlesztett, Prolog-alapú, természetes nyelvi rendszerben például, a valós idejű fordítási képességet csak az elemző folyamatra alkalmazott MAT révén sikerült elérni (Samuelsson és Rayner, 1991).

## 19.4. TANULÁS RELEVÁNS INFORMÁCIÓ ALAPJÁN

Úgy tűnik, hogy a Brazíliába utazónk képes a Brazíliában beszélt nyelvre vonatkozó megbízható általánosításra. A következtetésre az a háttértudása jogosítja, miszerint egy adott ország lakói (általában) ugyanazt a nyelvet beszélik. Ezt a tudást az elsőrendű logikában az alábbi módon fejezhetjük ki:<sup>3</sup>

$$\text{Nemzetisége}(x, n) \wedge \text{Nemzetisége}(y, n) \wedge \text{Nyelve}(x, l) \Rightarrow \text{Nyelve}(y, l) \quad (19.6)$$

(szó szerint: „Ha  $x$  és  $y$  azonos  $n$  állampolgárságú, és  $x$  az  $l$  nyelvet beszéli, akkor  $y$  szintén az  $l$  nyelvet beszéli.”). Nem nehéz kimutatni, hogy a fenti kijelentésből és a

$$\text{Nemzetisége}(\text{Fernando}, \text{Brazil}) \wedge \text{Nyelve}(\text{Fernando}, \text{Portugál})$$

megfigyelésből logikailag következik (lásd 19.1. feladat), hogy:

$$\text{Nemzetisége}(x, \text{Brazil}) \Rightarrow \text{Nyelve}(x, \text{Portugál})$$

A (19.6) típusú mondatok szigorú relevanciát fejeznak ki: a nemzetiség teljesen meghatározza a nyelvet. Másnéven fogalmazva: a nyelv a nemzetiség függvénye. Az ilyen mondatokat **funkcionális függőségeknek (functional dependencies)** vagy **meghatározásoknak (determinations)** nevezzük. Bizonyosfajta alkalmazásoknál (például az adatbázisrendszer-tervez specifikálásánál) ezek annyira általánosan fordulnak elő, hogy a felírásukhoz speciális szintaxist használnak. Davies (Davies, 1985) jelölésével élve:

$$\text{Nemzetisége}(x, n) \succ \text{Nyelve}(x, l)$$

Ez persze csak egy egyszerű szintaktikai nyelánkság, azonban világosan mutatja, hogy egy meghatározás valójában predikátumok közötti reláció: a nemzetiség meghatározza a nyelvet. Az anyag vezetőképességét és sűrűségét meghatározó releváns tulajdonságokat hasonlóképpen kifejezhetjük:

$$\text{Anyaga}(x, m) \wedge \text{Hőmérséklete}(x, t) \succ \text{Vezetőképessége}(x, p)$$

$$\text{Anyaga}(x, m) \wedge \text{Hőmérséklete}(x, t) \succ \text{Sűrűsége}(x, d)$$

A hozzá tartozó általánosítások a meghatározásokból és a megfigyelésekkel származnak.

<sup>3</sup> Az egyszerűség kedvéért feltételezzük, hogy egy személy csak egy nyelven beszél. Az olyan országok esetén, mint Svájc vagy India, a szabályt természetesen bővíteni kell.



## A hipotézistér meghatározása

Annak ellenére, hogy meghatározások igazolják az összes brazilra vagy az adott hőmér-séklelű réz minden darabjára vonatkozó általános következetések levonását, ahhoz természetesen nem elegendők, hogy egyetlenegy példából következtessünk ki egy általános prediktív tulajdonságú elmeletet, amely az összes nemzetiségre, az összes hőmérsékletre és anyagra vonatkozik. Hatásuk úgy képzelhető el legjobban, hogy leszű-kítik a tanuló ágens által figyelembe veendő hipotézisek terét. A vezetőképesség előre-jelzésénél például csupán az anyagra és a hőmérsékletre kell koncentrálnunk, figyelmen kívül hagyva a tömeget, a tulajdonviszonyokat, a hétfajtát, a jelenlegi államelnököt stb. A hipotézisek természetesen tartalmazhatnak olyan komponenseket, mint például a molekuláris struktúra, a termikus energia, a szabad elektronok sűrűsége, amit az anyag és a hőmérséklet befolyásol. *A meghatározások egy elégséges alapszótárt adnak meg ahhoz, hogy a célpredikátumra vonatkozó hipotéziseket megkonstruáljuk.* Ezt a kijelen-tést úgy láthatjuk be, hogy megmutatjuk: egy adott meghatározás logikailag ekvivalens azzal az állítással, miszerint a célpredikátum helyes definíciója a meghatározás bal oldalán lévő predikátumok által kifejezhető definíciók egyike.

Intuitív módon érthető, hogy a hipotézistér méretének számítási redukálásának könnyebbé kell tennie a célpredikátum megtanulását. A számítási tanulás elmelet alapered-ményeit (lásd 18.5. alfejezetet) felhasználva, a lehetséges nyereség mennyiségileg is kife-jezhető. Emlékezzünk arra, hogy egy Boole-függvény esetén  $\log(|H|)$  számú példa (ahol  $|H|$  a hipotézistér mérete) szükséges ahhoz, hogy egy elfogadható hipotézishez konver-gáljunk. Ha a tanuló  $n$  Boole-tulajdonsággal rendelkezik a hipotézisek megkonstruálásá-hoz, akkor egyéb korlátozások hiányában  $|H| = O(2^n)$ , így a példák száma  $O(2^n)$ . Ha a meghatározás bal oldalán  $d$  számú predikátum áll, a tanulónak  $O(2^d)$  példára van szük-sége, ami  $O(2^{n-d})$  méretű redukciót jelent. Elfogult hipotézistér, például konjunktív el-fogultságok esetén, a redukció még mindig számottevő lesz, bár kevésbé lesz látványos.

```
function MINIMALIS-KONZISZTENS-MEGH(E, A) returns az attribútumok egy halmaza
    inputs: E, a példák egy halmaza
    A, az n nagyságú attribútumok egy halmaza
```

```
for i ← 0, ..., n do
    for each  $A_i$ , az A i nagyságú részhalmaza do
        if KONZISZTENS-MEGH( $A_i$ , E) then return  $A_i$ 
```

```
function KONZISZTENS-MEGH?(A, E) returns egy igazságérték
```

```
inputs: A, az attribútumok egy halmaza
```

```
E, a példák egy halmaza
```

```
local variables: H, egy hash-tábla
```

```
for each e példa in E do
    if valamelyik példa a H-ban az A attribútumokra ugyanolyan értékű, mint e,
        de eltérő besorolással rendelkezik then return hamis
    tároljuk e besorolását H-ban, az e példa A attribútumainak értékeivel indexelve
    return igaz
```

19.8. ábra. Algoritmus egy minimális konzisztens meghatározás megkeresésére

## Tanulás releváns információ felhasználásával

Ahogy ezt a fejezet bevezetőjében már megemlítettük, az előzetes tudás hasznos a tanulás szempontjából, azonban valahogy ezt is meg kell tanulni. Hogy a relevanciaalapú tanulást teljes mértékben megfogalmazhassuk a meghatározások tanulási mechanizmusát meg kell még adnunk. A most bermutatásra kerülő algoritmus alapja a megfigyelésekkel konzisztens legegyszerűbb meghatározás megtalálása. Egy  $P \succ Q$  meghatározás azt jelenti, hogy ha egy példa illeszkedik  $P$ -re, akkor  $Q$ -ra is kell illeszkednie. Egy meghatározás konzisztens a példák egy halmazával, ha minden példapár, amely a bal oldali predikátumokon illeszkedik egymásra, a célpredikátumon is illeszkedik, magyarán, ha a besorolása azonos. Tegyük fel, hogy bizonyos anyagi mintákat megmérve a vezetőképesség alábbi méréseivel rendelkezünk:

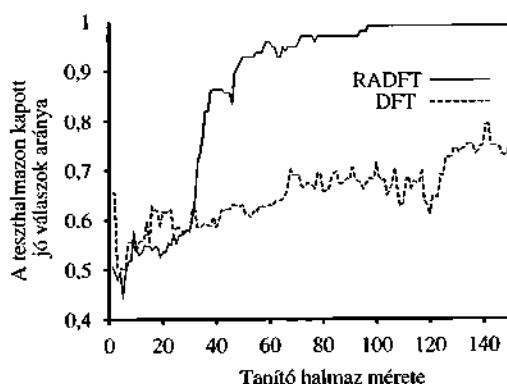
Minta	Tömeg	Hőmérséklet	Anyag	Méret	Vezetőképesség
S1	12	26	rész	3	0,59
S1	12	100	rész	3	0,57
S2	24	26	rész	6	0,59
S3	12	26	ólom	2	0,05
S3	12	100	ólom	2	0,04
S4	24	26	ólom	4	0,05

A minimális konzisztens meghatározás:  $\text{Anyaga} \wedge \text{Hőmérséklete} \succ \text{Vezetőképessége}$ . Létezik konzisztens, de nem minimális meghatározás is:  $\text{Tömege} \wedge \text{Nagysága} \wedge \text{Hőmérséklete} \succ \text{Vezetőképessége}$ . Ez azért konzisztens a példákkal, mert a tömeg és a méret meghatározza a sűrűséget, és az adatbázisunkban nincs két különböző anyag, amelynek sűrűsége azonos lenne. Mint mindig, most is nagyobb példahalmazra lenne szükségünk, hogy a közel helyes hipotézist kiszűrjük.

A minimális konzisztens meghatározás megkeresésének több lehetséges algoritmusára létezik. Kézenfekvő megközelítés a meghatározások terében keresni, először az egypredikátumos, a kétpredikátumos stb. meghatározásokat ellenőrizve, amíg rá nem lelünk egy konzisztens meghatározásra. Egyszerű attribútumalapú ábrázolást tételezzünk fel, hasonlóan ahhoz, mint amit a döntési fák tanulásánál a 18. fejezetben használtunk. Tekintettel arra, hogy a célpredikátumot rögzítettnek vesszük, a  $d$  meghatározást a bal oldal attribútumhalmaza fogja képviselni. Az algoritmust a 19.8. ábra mutatja.

Ennek az algoritmusnak az időigénye a legkisebb konzisztens meghatározás nagyságától függ. Tegyük fel, hogy az ilyen meghatározásnak  $p$  attribútuma van a lehetséges  $n$ -ből. Az algoritmus addig nem lesz képes a meghatározást megtalálni, amíg az  $A$ -nak  $p$  nagyságú részhalmazait nem kezdi végigvizsgálni. Miután  $\binom{n}{p} = O(n^p)$  ilyen részhalmaz létezik, következésképpen az algoritmus a minimális meghatározás méretét tekintve exponenciális lesz. Az is kiderült, hogy a probléma NP-teljes, így általános esetben sem várhatunk jobbat. A legtöbb tárgytartomány azonban elegendő lokális strukturáltsággal (a lokálisan strukturált tárgytartomány definíciójára lásd 14. fejezet) rendelkezik, így  $p$  minden bizonytalannal kicsi lesz.

Ha a meghatározások tanuló algoritmusára már adott, a tanuló ágensnek módja van egy olyan minimális hipotézist megkonstruálni, amelyen belül kell a célpredikátumot meg-tanulnia. Összemásolhatjuk a MINIMÁLIS-KONZISZTENS-MEGH és a DÖNTÉSI-FA-TANULÁS algoritmusait, és így az RADFT-t. A relevanciaalapú döntési fa tanuló algoritmust kap-juk, amely először a releváns attribútumok minimális halmazát azonosítja, majd ezen halmazt tanulás céljából döntési fa algoritmusának adja tovább. A DÖNTÉSI-FA-TANULÁS-sal ellentétben az RADFT egyidejűleg tanulja és használja a releváns információt ahhoz, hogy a hipotézisteret minimalizálja. Elvárjuk, hogy az RADFT tanulási görbüje a DÖNTÉSI-FA-TANULÁS tanulási görbüjénél jobb legyen, és tényleg ez a helyzet. A 19.9. ábrán a két algoritmus tanulási görbüje látható, véletlen módon generált adatokon és egy olyan tanulandó függvény esetén, amely a lehetséges 16 attribútumból csupán 5-öt használ fel. Természetesen azokban az esetekben, amikor az összes attribútum releváns, az RADFT nem mutat előnyöket.



19.9. ábra. Az RADFT és a DÖNTÉSI-FA-TANULÁS hatékonyságának összehasonlítása véletlen módon generált adatok és egy olyan célfüggvény esetén, amely a lehetséges 16 attribútumból csak 5-öt használ

Ez a rész csak érintette a **deklaratív elfogultság (declarative bias)** területét, melynek az a célja, hogy megértsük, hogy az előzetes tudást hogyan kell felhasználni a megfele-lő hipotézistér azonosításához, amelyben a korrekt céldefiníciót keressük. Sok kérdés megválaszolatlan maradt:

- Hogyan kell az algoritmust kiterjeszteni a zaj kezelésére is?
- Tudunk-e folytonos értékeket felvezvő változókat kezelni?
- Hogyan lehetne felhasználni az előzetes tudás más típusát is, a meghatározásokon kívül?
- Hogyan lehet az algoritmusokat általanosítani, hogy egy elsőrendű elméletre, és nem csupán az attribútumalapú reprezentációra vonatkozzanak ?

Az egyes kérdésekre a következő alfejezetben keresünk választ.

## 19.5. INDUKTÍV LOGIKAI PROGRAMOZÁS

Az induktív logikai programozás (ILP) az induktív módszereket az elsőrendű reprezentációk erejével kombinálva, az elmeletek logikai programok formájában történő kifejezésére helyezi a hangsúlyt.<sup>4</sup> Az ILP három oknál fogva terjedt el. Először is az ILP az általános tudásalapú induktív tanulási probléma precíz, szigorú megközelítését adja. Másodszor, teljes algoritmusokat szolgáltat az általános elsőrendű elmeletek indukciós úton, példák alapján történő előállítására. Következésképpen olyan területeken is képes sikeresen tanulni, ahol az attribútumalapú algoritmusokat nehéz alkalmazni. Ennek egy példája a fehérjemolekulák összehajtогatásának a tanulása (lásd 19.10. ábra). A fehérjemolekula háromdimenziós konfigurációját egy attribútumhalmazzal ésszerűen kifejezni nem lehet, mert a konfiguráció lényege az objektumok közötti relációkra vonatkozik, és nem az egyes objektumok attribútumaira. A relációk leírására megfelelő apparátus az elsőrendű logika. Harmadszor, induktív logikai programozás által létrehozott hipotézisek emberek számára (viszonylag) könnyen olvashatók. A 19.10. ábrabeli természetes nyelvű fordítást a gyakorló biológusok végiglemezhetik és kritizálhatják. Ez azt jelenti, hogy az induktív logikai programrendszer részt vehetnek a kísérletezés, a hipotézisgenerálás, a megvitatás és a cífolat tudományos ciklusában. Az ilyen részvétel a „fekete doboz” osztályozókat gyártó módszerek számára, mint amilyenek például a neurális hálók, lehetetlen lenne.

### Egy példa

Emlékezzünk a (19.5) egyenlet alapján, hogy az általános tudásalapú indukciós probléma az alábbi vonzatkéntyszer:

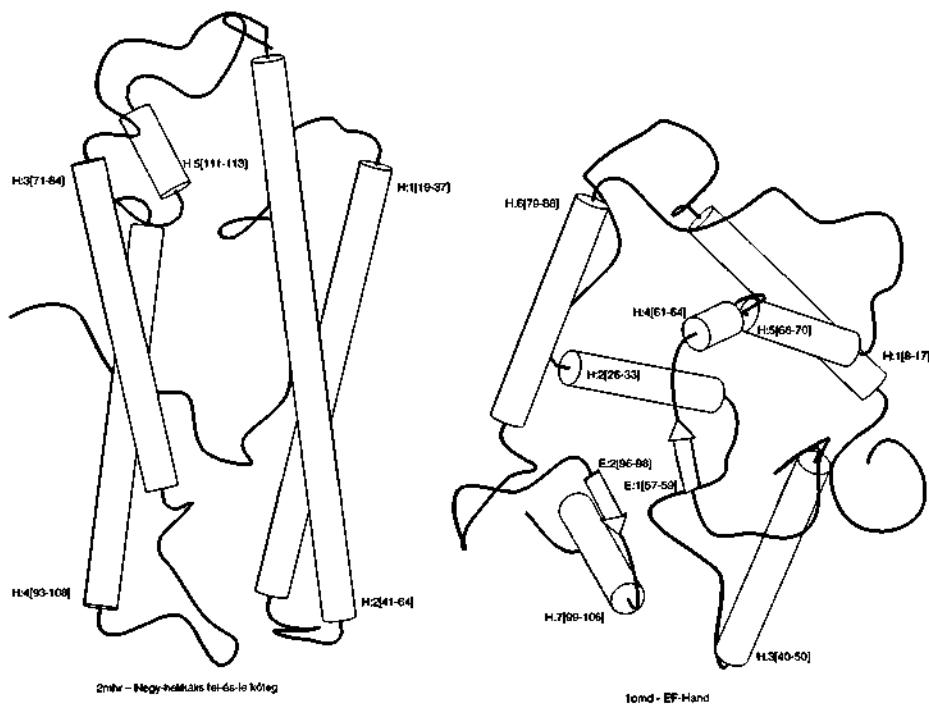
*Háttértudás  $\wedge$  Hipotézis  $\wedge$  Leírások  $\models$  Besorolások*

„megoldása” az ismeretlen *Hipotézis-re* nézve, feltéve, hogy adottak a *Háttértudás*, valamint a *Leírások* és a *Besorolások* által leírt példák. Hogy ezt megvilágítsuk, tekintsük a rokonsági relációk példák alapján történő tanulását. A megfigyeléseket egy kiterjedt családfa képezi, az *Anyja*, *Apja*, *Házas* relációkkal, valamint a *Férfi* és *Nő* tulajdon-ságokkal leírva. A 8.11. feladat családfáját fogjuk használni, amit a 19.11. ábra mutat. A példához tartozó leírások az alábbiak:

<i>Apja(Fülöp, Károly)</i>	<i>Apja(Fülöp, Anna)</i>	...
<i>Anyja(Mami, Margit)</i>	<i>Anyja(Mami, Erzsébet)</i>	...
<i>Házas(Diana, Károly)</i>	<i>Házas(Erzsébet, Fülöp)</i>	...
<i>Férfi(Fülöp)</i>	<i>Férfi(Károly)</i>	...
<i>Nő(Beatrix)</i>	<i>Nő(Margit)</i>	...

A *Besorolások* állításai azon múlnak, hogy milyen célfogalmat szeretnénk megtanulni. A célpredikátumok olyan fogalmak lehetnének, mint például a *Nagyszülője*, a *Ságora*, illetve az *őse*. A *Nagyszülője* esetén a *Besorolások* teljes halmaza  $20 \times 20 = 400$

<sup>4</sup> Azt javasoljuk, hogy ezen a ponton az olvasó ismételten fussen végig a 9. fejezetben bemutatott néhány fogalmon, beleértve a Horn-klózokat, a konjunktív normál formát, az egyesítést és a rezolúciót.



(a)

(b)

**19.10. ábra.** Az (a) és (b) ábra a fehérjemolekula összehajtjogatásának tárgyterületén a „négy-helikális fel-és-le köteg” fogalom pozitív és negatív példáját mutatja. Mindkét példa struktúráját egy olyan kb. 100 konjunktív tagot tartalmazó logikai kifejezésre kódolták, mint amilyen például a *TeljesHossz(D2mhr, 118)  $\wedge$  HelikálisSzám(D2mhr, 6)  $\wedge \dots$*  kifejezés. Az ilyen leírásokból és az Összehajtjogatás(NÉGY-HELIKÁLIS-FEL-ÉS-LE-KÖTEG, D2mhr) jellegű besorolásokból a PROGOL induktív logikai programrendszer (Muggleton, 1995) az alábbi szabályt tanulta meg:

*Összehajtjogatás(NÉGY-HELIKÁLIS-FEL-ÉS-LE-KÖTEG, f)  $\leftarrow$   
 Helikális(f, cs<sub>1</sub>)  $\wedge$  Hossz(cs<sub>1</sub>, MAGAS)  $\wedge$  Pozíció(f, cs<sub>1</sub>, n)  
 $\wedge$  (1  $\leq$  n  $\leq$  3)  $\wedge$  Szomszédos(f, cs<sub>2</sub>, cs<sub>2</sub>)  $\wedge$  Helikális(f, cs<sub>2</sub>)*

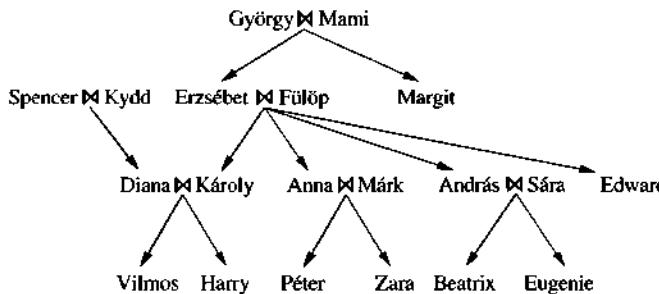
Ilyen típusú szabályt az előbbi fejezetekben látott, attribútumalapú mechanizmusokkal megtanulni, sőt kifejezni sem lehet. A szabály a természetes nyelvben kifejezve:

Az *F* fehérje a „Négy-helikális fel-és-le köteg” összehajtjogatási osztályhoz tartozik, ha egy hosszú *cs<sub>1</sub>* helikális tartalmaz a másodlagos struktúra 1. és 3. közötti pozícióban, valamint *cs<sub>1</sub>* a második helikálissal szomszédos.

*Nagyszülője(Mami, Károly)      Nagyszülője(Erzsébet, Beatrix)      ...  
 $\neg$ Nagyszülője(Mami, Harry)       $\neg$ Nagyszülője(Spencer, Péter)      ...*

alakú konjunktóból áll. Természetesen e teljes halmaz részhalmazából is tudnánk tanulni.

Az induktív tanuló program tárgya egy olyan állításhalmaz *Hipotézis*-ként való előállítása, amely a vonzatkényszert kielégíti. Átmenetileg tegyük fel, hogy az ágens nem rendelkezik háttértudással, a *Háttértudás* üres. Akkor egy lehetséges megoldás a *Hipotézis*-re a következő:



19.11. ábra. Egy tipikus családfa

$$\begin{aligned} \text{Nagyszülője}(x, y) \Leftrightarrow & [\exists z \text{ Anyja}(x, z) \wedge \text{Anyja}(z, y)] \\ \vee & [\exists z \text{ Anyja}(x, z) \wedge \text{Apja}(z, y)] \\ \vee & [\exists z \text{ Apja}(x, z) \wedge \text{Anyja}(z, y)] \\ \vee & [\exists z \text{ Apja}(x, z) \wedge \text{Apja}(z, y)] \end{aligned}$$

Vegyük észre, hogy egy attribútumalapú tanuló algoritmus, mint például a DÖNTÉSI-FATANULÁS sehogy sem fog ezzel a problémával boldogulni. Ahhoz, hogy a Nagyszülője-t attribútumként (azaz egy unáris predikátumként) fejezzük ki, az embereket párosával objektumként kell tudnunk kezelni:

*Nagyszülője(⟨Mami, Károly⟩)...*

Ezek után ott akadunk el, amikor megkíséreljük egy példa leírását formálisan reprezentálni. Csupán olyan elrettentő attribútumokat tudnánk használni, mint az:

*ErzsébetAnyjaAzElsőElem(⟨Mami, Károly⟩)*

A Nagyszülője-nek az ilyen attribútumokkal kifejezett definíciója nem lesz más, mint egy nagy diszjunkció, ami az egyes konkrét esetekből áll, és amit lehetetlen az új esetekre általánosítani. Az attribútumalapú tanuló algoritmusok képtelenek relációs predikátumokat megtanulni. Az ILP egyik legfontosabb előnye tehát az, hogy a problémák sokkal szélesebb választékában alkalmazhatók, beleértve a relációs problémákat is.

Az olvasó természetesen észrevései, hogy egy kis háttértdás segítene a Nagyszülője definíciójának reprezentációjánál. Így például, ha a Háttértdás tartalmazná az alábbi állítást:

*Szülője(x, y)  $\Leftrightarrow$  [Anyja(x, y)  $\vee$  Apja(x, y)]*

akkor a Nagyszülője definícióját a:

*Nagyszülője(x, y)  $\Leftrightarrow$  [\exists z Szülője(x, z)  $\wedge$  Szülője(z, y)]*

állításra lehetne redukálni. Ez megmutatja, hogy mennyire tud a háttértdás hozzájárulni a példák megmagyarázásához szükséges hipotézis méretének drasztikus csökkenéséhez.

Az ILP-algoritmus új predikátumokat is létesíthet, hogy a magyarázó jellegű hipotézis kifejezését egyszerűsítse. Az előbbi példát tekintve, teljesen ésszerű egy pótpredikátumot javasolni – amit „Szülője”-nek tudnánk nevezni –, hogy a célpredikátum definícióját egyszerűsítsek. Azok az algoritmusok, amelyek új predikátumokat tudnak létesíteni, az ún.

**konstruktív indukciós (constructive induction)** algoritmusok. Világos, hogy a konstruktív indukció a bevezetőben vázolt kumulatív tanulás egy szükséges komponense. A kumulatív tanulás a gépi tanulás egyik legnehezebb problémája, azonban egyes ILP-technikák kellően hatékony mechanizmusnak bizonyultak a megoldására.

A fejezet hátralévő részében az ILP két alapvető megközelítését fogjuk tanulmányozni. Az első a döntési fák általánosításán, a másik a rezoluciós bizonyítás invertálásán alapul.

## Felülről lefelé tanulási módszerek

Az ILP első megközelítésében egy igen általános szabállyal kezdünk és azt fokozatosan leszűkítjük, hogy az adatokra illeszkedjen. Ez az, ami lényegében a döntési fa tanulásánál történik, ahol a döntési fa fokozatosan növekszik, amíg a megfigyelésekkel konzisz-tens nem lesz. Hogy az ILP-t magvalósítsuk, elsőrendű literálokat használunk attribútumok helyett, a hipotézis pedig egy klózhalmaz, a döntési fa helyett. Ebben a részben az egyik legelső ILP-programmal, a FOIL-lal foglalkozunk (Quinlan, 1990).

Tegyük fel, hogy továbbra is a *Nagyszülője(x, y)* predikátum definícióját szeretnénk megtanulni az előbb között családi példák alapján. A döntési fa tanulásához hasonlóan a példákat pozitív és negatív példáakra bontjuk. A pozitív példák:

*⟨György, Anna⟩, ⟨Fülöp, Péter⟩, ⟨Spencer, Harry⟩, ...*

míg a negatív példák:

*⟨György, Erzsébet⟩, ⟨Harry, Zara⟩, ⟨Károly, Fülöp⟩, ...*

Figyeljük meg, hogy mindegyik példa egy objektumpáros, hiszen a *Nagyszülője* egy bináris predikátum. Egészét tekintve a családfában 12 pozitív és 388 negatív (a személyek minden más párosítása) példa található.

A FOIL program klózok egy halmazát alakítja ki, mindegyik *Nagyszülője(x, y)*-nal mint fejjel. A klózoknak a 12 pozitív példát a *Nagyszülője(x, y)* reláció példányosításának kell osztályozniuk, a többi 388-at viszont kizární. A klózok Horn-klózok, melyeket negált literálokkal bővítenek ki, ahol a negálást mint kudarcot használjuk, hasonlóan a Prologhoz. Egy üres testű klózzal kezdünk:

⇒ *Nagyszülője(x, y)*

Mivel ez a választás minden példát pozitívnak sorol be, leszűkítésre szorul. Ez úgy tehető meg, hogy egy lépésben egy literált adunk a bal oldalhoz. A három lehetséges eset:

*Apja(x, y) ⇒ Nagyszülője(x, y)*

*Szülője(x, z) ⇒ Nagyszülője(x, y)*

*Apja(x, z) ⇒ Nagyszülője(x, y)*

(Vegyük észre, hogy a *Szülője*-t definiáló klózzal a háttértudás már rendelkezik.) E három eset közül az első az összes 12 pozitív példát helytelenül negatívnak sorolja be, így ezt az esetet kizártuk. A második és a harmadik az összes pozitív példával összhangban van. A második eset azonban a negatív példák nagyobb halmazán – pontosabban kétszer annyi példán – helytelen, mert az anyákat és az apákat is megengedi. Így a választásunk a harmadik esetre esik.

Most az esetet tovább kell szűkíteni, kizárvva azokat az eseteket, amikor  $x$  valamelyen  $z$  szülője, de  $z$  mégsem  $y$  szülője. Egy egyedi  $Szülője(z, y)$  literál hozzáadása a következő klózt eredményezi:

$$Apja(x, z) \wedge Szülője(z, y) \Rightarrow Nagyszülője(x, y)$$

amely az összes példát helyesen sorolja be. A FoIL képes lesz ezt a literált megtalálni és kiválasztani, és ezzel a tanulási feladatot megoldja. Általánosságban, a helyes megoldás megtalálása előtt, a FoIL-nak számos sikertelen klóz között kell keresnie. Ez a példa a FoIL működésének egyszerű illusztrációja. A teljes algoritmus váza a 19.12. ábrán látható. Az algoritmus lényegében literárról literálra egy klózt konstruál, amíg az meg nem egyezik a pozitív példák valamelyen részhalmazával, és egyik negatív példával sem egyezik. A klóz által lefedett pozitív példákat ekkor a tanító halmazból elhagyjuk, és az eljárást folytatjuk, amíg egyetlen pozitív példa sem marad. Az algoritmus két fő komponense szorul magyarázatra – az ÚJ-LITERÁLOK, amely a klózhoz hozzáadandó összes új literált konstruálja, és a LITERÁL-MEGVÁLASZTÁSA, amely kiválasztja a hozzáadandó literált.

```

function FoIL(példák, cél) returns a Horn-klózok egy halmaza
  inputs: példák, a példák halmaza
    cél, egy literál a célpredikátum részére
  local variables: klózok, a klózok halmaza, kezdetben üres

  while példák tartalmaznak még pozitív példákat do
    klóz  $\leftarrow$  ÚJ-KLÓZ(példák, cél)
    a példák-ból hagyjuk el a klóz lefedte példákat
    a klóz-t adjuk hozzá a klózok-hoz
  return klózok

function ÚJ-KLÓZ(példák, cél) returns egy Horn-klóz
  local variables: klóz, a cél-ra mutató, üres testű klóz
    l, a klózhoz hozzáadandó literál
    kiterjedt-példák, a példák halmaza, az új változóknak adott értékekkel együtt

  kiterjedt_példák  $\leftarrow$  példák
  while a kiterjedt_példák tartalmaznak még negatív példákat do
    l  $\leftarrow$  LITERÁL-MEGVALASZTÁSA(ÚJ-LITERÁLOK(klóz), kiterjedt_példák)
    l hozzáfűzése a klóz testéhez
    kiterjedt_példák  $\leftarrow$  a PÉLDA-KITERJEDÉS-t a kiterjedt_példák minden
      példájára történő alkalmazásával kapott példahalmaz
  return klóz

function PÉLDA-KITERJEDÉS(példa, literál) returns
  if példa kielégíti a literál-t
    then return a példa kiterjesztével kapott példahalmaz úgy, hogy a literál
      minden új változóját minden lehetséges konstans értékre lekötöttük
  else return üres halmaz

```

19.12. ábra. A FoIL algoritmus váza, amely példákból elsőrendű Horn-klózokat tanul meg. Az ÚJ-LITERÁLOK és a LITERÁL-MEGVÁLASZTÁSA eljárásokat a szövegben magyarázzuk meg.

Az ÚJ-LITERÁLOK egy klózt kap bementként, és minden lehetséges „hasznos” literált konstruál, amit a klózhöz hozzá lehetne adni. Vegyük például az alábbi klózt:

$$Apja(x, z) \Rightarrow Nagyszülője(x, y)$$

Háromfajta literált adhatunk hozzá a klózhoz:

- Predikátumokat használó literálok:** a literál lehet negált vagy ponált, minden létező predikátum (a célpredikátumot is beleértve) használható, és minden argumentum csakis változó lehet. A predikátumok argumentumai tetszőleges változók lehetnek, egy kikötéssel: minden literálnak tartalmaznia kell *legalább* egy változót egy korábbi literálból vagy a klóz fejéből. Olyan literálok, mint: *Anyja(z, u)*, *Házas(z, z)*,  $\neg$ *Férfi(y)* és *Nagyszülője(v, x)* megengedettek, ám *Házas(u, v)* nem. Megjegyzendő, hogy a klóz fejéből származó predikátumok használata lehetővé teszi, hogy a FoIL a rekurzív definíciót is megtanulja.
- Egyenlőségi és egyenlőtlenségi literálok:** ezek a klózban már szereplő változókat kötik össze. Így például a klózhöz hozzáadhatjuk, hogy:  $z \neq x$ . Ezekben a literálokban a felhasználó által specifikált konstansok is felhasználhatók. Az aritmetikát tanulva célszerű kitüntetni a 0-t és az 1-et, a listafüggvényeknél pedig az üres [] listát.
- Aritmetikai összehasonlítások:** folytonos változójú függvények tanulásánál célszerű lehet az olyan literálok hozzáadása, mint az  $x > y$  és az  $y \leq z$ . A döntési fa tanulásához hasonlóan egy konstans küszöbértéket is lehet használni, ami a teszt diszkriminációs erejét maximalizálja.

Mindez a keresési téren nagyon nagy elágazási tényezőhöz vezet (lásd 19.6. feladat), azonban a FoIL implementációiban a típusinformációt is fel lehet használni a hipotézistér leszűkítésére. Az embereket és a számokat tartalmazó tárgytartományban például a típusleszűkítés nem engedné, hogy az ÚJ-LITERÁLOK olyan literálokat generáljon, mint a *Szülője(x, n)*, ahol  $x$  egy ember,  $n$  viszont egy szám.

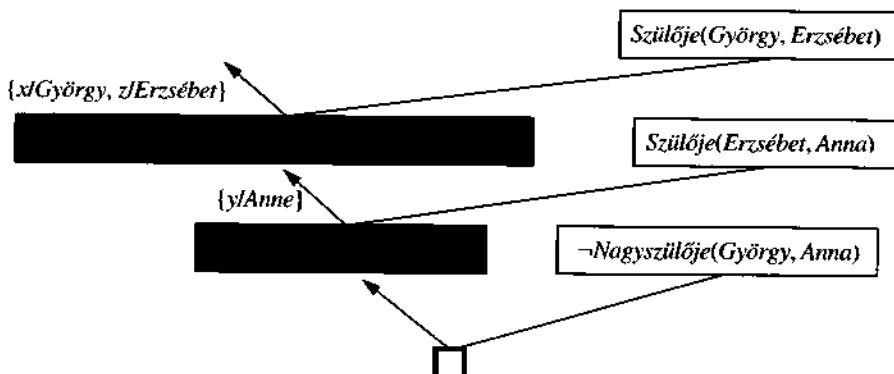
A LITERÁL-MEGVÁLASZTÁSA az információnyereségre (lásd 761. oldal) valamelyen mértékben hasonló heurisztikát használ, hogy elődontse, melyik literált adja hozzá a klózhöz. A részletek itt nem különösen fontosak, ráadásul számos kipróbált változat is létezik. A FoIL egyik érdekes járulékos tulajdonsága az Ockham borotvája elvének az alkalmazása bizonyos hipotézisek eliminálására. Ha (egy bizonyos mérték szerint) egy klóz hosszabbnak mutatkozik, mint az általa megmagyarázott pozitív példák összhossza, akkor ezt a klózi a továbbiakban nem tekintjük lehetséges hipotézisnek. Ezzel a technikával elkerülhetjük az adatokban található zajra illeszkedő, túlságosan bonyolult klózok származtatását. A zaj és a klózhossz kapcsolatának magyarázatát lásd a 821. oldalon.

A FoIL-t és társait számos definíció megtanulására vetették be. Az egyik leghatárosabb demonstráció (Quinlan és Cameron-Jones, 1993) a listakezelő függvényekre vonatkozó feladatok hosszú sorának a megoldása volt Bratko Prolog-tankönyvéből (Bratko, 1986). A program mindegyik esetben képes volt a függvény helyes definícióját kisszámú példa alapján megtanulni, miközben háttértudásként használta a korábban megoldott függvényeket.

## Induktív tanulás inverz rezolúcióval

Az ILP-nek másik fő megközelítése a megszokott deduktív bizonyítási folyamat invertálása. Az **inverz rezolúció** (*inverse resolution*) azon az észrevételek alapul, hogy ha a példa *Besorolások a Háttértudás*  $\wedge$  *Hipotézis*  $\wedge$  *Leírások*-ból következnek, akkor ezt a tényt rezolúcióval be kell tudnunk bizonyítani (hiszen a rezolúció teljes). Ha képesek volnának a bizonyítást „visszafelé pörgetni”, akkor tudnának egy olyan *Hipotézis*-t találni, amire a bizonyítás sikeres lesz. A kultusprobléma tehát, hogy hogyan tudnának a rezolúciós eljárást invertálni, hogy a bizonyítás visszafelé fussen le.

Az inverz rezolúció visszafelé haladó bizonyítási eljárását mutatjuk be, amely egyedi visszafelé mutató lépésekkel áll. Egy közönséges rezolúciós lépés a  $C_1$  és  $C_2$  klózból indul ki, és azokat rezolválja egy  $C$  **rezolvens** létrehozva eredményül. Az inverz rezoluciós lépés a  $C$  rezolvens alapján két klózt,  $C_1$ -et és  $C_2$ -t hoz létre úgy, hogy  $C$  a  $C_1$  és  $C_2$  rezolválásának az eredménye, vagy pedig  $C$  és  $C_1$  alapján állít elő egy lehetséges  $C_2$ -t.



19.13. ábra. Az inverz rezoluciós eljárás kezdő lépései. Az árnyékolt klózokat az inverz rezoluciós eljárás generálja a jobb oldalon és az alatta lévő klózokból. A nem árnyékolt klózok a Leírások-ból és a Besorolások-ból származnak.

A 19.13. ábra inverz rezoluciós eljárás kezdő lépéseit mutatja, ahol a figyelem középpontjában a *Nagyszülöje(György, Anna)* pozitív példa áll. Az eljárás a bizonyítás végén kezd (amit az ábra alján ábrázolunk). A  $C$  rezolvensnek üres klózt (azaz egy ellentmondást) választunk,  $C_2$ -nek pedig a  $\neg$ *Nagyszülöje(György, Anna)*-t, amely a célpélda negálása. Az első inverz lépés  $C$ -ból és  $C_2$ -ból  $C_1$ -nek a *Nagyszülöje(György, Anna)*-t generálja. A következő lépés e klózt  $C$ -nek veszi, a *Szülöje(Erzsébet, Anna)* klózt pedig  $C_2$ -nek, és az alábbi  $C_1$  klózt generálja:

$$\neg \text{Szülöje}(\text{Erzsébet}, y) \vee \text{Nagyszülöje}(\text{György}, y)$$

Az utolsó lépés ezt a klózt rezolvensként kezeli. A *Szülöje(György, Erzsébet)* klózt  $C_2$ -nek véve, egy lehetséges  $C_1$  klóz az alábbi hipotézis lehet:

$$\text{Szülöje}(x, z) \wedge \text{Szülöje}(z, y) \Rightarrow \text{Nagyszülöje}(x, y)$$

Egy rezolúciós bizonyításunk van tehát arra, hogy a hipotézis, a leírások és a háttér-tudás maga után vonzza a *Nagyszülője(György, Anna)* osztályozást.

Világos, hogy az inverz rezolúció keresést tartalmaz. minden egyes inverz rezolúciós lépés nemdeterminisztikus, hiszen az adott  $C$  és  $C_1$ -hez több, sőt végtelen sok olyan  $C_2$  klóz lehet, amely kielégíti azt a feltételt, miszerint  $C_1$ -gyel rezolválva  $C$ -t ad eredményül. Így a 19.13. ábra utolsó lépéseiben a  $\neg\text{Szülője(Erzsébet, y)} \vee \text{Nagyszülője(György, y)}$  megválasztása helyett az inverz rezolúciós lépés a következő állításokat is tudná generálni:

- $\neg\text{Szülője(Erzsébet, Anna)} \vee \text{Nagyszülője(György, Anna)}$
- $\neg\text{Szülője(z, Anna)} \vee \text{Nagyszülője(György, Anna)}$
- $\neg\text{Szülője(z, y)} \vee \text{Nagyszülője(György, y)}$

(Lásd a 19.4. és a 19.5. feladatot.) Az egyes lépésekben részt vevő klózokat meg lehet választani a *Háttértudás*-ból, a példa *Leírások*-ból, a negált *Besorolások*-ból vagy az inverz rezolúciós fában eddig már legenerált hipotézisklózokból. A nagyszámú lehetőség, egyéb kontroll nélkül, nagy elágazási tényezőt (és így kevessé hatékony keresést) jelent. A keresés kézben tartására több megközelítést probáltak ki az implementált ILP-rendszerekben:

1. A redundáns választások eliminálhatók – például a legspecifikusabb lehetséges hipotézisek generálásával, valamint megkövetelve, hogy a hipotézisklózok minden konzisztensek legyenek egymással és a megfigyelésekkel. Az utolsó kritérium kizárná az előbbieken megadott  $\neg\text{Szülője(z, y)} \vee \text{Nagyszülője(György, y)}$  klózt.
2. A bizonyítási stratégia szintén megszorítható. A 9. fejezetben láttuk például, hogy a **lineáris rezolúció (linear resolution)** egy teljes, megszorított stratégia, amely lehetővé teszi, hogy a bizonyítási fáknak csak lineáris elágazási struktúrája lehessen (mint a 19.13. ábrán).
3. A reprezentációs nyelv is megszorítható, a függvényszimbólumok eliminálásával és csak Horn-klózokat megengedve. A PROGOL például Horn-klózokkal dolgozik, **inverz vonzatrelációt (inverse entailment)** alkalmazva. Az ötlet a

$\text{Háttértudás} \wedge \text{Hipotézis} \wedge \text{Leírások} \models \text{Besorolások}$

vonzatkényszer megváltoztatása a

$\text{Háttértudás} \wedge \text{Leírások} \wedge \neg\text{Besorolások} \models \neg\text{Hipotézisek}$

logikailag ekvivalens formára.

Ebből a **Hipotézisek** származtatására már a Prolog-beli normál Horn-klóz dedukcióra hasonlító módszert használhatjuk, negálással mint kudarccal. Mivel ez Horn-klózokra korlátozott, így nem teljes módszer, azonban a teljes rezolúciónál hatékonyabb lehet. Inverz vonzatrelációval lehetséges a teljes következtetés alkalmazása is (Inoue, 2001).

4. A következtetést modellellenőrzéssel is megvalósíthatjuk a tételezbizonyítás helyett. A PROGOL rendszer (Muggleton, 1995) a modellellenőrzés egy formáját használja kereséskorlátozás céljára. Azaz, hasonlóan a válaszhalmaz-programozáshoz, a logikai változók lehetséges értékeit generálja, és a konziszenciát ellenőrzi.
5. A következtetést az alap ítéletklózokkal is megtehetjük az elsőrendű logika helyett. A LINUS-rendszerben (Lavrač és Džeroski, 1994) az elsőrendű elméleteket ítéletlogi-

kára fordítják le, ítéletlogikai tanuló rendszerrel oldják meg azokat, majd visszafordítják. Egyes problémák esetén ítéletlogikai állításokkal sokkal hatékonyabban lehet dolgozni, ahogy ezt a 11. fejezetben a SATPLAN esetében már láttuk.

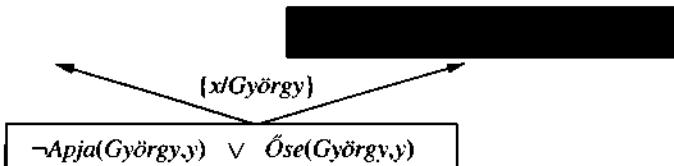
## Felfedezés induktív logikai programozással

Egy teljes rezolúciós stratégiát invertáló inverz rezolúciós eljárás elvben az elsőrendű elméletek tanulásának egy teljes algoritmusa. Ez azt jelenti, hogy ha valamely ismeretlen *Hipotézis* egy sor példát generál, akkor az inverz rezolúciós eljárás a példákból tudja generálni a *Hipotézis-t*. Ez a megfigyelés egy érdekes lehetőséget is sugall. Tegyük fel, hogy a rendelkezésre álló példák szabadon eső testek által leírt trajektóriák egy halmazát tartalmazzák. Lehetséges, hogy egy inverz rezolúciós program elvben képes a gravitáció törvényét megtanulni? A válasz természetesen igen, mert a gravitáció törvénye, megfelelő matematikai háttér mellett, lehetővé teszi a példák magyarázatát. Hasonlóképpen el lehet képzelni, hogy az elektromágneses hullámok elmélete, a kvantummechanika és a relativitáselmélet, mind beletartoznak az ILP-programok hatáskörébe. Persze a gépíró majom hatáskörébe is beletartoznak; jobb heurisztikákra és a keresési teret jobban strukturáló technikákra még minden szükség van.

Egy dolgot azért az inverz rezolúciós rendszer meg fog tenni a részünkre: új predikátumokat fog kitalálni. Ez a képessége kissé mágikusnak tűnik, hiszen a számítógépre szokás úgy tekinteni, mint ami „csak azzal dolgozik, amit bemenetként kapott”. Tény, hogy az új predikátumok egyenesen az inverz rezolúciós lépésből pottyanak ki. A legegyszerűbb az az eset, amikor egy adott  $C$  klóz esetén két új klóz,  $C_1$  és  $C_2$  létezését kellene felvetni. A  $C_1$  és  $C_2$  rezolúciója a két klóz által közösen tartalmazott literált eliminálta, lehetséges hát, hogy az eliminált literálban olyan predikátum is szerepelt, amely  $C$ -ben már nem lép fel. Visszafelé haladva tehát lehetőséggé válik egy új predikátum létrehozása, amiből viszont a hiányzó literál már rekonstruálható.

A 19.14. ábra arra mutat példát, ahogy az *Őse* definíciójának tanulása közben egy új  $P$  predikátumot generálunk. Ha már előállítottuk a  $P$ -t, akkor ez használható az inverz rezolúció későbbi lépéseiiben is. Egy későbbi lépés például feltételezheti, hogy az  $\text{Anyja}(x, y) \Rightarrow P(x, y)$ . Az új predikátum jelentését így a predikátumot felhasználó hipotézis generálása korlátozza. Egy másik példa elvezethet az  $\text{Apja}(x, y) \Rightarrow P(x, y)$ -ig. Más szóval a  $P$  predikátum olyan valami, ami minket általában a *Szülője* relációra emlékeztet. Korábban említettük, hogy új predikátumok bevezetése a célpredikátum definíciójának a méretét lényegesen csökkentheti. Az új predikátumok kialakításának beépített képességével, az inverz rezolúciós rendszer sokszor olyan tanulási problémákkal is megbirkózik, amelyekre más módszerek alkalmatlanok.

A tudomány legmélyebbre ható forradalmai új predikátumok és függvények bevezetésével függnek össze, ilyen például a Galilei felfedezte gyorsulás vagy Joule termikus energiája. Ha ezek a fogalmak már adottak, az új törvények felfedezése (viszonylag) egyszerű. A fő nehézség arra rájönni, hogy egy új entitás bevezetése, amely a létező entitásokkal meghatározott konkrét kapcsolatban van, lehetővé teszi a megfigyelések egész halmazának a korábban lehetségesnél sokkal egyszerűbb és elegánsabb elmélet keretein belül történő magyarázatát.



19.14. ábra. Egy új  $P$  predikátumot generáló inverz rezolúciós lépés

Az ILP-rendszerek egyelőre Galilei- vagy Joule-szintű felfedezéseket nem tettek, a felfedezéseit azonban tudományos irodalomban való publikálásra érdemesnek találták. Így például *Journal of Molecular Biology*-ban Turcotte írja le a fehérje összehajtatózási szabályok automatikus felfedezését a PROGOL ILP programmal (Turcotte és társai, 2001). A PROGOL által felfedezett szabályok közül sokat az ismert elvekből ugyan le lehetett volna következtetni, a többséget azonban a standard biológiai adatbázisok részeként korábban mégsem publikálták (lásd 19.10. ábra példájá). Egy ehhez kapcsolódó kutatás részeként Srinivasan (Srinivasan és társai, 1994) molekula-struktúraalapú szabályok felfedezésével foglalkozott nitroaromatikus komponensek mutagenicitására. Ilyen komponensek a gépkocsik által kibocsátott kipufogógázban találhatók. A standard adatbázisokban lévő komponensek 80%-ában lehetséges a négy fontos jellemzőt azonosítani, és az erre épülő lineáris regresszió jobb az ILP-nél. A maradék 20% esetén a jellemzők önmagukban nem elegendők az előrejelzéshez, az ILP relációkat azonosít, amelyekkel a lineáris regresszió, a neurális hálók és a döntési fák képességein túlteresz. King (King és társai, 1992) azt mutatta meg, hogy a különböző gyógyszerek gyógykezelési hatását hogyan lehetne előre jelezni. Ezen esetek mindegyikében úgy tűnik, hogy az ILP nagy hatékonyságához a relációk reprezentálási képessége és a háttérrelatív használata járulnak hozzá. Az a tény, hogy az ILP által megtalált szabályokat az emberek is képesek interpretálni, inkább segíti ezeknek a technológiáknak az elfogadását a biológiai folyóiratokban, mint a számítógépes tudománnyal foglalkozó folyóiratokban.

Az ILP a biológián túl más tudományokhoz is hozzájárult. A legfontosabbak egyike a természetes nyelvfeldolgozás, ahol az ILP-t komplex relációs információ szövegekből való kinyerésére alkalmazták. Ezeket az eredményeket a 23. fejezet foglalja össze.

## 19.6. ÖSSZEFOGLALÁS

Ebben a fejezetben különböző módszereket vizsgáltunk meg arra vonatkozóan, hogy az előzetes tudás hogyan segítheti az ágenst abban, hogy új tapasztalatokból tanuljon. Mivel az előzetes tudás zöme inkább a relációs modellekkel, és nem attribútumalapú modellekkel van megadva, a relációs modellekben tanuló rendszerekkel is foglalkozunk. Az alábbiakban összefoglaljuk a fontos tudnivalókat.

- Az előzetes tudás felhasználása a tanulásnál a **kumulatív tanuláshoz** (**cumulative learning**) vezet, ahol az ágens az új tudás beszerzésével javítja a tanulási képességét.
- Az előzetes tudás segít a tanulásban, mert a különben konzisztns hipotéziseket eliminálja, és a példák magyarázatának „kitöltésével” rövidebb hipotézisekhez vezet. Ezek a tulajdonságok javítják a tanulás minta- és számítási komplexitását.

- Az előzetes tudás által betöltött különböző – **vonzatkényszerek** (*entailment constraints*) formájában kifejezett – logikai szerepeknek a megértése segít a különféle tanulási módszerek definiálásában.
- A **magyarázatalapú tanulás** (**MAT**) (*explanation-based learning, EBL*) az egyedi példákból úgy emeli ki az általános szabályokat, hogy a példákat először megmagyarázza, majd a magyarázatot általánosítja. Ezzel a deduktív módszerrel az elsődleges tudást hasznos, hatékony, speciális rendeltetésű szaktudássá változtatjuk.
- A **relevanciaalapú tanulás** (**RAT**) (*relevance-based learning, RBL*) az előzetes tudást meghatározások formájában használja a releváns attribútumok azonosítására. Ily módon egy redukált hipotézisteret generál, és a tanulási folyamatot meggyorsítja. A RAT lehetővé teszi az egyedi példák deduktív általánosítását is.
- A **tudásalapú induktív tanulás** (**TIT**) (*knowledge-based inductive learning, KBIL*) induktív hipotéziseket keres, amelyek a háttérudás segítségével megmagyarázzák a megfigyelések halmazát.
- Az **induktív logikai programozási** (**ILP**) (*inductive logic programming*) technikák a TIT-hez folyamodnak, az elsőrendű logikában kifejezett háttérudást felhasználva. Az ILP-módszerek olyan relációs tudás megtanulására is alkalmasak, amelyet az attribútumalapú rendszerekben nem lehet kifejezni.
- Az ILP felülről lefelé megközelítéssel, a nagyon általános szabályok finomításával, vagy pedig lentről felfelé megközelítéssel, a deduktív folyamat invertálásával valósítható meg.
- Az ILP-módszerek természetes módon generálnak új predikátumokat is, amelyekkel új elmeletek tömören kifejezhetők, és általános célú tudományos elmeletformáló rendszerekként is igényesek mutatkoznak.

## Irodalmi és történeti megjegyzések

Bár az előzetes tudás felhasználása a tanulásban természetes témnak tűnhet a tudományfilozofusok számára, mostanáig meglepően kevés formális eredmény született. Nelson Goodman filozófus a *Fact, Fiction, and Forecast* c. művében megcáfolta azt a korábbi feltételezést, miszerint az indukció nem más, mint egy univerzálisan kvantifikált kijelentés megfelelően sok példájának a megszemlélése és hipotézisként való elfogadása (Goodman, 1954).

Vegyük például azt a hipotézist, hogy „Minden smaragd zölék”, ahol a **zölék** azt jelenti, hogy „zöld, ha  $t$  idő előtt figyeljük meg, utána azonban kék”. A  $t$  idő előtt bármikor példák millióit láthatnánk, amelyek mind alátámasztják, hogy a smaragdok zölékek, negatív példák nélkül, mégis hezitálnának a szabályt elfogadni. A jelenség csak az indukciós folyamat szempontjából releváns a priori tudás szerepével magyarázható. Goodman a hasznosítható a priori tudás teljes választékát javasolja, a meghatározások **túlzott hipotézisnek** (*overhypothesis*) nevezett változatát is beleértve. Sajnos a gépi tanulással foglalkozó korai kutatásnál Goodman munkájára nem figyeltek fel.

A MAT gyökerei a tervkészítésnél használt STRIPS módszereihez nyúlnak vissza (Fikes és társai, 1972). Egy terv elkészítésével a terv egy általánosított változatát a tervező könyvtárában tárolták, és később makróoperátorként (*macro-operator*) a tervkészítésnél felhasználták. Hasonló ötletek Anderson ACT\* architektúrájában is megjelentek,

**tudáskompilálás (knowledge compilation)** címen (Anderson, 1983) és SOAR architektúrájában **tudásdarabolás (chunking)** címen (Laird és társai, 1986). A sémabeszerzés (**schema acquisition**) (DeJong, 1981), az **analitikus általánosítás (analytical generalization)** (Mitchell, 1982) és a **kényszerlapú általánosítás (constraint-based generalization)** (Minton, 1984) a MAT tanulás irányában jelentkező és a (Mitchell és társai, 1986; DeJong és Mooney, 1986) publikációk stimulálta gyorsan növekvő érdeklődésnek közvetlen előfutárai voltak. A szövegben bemutatott MAT algoritmust Hirsh (Hirsh, 1987) vezette be, aki azt is megmutatta, hogy az algoritmust hogyan kell egy logikai programozási rendszerbe ágyazni. Van Harmelen és Bundy a MAT-ot a programelemző rendszerekben (Jones és társai, 1993) használt **részleges kiértékelő (partial evaluation)** módszer variánsaként magyarázzák (Van Harmelen és Bundy, 1988).

Az utóbbi időben a szigorú elemzés és a kísérleti kutatás a MAT jobb megértéséhez vezetett a problémamegoldás sebességében mért lehetséges költségeinek és előnyeinek terén. Minton azt mutatta meg, hogy hacsak tekintélyes munkát nem fektetünk bele, a MAT a programot lényegesen lelassíthatja (Minton, 1988). Tambe hasonló problémákkal küszködött a tudásdarabolásnál, és a szabálynyelv kifejezőjének a mérsékését javasolta, hogy a szabályok munkamemoriához való illesztésének jelentős költségét minimálizáljuk (Tambe és társai, 1990). Erős a párhuzam ezen kutatás és az elsőrendű logika leszűkített változatának következetettsé komplexitására vonatkozó jelenlegi eredményei között (lásd 9. fejezet). A MAT várható nyereségének formális valószínűségi elemzése a (Greiner, 1989; Subramanian és Feldman, 1990) munkákban található. Egy kiváló áttekintés Dietterichtől származik (Dietterich, 1990).

Ahelyett hogy a példákat mint az általánosítás fókuszpontjait tekintenénk, új problémák megoldására közvetlenül felhasználhatók az **analógiás következtetés (analogical reasoning)** elnevezésű megközelítésben. Az ilyen következtetésnek több változata is létezik. Idetartozik a hasonlóság mértékén alapuló plauzsibilis következtetés (Gentner, 1983) és a meghatározásokon alapuló deduktív következtetés (Davies és Russell, 1987), amely azonban megkívánja a példa jelenlétéit ahhoz, hogy a belőle kialakított „lusta” MAT a régi példa általánosítását az új probléma szükségletei szerint irányitsa. Az analógiás következtetésnek ezt az utóbbi formáját általában meg lehet találni az **eset-alapú következtetésnél (case-based reasoning)** (Kolodner, 1993) és a **származtatási analógiánál (derivational analogy)** (Veloso és Carbonell, 1993).

A releváns információval funkcionális függőségek formájában először az adatbázis-kutatásnál foglalkoztak, ahol ez a nagy attribútumhalmazok kezelhető részhalmazokra való strukturálásának eszköze volt. Funkcionális függőségeket analógiás következtetésre Carbonell és Collins, majd logikai színezettel Bobrow és Raphael használtak (Carbonell és Collins, 1973; Bobrow és Raphael, 1974). Davies és Russell (Davies, 1985; Davies és Russell, 1987) a függőségeket másuktól függetlenül újra felfedezték, és az analógiás következtetés szempontjából teljes körű logikai elemzésnek vetették alá. A függőségeket deklaratív elfogultság céljából Russell és Grosof használták (Russell és Grosof, 1987). A meghatározások és a leszűkített szótárral rendelkező hipotézistér köztöti ekvivalenciát Russell (Russell, 1988) bizonyította be. A meghatározásokat tanuló algoritmust és az RADFT megnövelt hatékonyságát először a FOCUS algoritmusban mutatták be (Almuallim és Dietterich, 1991). Tadepalli a meghatározások tanulására egy ügyes algoritmust közölt, amely a tanulási sebességen igen lényeges javulást mutat (Tadepalli, 1993).

Az a gondolat, hogy az induktív tanulás kivitelezhető inverz dedukcióval W. S. Jevonsig vezethető vissza (Jevons, 1874), aki azt írta, hogy: „Mind a formális logika, mind a valóságosság-elmélet tanulmányozása arra az álláspontra juttatott engem, hogy egy olyan dolog, mint egy külön indukciós módszer, a dedukcióval szembeállítva nincs, és az indukció egyszerűen a dedukció egy fordított alkalmazása.” A számítástudományi elemzések Gordon Plotkin figyelemre méltó PhD-disszertációjával kezdődtek Edinburghban (Plotkin, 1971). Bár Plotkin számos, a mai ILP területén használt tételeit és módszert dolgozott ki, az indukció egyes részproblémáira vonatkozó bizonyos eldönthetetlenségi eredmények elvették a kedvét. A MIS (Shapiro, 1981) újból bevezette a logikai programok tanulási problémáját, azonban ezt főleg az automatikus hibakeresés elméletéhez való hozzájárulásának tekintették. A szabályindukció kutatása, vagyis az olyan rendszerek, mint az ID3 (Quinlan, 1986) és CN2 (Clark és Niblett, 1989) a FOIL-hoz vezettek (Quinlan, 1990), amely első ízben tette lehetővé relációs szabályok gyakorlati indukcióját. A területet Muggleton és Buntine pezsdítették fel (Muggleton és Buntine, 1988). CIGOL programjuk az inverz rezolúció nem egészben teljes változatát tartalmazta, és új predikátumok generálására is alkalmas volt.<sup>5</sup> A predikátum felfedezésével Wirth és O'Rorke is foglalkoztak (Wirth és O'Rorke, 1991). A következő nagyobb rendszer a GOLEM volt (Muggleton és Feng, 1990), amely a Plotkin-féle relatíve legkevésbé általános általánosítás ötletén alapuló lefedő algoritmust használja. A FOIL-fentről lefelé módon, míg a CIGOL és a GOLEM lentről felfelé módon dolgoztak. E korszak más rendszerei az ITOU (Rouveiro és Puget, 1989) és a CLINT (De Raedt, 1992) rendszerek voltak. Újabban a PROGOL (Muggleton, 1995) az inverz vonzatreláció egy hibrid (fentről lefelé és lentről felfelé) megközelítését valósította meg, és a rendszert számos gyakorlati problémára alkalmazták, különösképpen a biológiában és a természetes nyelvfeldolgozásban. Muggleton a PROGOL egy kiterjesztését írja le (Muggleton, 2000), sztochasztikus logikai programok alakjában reprezentált bizonytalanság kezelésére.

Az ILP-módszerek formális elemzését (Muggleton, 1991) tartalmazza, míg a (Muggleton, 1992) egy nagy cikkgyűjtemény. A módszerek és az alkalmazások nagy gyűjteménye a (Lavrac és Džeroski, 1994) könyv. A terület történetéről és a jövő kilátásairól újabb áttekintést ad (Page és Srinivasan, 2002). Haussler korai komplexitási eredményei azt sugallták, hogy az elsőrendű állítások tanulása reménytelenül bonyolult (Haussler, 1989). A klózokra vonatkozó különböző szintaktikai korlátozások jobb megértésével azonban pozitív eredmények is megjelentek, még a rekurziót tartalmazó klózokra is (Džeroski és társai, 1992). Az ILP komplexitási eredményeiről Kietz és Džeroski, valamint Cohen és Page írtak átfogó cikkeket (Kietz és Džeroski, 1994; Cohen és Page, 1995).

Bár az ILP jelenleg a konstruktív indukció problémájának domináló megközelítése, nem csak ezzel próbálkoztak. Az ún. **felfedező rendszerek** (*discovery systems*) célja az új koncepciók tudományos felfedezésének általában a koncepció definícióterében történő közvetlen keresés útján történő modellezése. Doug Lenat AM (Automated Mathematician) programja (Davis és Lenat, 1982) szakértő szabályok formájában megfogalmazott felfedező heurisztikákat használt arra, hogy elemi származékok koncepciók és sejtések keresését irányítsa. Éles ellentében a matematikai következetettsére kifejlesztett rendszerekkel, az AM nélkülözte a bizonyítás fogalmát, és csak sejtés szin-

<sup>5</sup> Az inverz rezolúciós módszer (Russell, 1986)-ben is megjelenik, a teljes algoritmust egy látjegyzetben közlik.

ten tudott működni. A program újra felfedezte a Goldbach-sejtést és az egyértelmű prímtényezőkre bontást. Az AM architektúráját később az EURISKO rendszerben (Lenat, 1983) általánosították, a rendszert a saját felfedező heurisztikákat átírni képes szabályokkal bővítte. Az EURISKO-t más, a matematikai felfedezéstől eltérő tartományokban is alkalmazták, de az AM-nél kisebb sikерrel. Az AM és az EURISKO módszertana vitatt, lásd (Ritchie és Hanna, 1984; Lenat és Brown, 1984).

A felfedező rendszerek egy másik csoportja a valós fizikai adatokkal való munkát célozza és új törvények felfedezését kísérli meg. A DALTON, a GLAUBER és a STAHL (Langley és társai, 1987) olyan szabályalapú rendszerek, amelyek fizikai rendszerekből származó adatokban kvantitatív összefüggéseket keresnek. Mindegyik rendszer képes volt egy tudomány történetéből jól ismert felfedezés újból meghozatalára. A probabilisztikus technikákon – különösképpen az új kategóriákat felfedező klaszterező algoritmusokon – alapuló felfedező rendszerekkel a 20. fejezetben foglalkozunk.

## Feladatok

- 19.1.** Mutassa ki a konjunktív normál formába való átalakításával és a rezolúció alkalmazásával, hogy a 799. oldalon található és a brazílokra vonatkozó konklúzió helyes.
- 19.2.** Az alábbi meghatározás mindegyikére írja fel a meghatározás logikai reprezentációját, és magyarázza meg, hogy a meghatározás miért igaz (ha igaz egyáltalán).
- A postai kód meghatározza a megyét.
  - A külső kialakítás és a névleges érték meghatározzák az érme tömegét.
  - Adott program esetén a bemenetek meghatározzák a kimeneteket.
  - A klíma, a táplálkozás, a fizikai gyakorlat és a metabolizmus meghatározzák, hogy valaki fogyni vagy súlyban gyarapodni fog.
  - A kopaszságot (vagy annak hiányát) az anyai ágon az egyik nagyszülőnek a kopaszsága határozza meg.
- 19.3.** Hasznos lenne a meghatározásoknak egy valószínűségi verziója? Javasoljon egy definíciót.
- 19.4.** Pótolja a  $C_1$  és/vagy  $C_2$  klózok értékét az alábbi klózhalmazban, figyelembe véve, hogy  $C$  a  $C_1$  és a  $C_2$  rezolvense:
- $C = \text{Igaz} \Rightarrow P(A, B), C_1 = P(x, y) \Rightarrow Q(x, y), C_2 = ??$
  - $C = \text{Igaz} \Rightarrow P(A, B), C_1 = ??, C_2 = ??$
  - $C = P(x, y) \Rightarrow P(x, f(y)), C_1 = ??, C_2 = ??$
- Ha több megoldás is lehetséges, a különböző eseteket külön példákkal illusztrálja.
- 19.5.** Tegyük fel, hogy egy rezolúciós lépést végrehajtó logikai programot írunk. Legyen tehát a  $\text{Rezolválás}(c_1, c_2, c)$  sikeres, ha a  $c$  a  $c_1$  és a  $c_2$  rezolválásának az eredménye. Rendes körlémények között a  $\text{Rezolválás}-t$  a tételezbizonyítás részeként fogjuk használni úgy, hogy meghívjuk a  $c$  rezolvens generálására, miközben a  $c_1$  és a  $c_2$  változóba konkrét klózáértékeket helyettesítünk be. Tegyük most

fel, hogy az eljárást a behelyettesített  $c$ -vel és a szabad  $c_1$ -gyel, valamint  $c_2$ -vel hívjuk meg. Megszületik-e az inverz rezolúciós lépés helyes eredménye? Sziűk-séges-e a logikai program valamelyen speciális átalakítása, hogy az inverz lépés helyesen működjön?

- 19.6.** Tegyük fel, hogy a FoIL program egy klózhoz egy literált úgy add hozzá, hogy egy bináris  $P$  predikátumot használ, és hogy az előbbi literálok (a klöz fejrészét beleérte) öt különböző változót tartalmaznak.
- Hány funkcionálisan különböző literált lehet generálni? A két literál funkcionálisan egybeesik, ha csupán az általuk tartalmazott új változók neveiben különböznek.
  - Meg tudna adni általános képletet a különböző,  $r$  argumentummal rendelkező predikátumot használó literálok számának meghatározására, ha előzőleg  $n$  változót használtunk?
  - A FoIL miért nem enged olyan literálokat használni, amelyek az előzőleg használt változókat nem tartalmazzák?
- 19.7.** A 19.11. ábrán látható családfa adatainak vagy annak részhalmazának felhasználásával alkalmazza az Őse predikátum megtanulására a FoIL algoritmust.

# 20. STATISZTIKAI TANULÁSI MÓDSZEREK

*Ebben a fejezetben megfigyelések alapján történő bizonytalan következetetésnek tekintjük a tanulást.*

Az V. részben rámutattunk, hogy a valós életre jellemző környezetben gyakran előfordul, hogy a tudás bizonytalan. Az ágensek a bizonytalanságot valószínűség- és döntés-elméleti módszerekkel tudják kezelni, de ehhez először tapasztalataik alapján fel kell állítaniuk a világra vonatkozó valószínűségi modelljüket. Ez a fejezet bemutatja, hogy milyen módon tudják ezt megtenni. Látni fogjuk, hogyan kell úgy formalizálni a tanulási feladatot, mint egy valószínűségi következetetési folyamatot (lásd 20.1. alfejezet). Bemutatjuk, hogy a tanulás bayesi megközelítése rendkívül hatékony: általános megoldást ad a zaj, a túllilleszkedés és az optimális predikció problémáira. Továbbá figyelembe veszi azt a tényt, hogy a nem-egészen-mindentudó ágens soha nem tudhatja biztosan, hogy a világról alkotott melyik elmélet helyes, mégis döntéseket kell hoznia.

A 20.2. és 20.3. alfejezetben valószínűségi modellek – elsősorban a Bayes-hálók – tanulási módszereit ismertetjük. A 20.4. alfejezet olyan módszereket tárgyal, amelyek speciális minták tárolását és előhívását végzik. A 20.5. alfejezet a **neurális háló (neural network)** tanítással foglalkozik, míg a 20.6. alfejezet a **kernelgépekkel (kernel machine)**. Ennek a fejezetnek egy része erősebben matematikai megközelítésű (feltételezi a többváltozós analízis alapjainak ismeretét), bár az általános tanulságok megérthetők anélkül is, hogy elmerülnénk a matematikai részletekben. Hasznos lehet, ha az olvasó ezen a ponton először átnézi a 13. és a 14. fejezeteket, valamint az A) függeléket.

## 20.1. STATISZTIKAI TANULÁS

Az ebben a fejezetben használt alapvető koncepciók ismét az **adat (data)** és a **hipotézis (hypothesis)**, éppúgy, mint ahogyan a 18. fejezetben volt. Itt az adatok a **tények (evidence)** – ezek a területet leíró valószínűségi változók egy részének vagy mindegyikének egy konkrét megvalósulását jelentik. A hipotézis valamilyen valószínűségi elmélet arról, hogy a világ adott területe hogyan is működik, ennek speciális esetei a logikai elmélettel leírt területek.

Vegyük egy *nagyon* egyszerű példát. Kedvenc „melegétés” cukorkánk kétféle, meggy- (nyam-nyam) és citrom- (brrrr) ízben kapható. A cukorkagyártónak sajátos humora van, és mindegyik cukorkát – ízétől függetlenül – ugyanolyan átlátszatlan papírba csomagolja. Az édességet nagyon nagy zsákokban árulják, amelyekből ötféle van – kívülről megint csak megkülönböztethetetlenek egymástól:

- $h_1$ : 100% meggy  
 $h_2$ : 75% meggy + 25% citrom  
 $h_3$ : 50% meggy + 50% citrom  
 $h_4$ : 25% meggy + 75% citrom  
 $h_5$ : 100% citrom

Egy adott új zsák cukorka esetén a  $H$  (*hipotézist jelölő*) véletlen változó a zsák típusát jelenti, lehetséges értékei  $h_1$ -től  $h_5$ -ig terjednek. Természetesen  $H$  közvetlenül nem figyelhető meg. Ahogy a cukorkákat felbontjuk és megvizsgáljuk, adatokat gyűjtünk – amelyeket  $D_1, D_2, \dots, D_N$  jelöl. Mindegyik  $D_i$  egy olyan véletlen változó, amelynek lehetséges értékei a *meggy*, illetve a *citrom*. Az ágens feladata, hogy jóslást adjon a következő cukorka ízére.<sup>1</sup> Ez a nyilvánvalóan triviális feladat érdekes módon mégis jó betekintést nyújt a legfontosabb problémák közül többre is. Az ágensnek valóban elméletet kell alkotnia a vilagról, bár csak nagyon egyszerűt.

A **Bayes-tanulás (Bayesian learning)** során egyszerűen kiszámítjuk minden egyes hipotézis valószínűségét az adatokra támaszkodva, majd ennek alapján adunk predikciót. Azaz nem egyetlen „legjobb” hipotézist használunk a predikcióhoz, hanem az összes hipotézist használjuk, valószínűségükkel súlyozva őket. Reprezentálja  $\mathbf{D}$  az összes adatot, legyen  $\mathbf{d}$  a megfigyelt értékek vektora, ekkor az egyes hipotézisek valószínűségét a Bayes-szabállyal adhatjuk meg:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i) \quad (20.1)$$

Tegyük fel, hogy egy ismeretlen  $X$  mennyiségre vonatkozó predikció a célunk. Ebben az esetben:

$$P(X|\mathbf{d}) = \sum_i P(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i P(X|h_i)P(h_i|\mathbf{d}) \quad (20.2)$$

ahol azt feltételeztük, hogy az összes hipotézis meghatároz  $X$ -re valamilyen eloszlást. Ez az összefüggés azt mutatja, hogy a predikció az egyes hipotézisekből adódó predikciók súlyozott összege. Maguk a hipotézisek tulajdonképpen a nyers adatok és a predikciók közti „közvetítők”. A Bayes-megközelítésben a  $P(h_i)$  **prior hipotézisek (hypothesis prior)**, illetve a hipotézisek mellett fellépő  $P(\mathbf{d}|h_i)$  **adatvalószínűségek (likelihood)** a kulcsmennyiségek.

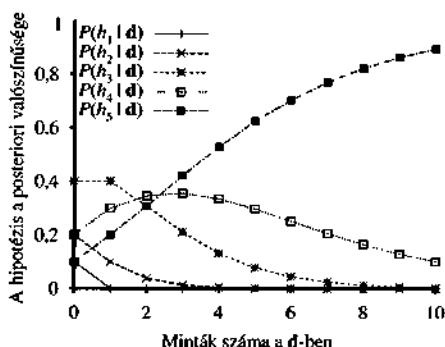
A cukorka példánkban azt feltételezzük, hogy a  $h_1, \dots, h_5$  a priori valószínűségei megfelelnek a gyártó reklámjában közölt  $(0,1, 0,2, 0,4, 0,2, 0,1)$  értékeknek. Az adatok valószínűségét **e.f.e** (angol rövidítése: **i.i.d.**) feltételezéssel számítjuk – azaz egyforma és független eloszlást (**independently and identically distributed**) teszünk fel, így:

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i) \quad (20.3)$$

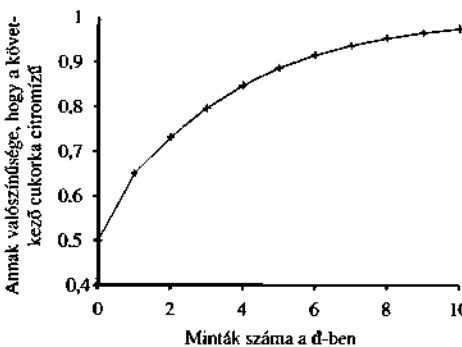
Tegyük fel például, hogy a cukorkás zsák valójában csupa citromtípusú ( $h_5$ ), és az első 10 kibontott cukorka mind citromízű, ekkor  $P(\mathbf{d}|h_5) = 0,5^{10}$ , mivel a  $h_5$  típusú zsákok-

<sup>1</sup> A statisztikában jártasabb olvasó felismeri, hogy ez a példa valójában az **urna és golyó (urn and ball)** feladat változata. Úgy találtuk, hogy az urna és a golyó kevesebb kihívást jelent, mint a cukorka, továbbá a cukorka példa elvezet egy másik feladathoz – elcseréljük-e a zacskót egy barátunkkal, vagy sem (lásd 20.3. feladat).

ban a cukorkák fele citrom.<sup>2</sup> A 20.1. (a) ábra mutatja, hogyan változik az öt hipotézis a posteriori valószínűsége, ahogy sorban észleljük a 10 citromízű cukorkát. Vegyük észre, hogy a valószínűségek az a priori értékekről indulnak, ennek megfelelően kezdetben  $h_3$  a legvalószínűbb lehetőség, és ez így is marad még az első cukor felbontása után is. A második citromízű cukorka felbontása után  $h_4$  a legvalószínűbb, 3 és több esetén  $h_5$  (a rettegett csupa citrom zsák). 10 citromízű cukorka után már meglehetősen biztosak vagyunk végzetünket illetően. A 20.1. (b) ábra mutatja annak a (20.2) egyenlet alapján jóolt valószínűségét, hogy a következő cukorka citromízű. Várakozásunknak megfelelően monoton növekszik az 1 felé.



(a)



(b)

20.1. ábra. (a) A (20.1) egyenletből számított  $P(h_i | d_1, \dots, d_N)$  a posteriori valószínűségek. A megfigyelések száma 1-től 10-ig terjed, és minden egyik megfigyelés citromízű cukorka. (b) A (20.2) egyenlet alapján számított  $P(d_{N+1} = \text{citrom} | d_1, \dots, d_N)$  Bayes-predikció.

 Példánk azt mutatja, hogy a Bayes-predikcióban az igaz hipotézis végülis dominánsáválik. Ez jellemző a Bayes-tanulásra. Bármilyen rögzített priorra, amely nem zárja ki a helyes megoldást, a hamis hipotézisek a posteriori valószínűsége végülis nullához tart. Ennek egyszerűen az az oka, hogy elhanyagolhatóan kicsi annak valószínűsége, hogy végig telen ideig „nem jellemző” adatokat generálunk. (Ezen a ponton érvelésünk hasonló a 18. fejezetben, a VKH-tanulásnál alkalmazottal.) Ennél is fontosabb, hogy a Bayes-predikció optimális, akár kicsi, akár nagy adathalmazunk van. Adott a priori hipotéziseloszlás mellett bármely más predikció ritkábban lesz helyes, mint a Bayes-predikció.

A Bayes-tanulás optimalitásának természetesen ára van. Mint a 18. fejezetben láttuk, a valós tanulási problémáknál a hipotézistér rendszerint nagyon nagy vagy végtelen. Néha a (20.2) egyenletben az összegzés (vagy folytonos esetben az integrálás) pontos elvégzése kezelhető problémára vezet, de a legtöbb esetben közelítő vagy egyszerűsített megoldásokra kell szorítkoznunk.

Nagyon elterjedt approximációs módszer – a tudományos feladatokban rendszerint ezt alkalmazzuk –, hogy egyetlen, a legvalószínűbb hipotézis alapján végezzük a predikciót.

<sup>2</sup> Korábbiakban leszögeztük, hogy a cukorkás zsákok nagyon nagyok, másképp az e.f.e. feltétel nem áll fenn. Korrektebb lenne (de kevésbé higiénikus) feltételezni, hogy minden vizsgálat, kóstolás után visszacsomagoljuk a cukorkát, és visszatesszük a zsákba.

azaz olyan  $h_i$  alapján, amely maximálja a  $P(h_i|\mathbf{d})$ -t. Ezt **maximum a posteriori** vagy MAP hipotézisnek nevezzük. A MAP hipotézis alapján végzett predikciók közelítőleg Bayes-predikciók. Ez a közelítés annyira jó, amennyire jó a  $\mathbf{P}(\mathbf{X}|\mathbf{d}) \approx \mathbf{P}(\mathbf{X}|h_{\text{MAP}})$  közelítés. A cukorka példánkban három egymás utáni citromízű cukor észlelése után  $h_{\text{MAP}} = h_5$ , így a MAP-tanulás alapján a negyedik cukorra 1,0 valószínűséggel citromízűt jó-solunk. Ez nyilván sokkal veszélyesebb jóslat, mint a Bayes-predikció, amely 0,8 valószínűségi, ahogy a 20.1. ábrán is láthatjuk. Ahogy egyre több adatunk van, a MAP és Bayes-predikciók egyre inkább konvergálnak egymáshoz, mivel a MAP hipotézis alternatívái egyre kevésbé valószínűvé válnak. Bár példánk ezt nem mutatja, de a MAP hipotézis előállítása sokszor lényegesen egyszerűbb, mint a Bayes-tanulásé. Ennek oka, hogy csupán egy optimalizálási probléma megoldását igényli, szemben egy nagyon nagy összegzési (vagy integrálási) problémával. A fejezet későbbi részében majd látunk példákat erre.

Mind a Bayes-tanulásban, mind a MAP-tanulásban a  $P(h_i)$  a priori hipotézis valószínűségek nagy szerepet játszanak. Láttuk a 18. fejezetben, hogy **túlilleszkedés (overfitting)** léphet fel, ha túlzottan nagy a hipotézistér kifejezőképessége, vagyis túl sok olyan hipotézist tartalmaz, amely jól illeszkedik az adatokra. A Bayes- és a MAP-tanulás nem alkalmaz valamilyen önkényes korlátot a figyelembe vett hipotézisekre, inkább az a priori valószínűségeket használják fel arra, hogy *büntessék a hipotézisek komplexitását*. A komplex hipotéziseknek tipikusan kisebb az a priori valószínűsége – részben azért, mert sokkal több komplex hipotézis van, mint egyszerű. Másrészről a komplex hipotéziseknek nagyobb kapacitása van az adatokra való illeszkedéshez. (Extrém esetet véve egy táblázat 1,0 valószínűséggel és tökéletes pontossággal reprodukálni tudja az adatokat.) Ennek megfelelően az a priori valószínűségben testesül meg a hipotéziskomplexitás és az adatokra való illeszkedési képesség közötti kompromisszum.

Ezt a kompromisszumot a logikai esetben figyelhetjük meg legjobban, amikor is  $H$  csak *determinisztikus* hipotéziseket tartalmaz. Ebben az esetben  $\mathbf{P}(\mathbf{d}|h_i)$  értéke akkor 1, ha  $h_i$  konziszens az adatokkal, különben 0. A (20.1) egyenlet alapján azt látjuk, hogy  $h_{\text{MAP}}$  a *legegyszerűbb logikai hipotézis lesz, amely konziszens az adatokkal*. Tehát a maximum a posteriori tanulás Ockham borotvájának egy természetes megvalósulása.

A hipotéziskomplexitás és az adatokra való illeszkedési képesség közötti kompromisszumot új módon világítja meg, ha a (20.1) egyenlet logaritmusát képezzük. A  $h_{\text{MAP}}$  hipotézis olyan kiválasztása, amely maximálja a  $\mathbf{P}(\mathbf{d}|h_i)\mathbf{P}(h_i)$ -t ugyanaz, mint amikor minimalizáljuk a következő kifejezést:

$$-\log_2 P(\mathbf{d}|h_i) - \log_2 P(h_i)$$

Használjuk fel az információkódolás és a valószínűség között a 18. fejezetben bevezetett kapcsolatot. Azt látjuk, hogy a  $-\log_2 P(h_i)$  tag nem más, mint a  $h_i$  hipotézis specifikálásához szükséges bitek száma. A  $-\log_2 P(\mathbf{d}|h_i)$ -tag viszont azoknak a további biteknek a száma, amelyek ahhoz szükségesek, hogy az adott hipotézis feltételezésével specifikáljuk az adatokat. (Ennek demonstrálására mutatjuk be azt az esetet, amikor a hipotézis pontosan megjósolja az adatokat, ilyenkor nincs szükség egyetlen bite sem az adatok specifikálásához. Ilyen például a  $h_5$  hipotézis esete, amikor sorban érkeznek a citromízű cukorkák – és valóban  $\log_2 1 = 0$ .) Ebben az értelemben a MAP-tanulás jellemzője, hogy maximálisan *tömöríti* az adatokat. Ezt a feladatot sokkal közvetlenebbül

célozza a **minimális hosszúságú leírás (MHL)** (**minimum description length, MDL**) tanulási módszer, amely a valószínűségekkel való foglalkozás helyett a hipotézis méreteinek és az adat kódolásának minimalizálására törekszik.

Az utolsó egyszerűsítést az adja, ha a hipotézistérben **egyenletes (uniform)** priort feltételezzük. Ebben az esetben a MAP-tanulás egy olyan  $h_i$  választására redukálódik, amely maximálja  $P(\mathbf{d}|h_i)$ -t. Ezt **maximum-likelihood (ML)** hipotézisek nevezik, és  $h_{ML}$ -lel jelöljük. A maximum-likelihood tanulás nagyon elterjedt a statisztikában. Ez egy olyan tudomány, amelynek sok kutatója nem bízik az a priori hipotézisek szubjektív természetében. Ez józan megközelítés akkor, amikor nincs semmi okunk, hogy a priori kitüntessük az egyik hipotézist egy másikkal szemben, például amikor az összes hipotézis egyformán komplex. Az ML-tanulás jó közelítését adja a Bayes- és MAP-tanulásnak olyankor, amikor az adathalmaz nagy, hiszen az adatok végül is felülírják a hipotézisek a priori eloszlását, de kis adathalmazok esetén problémák merülnek fel az alkalmazásánál (mint látni foguk).

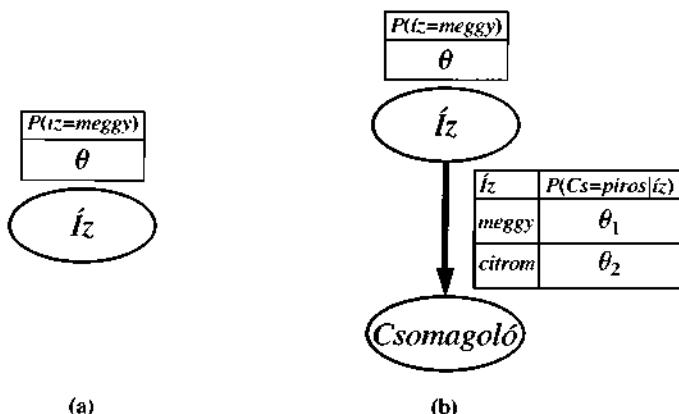
## 20.2. TELJES ADATTAL TÖRTÉNŐ TANULÁS

A statisztikai tanulási módszerek tárgyalását a legegyszerűbb feladattal kezdjük: **paramétertanulás (parameter learning) teljes adat (complete data)** alapján. A paramétertanulás egy rögzített struktúrájú valószínűségi modell paramétereinek megtalálását foglalja magában. Például meg akarjuk tanulni egy adott struktúrájú Bayes-háló feltételes valószínűségeit. Az adatokat akkor nevezzük teljesnek, ha minden adatpont értékeit hordoz a megtanulandó valószínűségi modell minden paramétereire. A teljes adatok nagyban egyszerűsítik a komplex modellek paramétereinek tanulását. Nagy vonalakban áttekintjük majd a struktúratanulás problematikáját is.

### Maximum-likelihood paramétertanulás: diszkrét modellek

Tegyük fel, hogy új gyártótól vásárolunk egy zsák citrom- és meggycukorkát, a meggycarány teljesen ismeretlen, bárhol lehet 0 és 1 között. Ez esetben kontinuum számosságú hipotézünk van. A **paraméter (parameter)**, amelyet  $\theta$ -val jelölünk, most a meggycukorkák aránya, a hipotézis pedig  $h_\theta$ . (A citromízek aránya egyszerűen  $1 - \theta$ .) Ha feltételezzük, hogy a priori minden arány egyformán valószínű, akkor a maximum-likelihood megközelítés az ésszerű. Ha Bayes-hálóval modellezünk a helyzetet, akkor csupán egyetlen véletlen változóra van szükségünk. Legyen ez az  $\bar{z}$  nevű változó (a zacskóból véletlenszerűen választott cukorka íze). Lehetséges értékei a *meggy* és a *citrom*, ahol a *meggy* valószínűsége  $\theta$  (lásd 20.2. (a) ábra). Tegyük fel, hogy kibontunk  $N$  cukorkát, amelyek közül  $c$  meggýízű és  $\ell = N - c$  citromízű. A (20.3) egyenlet alapján ennek a speciális adathalmaznak a valószínűsége:

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^N P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell$$



20.2. ábra. (a) Bayes-háló modell az ismeretlen arányban citrom-, illetve meggyízű cukorkák esetére.  
 (b) Annak a modellje, amikor a csomagolópapír színe függ (valószínűségi alapon) a cukorka ízétől.

A maximum-likelihood hipotézist az a  $\theta$  érték adja, amely maximálja ezt a kifejezést. Ugyanezt az értéket kapjuk, ha a **log likelihood** függvényt maximáljuk.

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^N \log P(d_j|h_\theta) = c \log \theta + \ell \log(1 - \theta)$$

(A kifejezés logaritmusát képezve a szorzatot szummává redukáltuk, amit rendszerint egyszerűbb maximálni.) A maximum-likelihood  $\theta$  érték megtalálása érdekében differenciáljuk  $L$ -et  $\theta$  szerint, a kapott kifejezést pedig tegyük egyenlővé nullával:

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

Magyaráz a  $h_{ML}$  maximum-likelihood hipotézis azt állítja, hogy a zacskóban a meggyízű cukorkák valós aránya megegyezik az eddig kibontott cukorkáknál megfigyelt aránnyal!

Úgy tűnik, rengeteget dolgoztunk, hogy felfedezzünk egy nyilvánvaló eredményt. Valójában lefektettünk egy standard módszert a maximum-likelihood paramétertanulásra:

1. Írunk fel egy – a paraméter(ek)től függő – kifejezést az adatok együttes valószínűségére (írjuk fel a likelihood függvényt).
2. Írjuk fel minden egyes paraméter szerint a log likelihood függvény deriváltját.
3. Keressük meg azokat a paraméterértékeket, amelyek mellett a deriváltak nulla értéket vesznek fel.

A legtrükkösebb lépés általában az utolsó. Az előző példánkban triviálisan megoldható volt, de látni fogjuk, hogy sokszor iteratív megoldásokhoz vagy más numerikus optimalizálási technikákhoz kell folyamodnunk, mint ahogy a 4. fejezetben tárgyalunk. A példa a maximum-likelihood tanulás egy általános problémáját is illusztrálja: *ha az adathalmaz elég kicsi ahhoz, hogy néhány eseményt még nem figyeltünk meg – például nem találtunk még meggyízű cukorkát –, akkor a maximum-likelihood hipotézis nulla valószínűséget rendel ezekhez az eseményekhez. Számos trükköt használnak, hogy*



elkerüljék ezt a problémát, mint például minden esemény kezdeti valószínűségét 1-re állítják nulla helyett.

Nézzünk egy másik példát. Tegyük fel, hogy a cukorka gyártója némi információt akar adni a fogyasztónak, ezért piros és zöld csomagolópapír használ. A *Csomagoló* mindegyik cukorkához véletlenszerűen kerül kiválasztásra, valamilyen ismeretlen – az íztől függő – feltételes valószínűség-elozlás szerint. Az ehhez tartozó valószínűségi modellt a 20.2. (b) ábra mutatja. Vegyük észre, hogy három paramétere van:  $\theta$ ,  $\theta_1$  és  $\theta_2$ . Ezekkel a paraméterekkel a Bayes-hálók standard szemantikáját használva megadható annak a valószínűsége, hogy – mondjuk – egy meggyízű cukorkát találunk egy zöld csomagolóban (581–582. oldal):

$$\begin{aligned} P(\bar{I}_Z = \text{meggy}, \text{Csomagoló} = \text{zöld} | h_{\theta}, \theta_1, \theta_2) \\ = P(\bar{I}_Z = \text{meggy} | h_{\theta}, \theta_1, \theta_2)P(\text{Csomagoló} = \text{zöld} | \bar{I}_Z = \text{meggy}, h_{\theta}, \theta_1, \theta_2) \\ = \theta(1 - \theta_1) \end{aligned}$$

Most kibontunk  $N$  cukorkát, amelyek közül  $c$  meggyízű és  $\ell$  citromízű. A csomagolásfajták számai a következők:  $r_c$  meggyízű volt pirosba csomagolva és  $g_c$  zöldbe, míg  $r_\ell$  citromízű volt pirosba csomagolva, míg  $g_\ell$  zöldbe. Ezen adatok együttes valószínűsége:

$$P(\mathbf{d} | h_{\theta}, \theta_1, \theta_2) = \theta^c(1 - \theta)^\ell \theta_1^{r_c}(1 - \theta_1)^{g_c} \theta_2^{r_\ell}(1 - \theta_2)^{g_\ell}$$

Ez meglehetősen ijesztőnek tűnik, de segít, ha a logaritmusát vesszük:

$$L = [c \log \theta + \ell \log(1 - \theta)] + [r_c \log \theta_1 + g_c \log(1 - \theta_1)] + [r_\ell \log \theta_2 + g_\ell \log(1 - \theta_2)]$$

A logaritmusképzés előnye nyilvánvaló: a log likelihood függvény három tag összege, ahol mindenki tag csupán egyetlen paramétert tartalmaz. Amikor sorban mindenki paraméter szerint vesszük az összefüggés deriváltját, majd nullává tesszük a deriváltakat, három független egyenlethez jutunk, és mindenki csupán egyetlen paramétert tartalmaz:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c+g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell}$$

A  $\theta$ -ra kapott megoldás ugyanaz, mint az előbb. A  $\theta_1$ -re kapott megoldás, tehát annak valószínűsége, hogy egy meggyízű cukorka piros papírba csomagolt, nem más, mint a megfigyelt meggyízű cukorka – piros papír arány, hasonló a helyzet  $\theta_2$ -vel.

Ezek az eredmények nagyon kényelmesek, és könnyen belátható, hogy kiterjeszhetők bármely Bayes-hálóra, amelynek feltételes valószínűségeit táblázatokkal adjuk meg. A legfontosabb eredmény a következő: *teljes adatok esetén a Bayes-háló paramétertanulási problémája elkölönlő tanulási problémára dekomponálható, egy-egy*



probléma egy-egy paraméterre.<sup>3</sup> A második eredmény, hogy az egyes paraméterek adott szülő melletti értékei éppen a szülőértékek mellett megfigyelt gyakoriságokkal egyeznek meg. Éppúgy, mint az előző helyzetben, itt is óvatosnak kell lennünk, hogy kis adathalmazok esetén el tudjuk kerülni a nulla értékeket.

## Naív Bayes-modellek

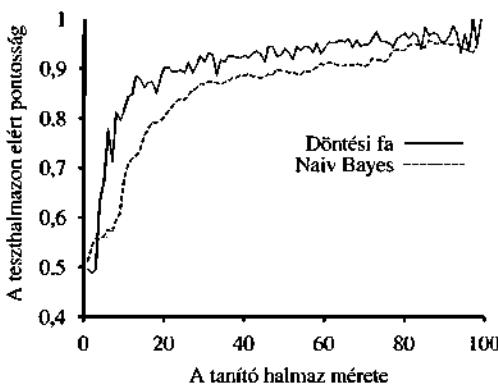
Valószínűleg a gépi tanulás területén használt legelterjedtebb Bayes-háló modell a **naiv Bayes-modell** (*naive Bayes*). Ebben a modellben a  $C$  „osztályváltozó” (amelyet meg akarunk játsolni) a gyökér, míg az  $X_i$  attribútumváltozók a levelek. A modell azért „naiv”, mert feltételezi, hogy adott osztály mellett az attribútumok feltételesen függetlenek egymástól. (A 20.2. (b) ábra modellje egy egyváltozós naiv Bayes-modell.) Logikai változókat feltételezve a paraméterek:

$$\theta = P(C = \text{igaz}), \quad \theta_{i1} = P(X_i = \text{igaz}|C = \text{igaz}), \quad \theta_{i2} = P(X_i = \text{igaz}|C = \text{hamis})$$

A maximum-likelihood paramétereket pontosan úgy kapjuk meg, mint a 20.2. (b) ábra esetén. Ha a modellt ezen az úton megtanítottuk, akkor felhasználható arra, hogy olyan új példákat osztályozzon, amelyekre a  $C$  osztályváltozó nem ismert. A megfigyelt  $x_1, \dots, x_n$  attribútumértékek mellett az egyes osztályok valószínűségét a következő összefüggés adja:

$$P(C|x_1, \dots, x_n) = \alpha P(C) \prod_i P(x_i|C)$$

Ha a legvalószínűbb osztályt választjuk, determinisztikus predikció adható. A 20.3. ábra mutatja a módszer tanulási görbéjét, ha a 18. fejezet éterem problémájára alkalmazzuk. A módszer elég jól tanul, de nem olyan jól, mint egy döntési fa tanulás. Ennek oka valószínűleg az, hogy a helyes hipotézis – ami egy döntési fa – nem reprezentálható pontosan naiv Bayes-modellel. A naiv Bayes-tanulás sok alkalmazási területen meglepően jól



20.3. ábra. A 18. fejezet éterem problémájára alkalmazott naiv Bayes-tanulás tanulási görbéje. A döntési fa tanulási görbüjét összehasonlítás céljából ábrázoltuk.

<sup>3</sup> Nem táblázatos formára lásd a 20.7. feladatot, amelyben mindegyik paraméter hatással van számos feltételezett valószínűségre.

 teljesít, a fokozott teljesítményű, turbó változata (*boosted version*) egyike a leghatékonyabb általános célú tanuló algoritmusoknak (lásd 20.5. feladat). A módszer nagyon nagy méretű problémákhoz is jól alkalmazható,  $n$  logikai változó esetén is csak  $2n + 1$  paramétere lesz, és *ahhoz, hogy a naiv Bayes-tanulás meghatározza  $h_{ML}$ -t, nincs szükség keresésre*. Végül a naiv Bayes-tanulásnak nem jelentenek gondot a zajos adatok, továbbá ha szükséges, akkor képes valószínűségi predikciókat is adni.

## Maximum-likelihood paramétertanulás: folytonos eset

A 14.3. alfejezetben vezettük be a folytonos valószínűségi modelleket, mint például a **lineáris Gauss-** (**linear-Gaussian**) modelleket. Mivel a valós alkalmazásokban mindenütt folytonos változókkal találkozunk, fontos ismernünk a folytonos modellek adatokból történő megtanulásának módszereit. A maximum-likelihood tanulás elvei azonosak a diszkrét esetre vonatkozó tanulás elveivel.

Kezdjük egy rendkívül egyszerű esettel: egyetlen skalár változó Gauss-sűrűségfüggvényének a paramétereit tanuljuk. Azaz az adatokat a következő összefüggéssel generáljuk:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

A modell paraméterei a  $\mu$  átlag és a  $\sigma$  szórás. (Vegyük észre, hogy a normalizáló konstans is függ  $\sigma$ -től, ezért nem hanyagolhatjuk el.) Legyenek a megfigyelt értékek  $x_1, \dots, x_N$ . Ekkor a log likelihood:

$$L = \sum_{j=1}^N \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log \sqrt{2\pi} - \log \sigma) - \sum_{j=1}^N \frac{(x_j-\mu)^2}{2\sigma^2}$$

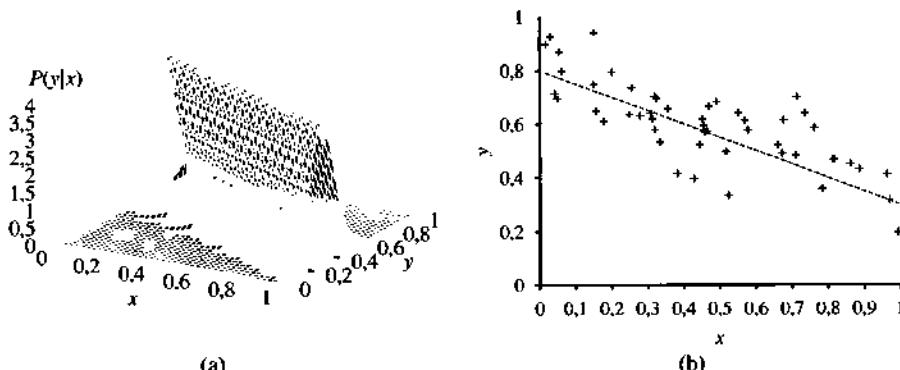
A deriváltakat szokásos módon nullává téve a következőket kapjuk:

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2} \sum_{j=1}^N (x_j - \mu) = 0 \quad \Rightarrow \quad \mu = \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^N (x_j - \mu)^2 = 0 \quad \Rightarrow \quad \sigma = \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}} \end{aligned} \quad (20.4)$$

Tehát az átlag maximum-likelihood becslése a mintaátlag, a szórás maximum-likelihood becslése pedig a minta átlagos szórásnégyzetének négyzetgyöke. Ezek ismét kedvező eredmények, mivel megerősítik a „józan ésszel” követett gyakorlatot.

Vizsgálunk most egy lineáris Gauss-modellt, amelyben egy  $X$  folytonos szülő és  $Y$  folytonos gyermek van. Mint az 589. oldalon megmutattuk,  $Y$  Gauss-eloszlású, átlaga lineárisan függ  $X$ -től, míg varianciája rögzített. A  $P(X|Y)$  feltételes eloszlás tanulásához maximalizálhatjuk a feltételes likelihood függvényt:

$$P(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y-(\theta_1 x + \theta_2)^2}{2\sigma^2}} \quad (20.5)$$



**20.4. ábra.** (a) Egy  $y = (\theta_1 + \theta_2 + \epsilon)$  egyenlettel leírható lineáris Gauss-modell additív, rögzített varianciájú Gauss-zajjal. (b) Ez modell alapján generált 50 adatpontból álló halmaz.

Itt a paraméterek  $\theta_1$ ,  $\theta_2$  és  $\sigma$ . Mint a 20.4. ábrán szemléltettük, az  $(x_j, y_j)$  párok gyűjteménye adja az adatokat. A szokásos módszereket használva (lásd 20.6. feladat) megkaphatjuk a paraméterek maximum-likelihood értékét. Itt most egy másik dolgot akarunk megmutatni. Végük csupán a  $\theta_1$  és  $\theta_2$  paramétereket, amelyek az  $x$  és  $y$  közti lineáris összefüggést definiálják! Nyilvánvaló, hogy a log likelihood ezen paraméterekkel történő maximalizálása azonos azzal, mintha a (20.5) kifejezésben a kitevő számlálóját *minimalizálnánk*:

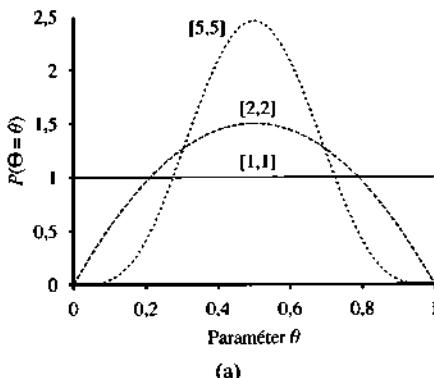
$$E = \sum_{j=1}^N (y_j - (\theta_1 x_j + \theta_2))^2$$

Az  $(y_j - (\theta_1 x_j + \theta_2))$  mennyiség valójában az  $(x_j, y_j)$  hibája (**error**) – azaz a tényleges  $y_j$  érték és a becsült érték különbsége. Így  $E$  nem más, mint a jól ismert **hibanégyzetek összege** (**sum of squared errors**). Ezt a standard **lineáris regresszió** (**linear regression**) minimalizálja. Most megérhetjük, hogy miért: a hibanégyzetek összegének minimalizálása nem más, mint a maximum-likelihood lineáris (egyenessel ábrázolható) modell megadása, feltéve, hogy az adatokat rögzített varianciájú Gauss-zaj mellett generáltuk.

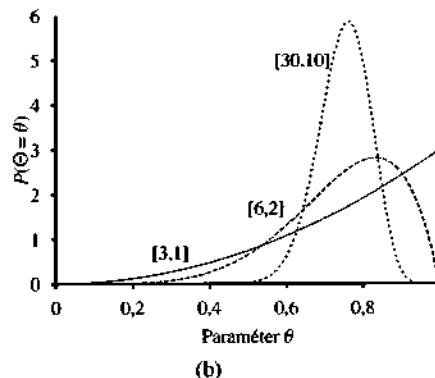
## Bayes-paramétertanulás

A maximum-likelihood tanulás alkalmat ad néhány nagyon egyszerű eljárás létrehozására, de kis adathalmazok esetén súlyos hiányosságokat mutat. Például egyetlen meggiszű cukorka észlelése után az a maximum-likelihood hipotézis, hogy a csomag 100%-a meggtípusú (azaz  $\theta = 1.0$ ). Ha nincs olyan hipotézis prior, hogy a csomagok mind vagy csupa megg-, vagy csupa citromtípusúak, akkor ez nem józan következetes. A Bayes-megközelítésű paramétertanulás egy hipotézis priorral állít fel a lehetséges paraméterértékekre, és ahogyan az adatok érkeznek, úgy frissíti az eloszlást.

A 20.2. (a) cukorka példának egyetlen  $\theta$  paramétere van; annak valószínűsége, hogy egy véletlenszerűen kiválasztott cukorka meggiszű. Bayes-megközelítésben  $\theta$  a  $\Theta$  valószínűségi változó (ismeretlen) értéke, a hipotézis prior pedig nem más, mint a  $P(\Theta)$



(a)



(b)

20.5. ábra. Példák a béta $[a,b]$  eloszlásra különböző  $[a,b]$  értékek esetén

a priori eloszlás. Így  $P(\Theta = \theta)$  annak a priori valószínűsége, hogy a csomag  $\theta$  arányban tartalmaz meggyűzű cukrokat.

Ha a  $\theta$  tetszőleges értéket felvehet 0 és 1 között, akkor a  $P(\Theta)$ -nek egy folytonos eloszlásnak kell lennie, amely csak 0 és 1 között nem nulla értékű, és integrálja 1. Egy lehetséges jelölt az egyenletes eloszlás  $P(\theta) = U[0, 1](\theta)$ . (Lásd 13. fejezet.) Az egyenletes eloszlás a **béta-eloszlások (beta distributions)** családjának tagja. minden egyes béta-eloszlás két **hiperparaméterrel**<sup>4</sup> (hyperparameter) –  $a$ -val és  $b$ -vel – definiálható a következő egyenlet szerint:

$$\text{beta}[a,b](\theta) = \alpha \theta^{a-1} (1-\theta)^{b-1} \quad (20.6)$$

A (20.6) megadja  $\theta$ -ra a  $[0, 1]$  tartományban a sűrűségfüggvény értékét. Az  $\alpha$  normális konstans  $a$ -tól és  $b$ -től függ. (Lásd 20.8. feladat.) A 20.5. ábrán bemutatjuk, hogy hogyan néz ki az eloszlás különböző  $a$ -k és  $b$ -k esetén. Az eloszlás átlaga  $a/(a+b)$ , tehát nagyobb  $a$  értékek arra utalnak, hogy  $\Theta$ -t 1-hez közelebb hisszük, mint 0-hoz. Az  $a+b$  nagyobb értékei az eloszlást csúcsosabbá teszik, ami a  $\Theta$  értéke felőli nagyobb bizonyosságunkat jelenti. Látható, hogy a béta-család a hipotézis prior lehetőségek hasznos választékát nyújtja.

Rugalmasságán túl a béta-családnak van még egy csodálatos tulajdonsága: ha a  $\Theta$  priorja béta $[a, b]$ , akkor egy adatpont megfigyelése után  $\Theta$  a posteriori eloszlása is béta-eloszlás. A béta-családot a logikai változók eloszlása **konjugált priorjának (conjugate prior)** nevezik.<sup>5</sup> Lássuk, hogyan is működik ez. Tegyük fel, hogy megfigyeltünk egy meggyűzű cukrot, ekkor:

$$\begin{aligned} P(\theta | D_1 = \text{meggy}) &= \alpha P(D_1 = \text{meggy} | \theta) P(\theta) \\ &= \alpha' \theta \cdot \text{beta}[a,b](\theta) = \alpha' \theta \cdot \theta^{a-1} (1-\theta)^{b-1} \\ &= \alpha' \theta^a (1-\theta)^{b-1} = \text{beta}[a+1, b](\theta) \end{aligned}$$

<sup>4</sup> Azért hívjuk hiperparamétereiknek, mert  $\theta$  eloszlásának paramétereiről van szó, ahol  $\theta$  maga is egy paraméter.

<sup>5</sup> További konjugált priorok: a diszkrét többváltozós eloszlások paramétereire a Dirichlet család, a Gauss-eloszlások paramétereire a Normal-Wishart család. Lásd Bernardo és Smith (1994).

Tehát egy meggyűzű cukrot észlelve egyszerűen inkrementáljuk az  $a$  paramétert, hasonlóképpen, ha egy citromízű észlelünk, akkor inkrementáljuk a  $b$  paramétert – ezzel megkapjuk az a posteriori eloszlást. Ezek szerint úgy tekinthetünk  $a$ -ra és  $b$ -re, mint virtuális számlálókra (virtual counts), abban az értelemben, hogy a béta $[a, b]$  prior pontosan úgy viselkedik, mintha egyenletes eloszlású priorral indultunk volna, és  $a - 1$  meggycukorkát és  $b - 1$  citromízű cukorkát láttunk volna.

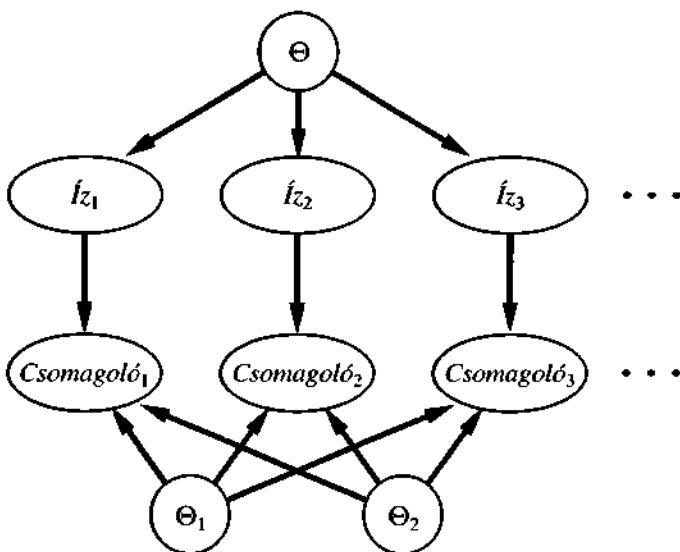
Tanulmányozva a béta-eloszlások sorozatát növekvő – de állandó arányú –  $a$  és  $b$  mentén, jól láthatjuk, hogyan változik a  $\Theta$  paraméter a posteriori eloszlása az adatok beérkezése során. Tegyük fel például, hogy a vizsgált zacskó 75%-a meggytípusú. A 20.5. (b) ábra mutatja a béta $[3, 1]$ , béta $[6, 2]$ , béta $[30, 10]$  eloszlássorozatot. Nyilvánvaló, hogy az eloszlás egy – a valós  $\Theta$  körül elhelyezkedő – keskeny csúcs felé tart. Nagy adathalmazok esetén a Bayes-tanulás (legalábbis ebben az esetben) ugyanahhoz az eredményhez konvergál, mint amit a maximum-likelihood tanulás adott.

A 20.2. (b) ábrán a hálonak három paramétere volt:  $\theta$ ,  $\theta_1$  és  $\theta_2$ , ahol  $\theta_1$  volt annak valószínűsége, hogy piros csomagolás van egy meggycukorkán, míg  $\theta_2$  az, hogy egy citromízű cukron van piros csomagolás. A Bayes hipotézis priornak minden háróm paramétert le kell fednie – azaz  $P(\Theta, \Theta_1, \Theta_2)$ -t kell specifikálnunk. Rendszerint paraméter-függetlenséget (parameter independence) tételezünk fel:

$$P(\Theta, \Theta_1, \Theta_2) = P(\Theta)P(\Theta_1)P(\Theta_2)$$

Ezen feltételezés esetén minden egyes paramétermek saját béta-eloszlása lehet, amelyet külön-külön frissíthetünk az adatok érkeztekor.

Ha már az volt az ötletünk, hogy az ismeretlen paramétereket valószínűségi változókkal reprezentáljuk – amilyen például  $\Theta$  –, akkor természetes módon adódik, hogy azokat beépítük magába a Bayes-hálóba. Ahhoz, hogy ezt megtehessük, minden egyes



20.6. ábra. Egy Bayes-tanulásnak megfelelő Bayes-háló. A  $\Theta$ ,  $\Theta_1$ ,  $\Theta_2$  változók a posteriori eloszlásai kikövetkeztethetők az a priori eloszlásokból és az  $\bar{z}_i$ ,  $Csomagoló_j$  változókra vonatkozó tényekből.

példa leírásához másolatot kell készítenünk a változókról. Ha például három cukorkát figyeltünk meg, akkor szükségünk van a következő változóra:  $\bar{Iz}_1$ ,  $\bar{Iz}_2$ ,  $\bar{Iz}_3$  és  $Csomagoló_1$ ,  $Csomagoló_2$ ,  $Csomagoló_3$ . A  $\Theta$  paraméterváltozó határozza meg minden egyes  $\bar{Iz}_i$  változó valószínűségét:

$$P(\bar{Iz}_i = \text{meggy} | \Theta = \theta) = \theta$$

Hasonlóképpen a csomagoló valószínűsége  $\Theta_1$ -től és  $\Theta_2$ -től függ. Például:

$$P(Csomagoló_i = \text{piros} | \bar{Iz}_i = \text{meggy}, \Theta_1 = \theta_1) = \theta_1$$

Ezek után az egész Bayes-tanulási folyamat formalizálható egy megfelelően konstruált Bayes-háló következetési problémájaként, amint a 20.6. ábrán látható. Egy új példány predikciója egyszerűen azt jelenti, hogy új példányváltoztatást adunk a hálóhoz, amelyből egyesekre rákérdezünk. A tanulás és a predikció ezen formalizmusa nyilvánvalóvá teszi, hogy a Bayes-tanuláshoz nem kell semmilyen extra „tanulási elv”. Megállapíthatjuk továbbá, hogy *lényegében csak egyetlen tanulási algoritmus van*, ami a Bayes-háló következetési algoritmusa.



## Bayes-hálóstruktúrák tanulása

Az eddigiekben azt feltételeztük, hogy a Bayes-háló struktúrája ismert, és csak a paramétereit próbáljuk megtanulni. A háló struktúrája a terület alapvető oksági viszonyaira vonatkozó tudást reprezentálja, amit sok esetben egy szakember, de még egy naiv felhasználó is, nagyon egyszerűen meg tud adni. Néhány esetben azonban az oksági összefüggések nem állnak rendelkezésre vagy vitatottak – például bizonyos nagyvállalatok régóta állítják, hogy a dohányzás nem okoz rákot. Ilyenkor fontos megérteni, hogy a Bayes-háló struktúrája mi módon tanulható meg az adatkból. Jelenleg a struktúratanulási algoritmusok gyerekcipőben járnak, ezért csak egy elnagyolt vázlatot adunk a legfontosabb elvekről.

A legkézenfekvőbb megközelítés, ha egy jó modell érdekében *keresést* folytatunk. Elindulhatunk egy kapcsolatokat nem tartalmazó modellel, majd elkezdünk szülcőcsmóppontot adni minden csomóponthoz, az előbbiekbén bemutatott módszerekkel illesztve a paramétereket, és mérjük a modell pontosságát. Másik lehetőség, hogy egy becsült struktúrával indulunk, és hegymászó vagy szimulált lehűtést alkalmazó algoritmusokat használunk a módosításokhoz, minden egyes struktúrváltoztatás után újrahangolva a paramétereket. A módosítások közé tartozik az élek megfordítása, hozzáadása, törlése. Nem szabad ciklusokat létrehoznunk a folyamat során, ezért sok algoritmus azt feltételezi, hogy adott a változók egy rendezése, és egy csomópont szülője csak azon csomópontok közül kerülhet ki, amelyek előbb jönnek a rendezésben (éppúgy, mint a 14. fejezet konstrukciós eljárásában). A teljes általánosság kedvéért a lehetséges rendezések között is keresünk kell.

Két alternatív módszer van arra, hogy észrevegyük, amikor egy jó megoldást találtunk. Az első annak tesztelése, hogy az aktuális adatok kielégítik-e azokat a feltételes függetlenségre vonatkozó állításokat, amelyek a struktúrában implicit módon benne vannak. Például az értelem probléma egy naiv Bayes-modellje feltételezi, hogy:

$$\mathbf{P}(\text{Péntek/Szombat, Bár} | \text{VárjunkE}) = \mathbf{P}(\text{Péntek/Szombat} | \text{VárjunkE})\mathbf{P}(\text{Bár} | \text{VárjunkE})$$

és leellenőrizhetjük az adatokon, hogy ugyanez az egyenlet fennáll-e a megfelelő feltétes gyakoriságok között is. Viszont még akkor is, ha a struktúra a terület valós oksági természetét írja le, az adathalmaz – a statisztikai ingadozások miatt – az egyenletet soha nem elégíti ki *pontosan*. Ezért megfelelő statisztikai próbákat kell elvégezni, hogy eldöntsük: elég bizonyítékunk van-e arra, hogy a függetlenségi hipotézisek sérülnek. Az eredményül kapott háló bonyolultsága az ezen tesztekben alkalmazott küszöbtől függ majd – minél szigorúbb a függetlenségi teszt, annál több kapcsolatot adunk majd a struktúrához, és annál nagyobb lesz a túlleszkedés veszélye.

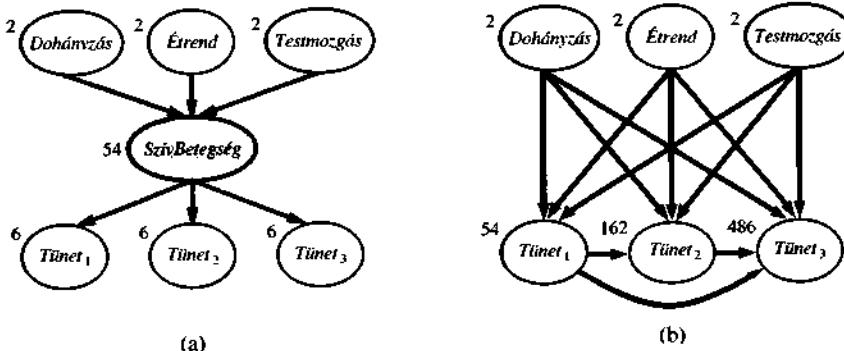
A jelen fejezetben bemutatott gondolatoknak jobban megfelelő megközelítés annak értékelése, hogy a javasolt modell mennyire magyarázza meg az adatokat (valószínűségi értelemben). Mindamellett óvatosnak kell lennünk ennek mérésénél. Ha egyszerűen a maximum-likelihood hipotézist akarjuk megtalálni, akkor egy teljesen összekötött hálónál fogunk kikötni, mivel további szülőcsomópontok hozzáadása egy csomóponthoz nem csökkentheti a valószínűséget (lásd 20.9. feladat). Valamilyen módon büntetnünk kell a modell bonyolultságát. A MAP (vagy MLH) megközelítés egyszerűen levon egy büntetőtagot az egyes struktúrák valószínűségéből (miután hangolta paramétereiket), ezek után hasonlíta össze a különböző struktúrákat. A Bayes-megközelítés a struktúrák és paraméterek együttes priorját használja. Rendszerint túl sok – a változók számán szuperexponenciális – struktúra van ahhoz, hogy minden felett összegezzünk, így a gyakorlatban legtöbben az MCMC (Markov lánc Monte Carlo) módszert használják, hogy mintát vegyenek a struktúrákból.

A bonyolultság büntetése (akár MAP, akár Bayes-megközelítésben) fontos kapcsolatot hoz be az optimális struktúra és a feltételes valószínűségek hálóbeli reprezentációs módja között. Táblázatosan ábrázolt eloszlás esetén a bonyolultság büntetése a szülőcsomópontok számával exponenciálisan nő, míg, mondjuk, zajos-VAGY eloszlások esetén csak lineárisan. Ez azt jelenti, hogy a zajos-VAGY (illetve más tömören paraméterezeit) modellek tanulása több szülőcsomópontot eredményez, mint a táblázatos eloszlás tanulása.

## 20.3. REJTETT VÁLTOZÓKKAL TÖRTÉNŐ TANULÁS: AZ EM ALGORITMUS

Az előző rész a teljesen megfigyelhető esettel foglalkozott. Számos valós problémában vannak **rejtett változók (hidden variables)**, amelyeket néha **latens változóknak (latent variables)** is hívnak. Ezek nem figyelhetők meg a tanulás céljára rendelkezésre álló adatokban. Például az orvosi feljegyzések gyakran tartalmazzák a megfigyelt tüneteket, az alkalmazott kezelést, esetleg e kezelés eredményét, de ritkán tartalmazzák magának a betegségnak a közvetlen megfigyelését.<sup>6</sup> Megkérdezhetnénk „ha a betegséget magát nem figyelhetjük meg, akkor miért nem állítunk fel egy modellt nélküle? A választ a 20.7. ábrán tüntettük fel, amely szívbetegségek egy kis, fiktív modelljét mutatja. Három hajlamosító tényező és három megfigyelhető tünet van (amelyek túl nyomasztók ahhoz, hogy megnevezzük őket). Tegyük fel, hogy mindegyik háromfélé értéket vehet fel (pl. *nincs*, *közepes*, *súlyos*). A rejtett változó eltávolítása az (a) hálóból a (b) hálóhoz

<sup>6</sup> Néhány feljegyzés tartalmazza az orvos által felállított diagnózist, de ez a szimptomák oksági következménye, amit viszont a betegség okozott.



**20.7. ábra.** (a) A rejtett változónak feltételezett szívbetegség egyszerű diagnosztikai hálója. minden egyes változónak három lehetséges értéke van, a változókat a feltételes eloszlásukban szereplő flüggetlen változók számával címkéztük, a végső számuk 78. (b) A SzívBetegség eltávolításával készült ekvivalens háló. Vegyük észre, hogy a tünetváltozók most már nem feltételesen flüggetlenek, ha a szülőket megadjuk. Ez a háló 708 paramétert igényel.



vezet; a paraméterek száma 78-ról 708-ra nő. Tehát a rejtett változók drámaián csökkenthetik a Bayes-háló megadásához szükséges paraméterek számát. Ez viszont drámaián csökkentheti a paraméterek megtanulásához szükséges adatok mennyiségét.

A rejtett változók fontosak, de bonyolítják a tanulási problémát. A 20.7. (a) ábrán például nem világos, hogy adott szülőcsomópontok esetén hogyan tanulható meg a SzívBetegség feltételes eloszlása, mivel nem ismert az értéke az egyes esetekben. Ugyanez a probléma fellép a tünetek eloszlásának tanulásánál is. Ebben a részben egy olyan – expectation–maximization,<sup>7</sup> EM nevű – algoritmust ismertetünk, amely nagyon általános megoldást ad erre a problémára. Hárrom példát mutatunk be, majd általános leírást adunk. Ez az algoritmus először holmi mágiának tűnik, de amint a mögöttes intuíciót megértettük, a tanulási problémák hatalmas területén egy sor helyen alkalmazhatjuk.

## Nem ellenőrzött osztályozás: Gauss-eloszlások keverékének tanulása

A **nem ellenőrzött osztályozás** vagy **klaszterezés** (*unsupervised clustering*) az a probléma, amikor objektumok valamelyen gyűjteményében több kategóriát különböztetünk meg. A probléma nem ellenőrzött, mivel a kategóriacímek nincsenek megadva. Tegyük fel például, hogy felvesszük százezer csillag spektrumát. Az a kérdés, hogy a spektrumok alapján vannak-e különböző típusú csillagok, és ha igen, akkor hányfélé, és mik a jellemzőik? Mindannyian jól ismerünk olyan neveket, mint „vörös óriás” és „fehér törpe”, de a csillagoknak nincs ilyen címke a kalapjukon – a csillagászoknak nem ellenőrzött osztályozást kell végezniük ahhoz, hogy ezeket a kategóriákat felállítsák. Más példák lehetnek: a fajok, nemek, rendek stb. azonosítása a Linné-féle taxonómiaiban vagy hétköznapi tárgyak természeti kategóriáinak megalkotása (lásd 10. fejezet).

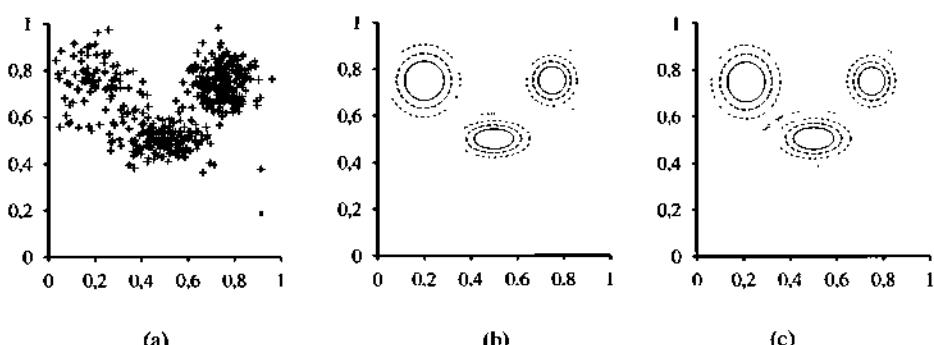
<sup>7</sup> Az eljárást magyarul **várhatóérték-képzés-maximalizás** eljárásnak nevezhetnénk, de a magyar szakirodalomban is az angol elnevezés terjedt el. (A szerk.)

A nem ellenőrzött osztályozás az adatokkal kezdődik. A 20.8. (a) ábra 500 adatpontot ábrázol, mindegyik két folytonos attribútum értékét adja meg. Az adatpontok megfelelhetnek pl. csillagoknak, a két attribútum pedig pl. a spektrumvonalak erőssége két adott frekvencián. A következő lépés, hogy meg kell értenünk, milyen valószínűségi eloszlás generálhatja az adatokat. Az osztályozás azt feltételezi, hogy az adatok a **P kevert eloszlásból** (**mixture distribution**) származnak. Egy ilyen eloszlás  $k$  komponensből (**component**) áll, amelyek mindegyike önmagában egy-egy eloszlás. Az adatpont úgy jön létre, hogy először kiválasztjuk az egyik komponensem, majd ebből a komponensből generálunk egy mintát. Jelöljük a  $C$  véletlen változóval a komponenseket, amely változik az 1, ...,  $k$  értékeket veheti fel. A kevert eloszlást ekkor a

$$P(\mathbf{x}) = \sum_{i=1}^k P(C=i)P(\mathbf{x}|C=i)$$

összefüggés írja le, ahol  $\mathbf{x}$  jelenti az adatpont attribútumait. Folytonos adatok esetén a komponensekre természetesen addódik a többváltozós Gauss-eloszlás, ami az úgynevet **kevert Gauss**- (**mixture of Gaussians**) eloszlás családot adja. A kevert Gauss-eloszlás paraméterei:  $w_i = P(C=i)$  (az egyes komponensek súlya),  $\mu_i$  (az egyes komponensek várható értéke)  $\Sigma_i$  (az egyes komponensek kovarianciája). A 20.8. (b) ábra három Gauss-eloszlás keverékét mutatja, valójában ez volt az (a) ábrán látható adathalmaz forrása.

Ez esetben a nem ellenőrzött osztályozási probléma abban áll, hogy a 20.8. (a) ábrán látható nyers adatokból kell visszaállítani a 20.8. (b) ábrán láthatóhoz hasonló kevert modellt. Nyilvánvaló, hogy ha tudnánk, melyik adatot melyik komponens generálta, akkor könnyű lenne visszaállítanunk az egyes Gauss-komponenseket. Egyszerűen kiválaszthatnánk az egyes komponensekhez tartozó összes adatpontot, majd alkalmazhatnánk a (20.4) egyenletet (tulajdonképpen annak többváltozós verzióját) a Gauss-paramétereknek az adatokhoz való illesztésére. Másrésztől, ha tudnánk az egyes komponensek paramétereit, akkor – legalábbis valószínűségi értelemben – meg tudnánk adni, hogy melyik adatpont melyik eloszláshoz tartozik. A probléma az, hogy sem az adatpontok komponensekhez rendelését, sem a komponensek paramétereit nem ismerjük.



**20.8. ábra.** (a) 500 kétdimenziós adatpont, amely három osztály jelenlétéit sugallja. (b) Egy három-komponensű kevert Gauss-modell, a súlyok (balról jobbra) 0,2, 0,3 és 0,5. Az (a) ábrán látható adatokat ebből a modellből generáltuk. (c) Az adatokból az EM algoritmussal rekonstruált modell

Az EM algoritmus alapötlete ebben az összefüggésben az, hogy úgy tesziink, mintha ismernénk a modell paramétereit, majd kiszámítjuk minden egyes pont minden egyes komponenshez való tartozásának valószínűségét. Ezek után újraillesztjük a komponenseket az adatokhoz: minden egyik komponensem a teljes adathalmazhoz illesztjük, az egyes adatpontokat az adott komponenshez tartozás valószínűségével súlyozva. Az eljárásról a konvergencia elérteig folytatjuk. Lényegében „teljessé tesszük” az adatokat azzal, hogy a rejtett változók – melyik adat melyik komponenshez tartozik – valószínűségi eloszlását kikövetkeztetjük a pillanatnyilag használt modell segítségével. A kevert Gauss-eloszlás esetén a paramétereknek tetszőleges módon kezdeti értéket adunk, majd a következő két lépést ismételjük:

1. E-lépés: Számítsuk ki a  $p_{ij} = P(C = i | \mathbf{x}_j)$  valószínűségeket, amelyek azt fejezik ki, hogy milyen valószínűsggel generálta az  $i$ -edik komponens  $\mathbf{x}_j$ -t. A Bayes-szabály alapján  $p_{ij} = \alpha P(\mathbf{x}_j | C = i) / P(C = i)$ . A  $P(\mathbf{x}_j | C = i)$  tényező az  $\mathbf{x}_j$  valószínűsége az  $i$ -edik Gauss-eloszlásban, míg  $P(C = i)$  az  $i$ -edik Gauss-eloszlás súlya. Használjuk a  $p_i = \sum_j p_{ij}$  definíciót.
2. M-lépés: Számítsuk ki az új átlag, szórás és komponens súly paramétereket a következők szerint:

$$\boldsymbol{\mu}_i \leftarrow \sum_j p_{ij} \mathbf{x}_j / p_i$$

$$\boldsymbol{\Sigma}_i \leftarrow \sum_j p_{ij} (\mathbf{x}_j \mathbf{x}_j^\top / p_i)$$

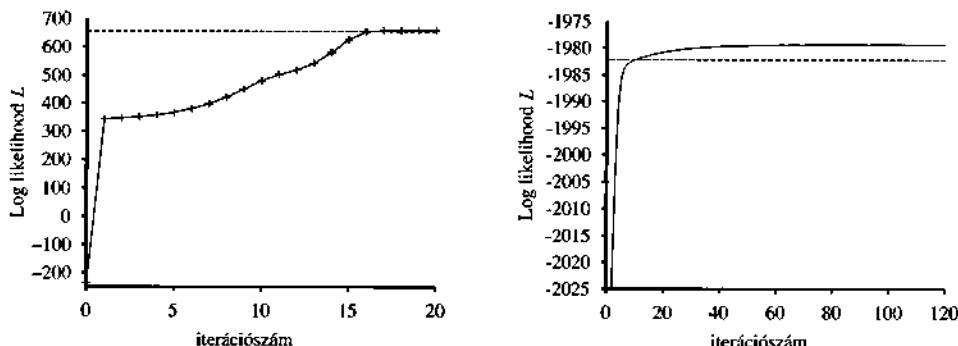
$$w_i \leftarrow p_i$$

Az E-lépés vagy *expectation* (várhatóérték-képzés) lépés úgy fogható fel, mintha a rejtett indikátor változó (*indicator variables*)  $Z_{ij}$  valószínűségének  $p_{ij}$  várható értékét (*expected value*) számítanánk ki. Ha az  $i$ -edik komponens generálta  $\mathbf{x}_j$ -t, akkor  $Z_{ij}$  értéke 1, különben pedig 0. Az M-lépés vagy *maximization* (maximumkeresés) lépés a rejtett változók adott várható értéke mellett megkeresi azokat a paramétereket, amelyek maximálják az adatokra felállított log likelihood függvényt.

A végső modellt – amelyet az EM segítségével a 20.8. (a) ábra adatain tanultunk – a 20.8. (c) ábra mutatja. Látszólag megtörökölhetetlen attól az eredeti modelltől, amellyel az adatokat generáltuk. A 20.9. (a) ábra az adatoknak a pillanatnyi modellből számított log likelihood értékét mutatja az EM algoritmus futása során. Két pontra hívjuk fel a figyelmet. Először is, a végső modellre a log likelihood nemileg meghaladja az arra a valós modellre számított értéket, amelyekkel az adatokat generáltuk. Ez meglepőnek tűnhet, de egyszerűen azt mutatja, hogy az adatokat véletlenszerűen generáltuk, és nem pontosan tükrözik a mögöttes modellt. A második, hogy az EM minden iterációs lépésben növeli az adatok log likelihood értékét. Ez általánosságban is bizonyítható. Továbbá bizonyos feltételek mellett belátható, hogy az EM biztosan eléri a likelihood egy lokális maximumát. (Ritka esetekben elérhet egy nyeregpontba, vagy akár egy lokális minimumba is.) Ebben az értelemben az EM egy gradiensalapú hegymászó algoritmusra emlékeztet, de vegyük észre, hogy nincs „lépésméret” paramétere!

A dolgok nem mennek mindenkor jól, mint ahogyan a 20.9. (a) ábra esetleg sugallja. Előfordulhat, hogy egyik Gauss-komponens úgy összezsugorodik, hogy csak egyetlen





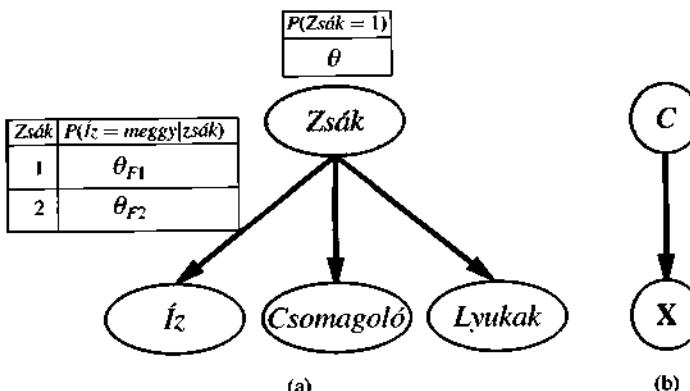
**20.9. ábra.** Az  $L$  log likelihood értéknek – az EM iterációs zámának függvényében való – alakulását bemutató görbék. A vízszintes vonal a helyes modellnek megfelelő log likelihood értéket mutatja. (a) A 20.8. ábrán látható kevert Gauss-modellre vonatkozó görbe. (b) A 20.10. (a) ábrán látható Bayes-hálóra vonatkozó görbe.

adatpontot fed le. Ebben az esetben a varianciája nullához tart, a belőle számított likelihood pedig végtelenhez! Másik probléma lehet, hogy két komponens „összeolvad”, azonos átlagot és varianciát vesz fel, az adatpontokon pedig osztoznak. Az ilyen típusú degenerált lokális maximumok súlyos problémát jelentenek, elsősorban sokdimenziós terekben. Ennek a problémának egyik megoldása lehet megfelelő priorok állítása a modell paramétereire és az EM algoritmus MAP-változatának alkalmazása. Másik lehetőség a komponens újraindítása új paraméterekkel, ha túl kicsivé válik vagy túl közel kerül más komponensekhez. A paraméterek ésszerű kezdeti értékei szintén segítenek a probléma megoldásában.

## Rejtett változókkal felépített Bayes-hálók tanulása

A rejtett változókkal felépített Bayes-hálók tanulásánál ugyanazokat a megközelítéseket alkalmazzuk, amelyek jól működtek a kevert Gauss-eloszlásokra. A 20.10. ábra azt a helyzetet mutatja, amikor két zsák cukorkát összekevertek. A cukrokat három tulajdon-ság írja le, az  $\bar{I}z$  és a *Csomagolás* mellett néhány nél megjelenik egy  $L_{yuk}$  a cukorka közepén, másoknál pedig nincs lyuk.

A cukorkák eloszlását minden egyes zsákban egy **naiv Bayes**-modell írja le; a tulaj-donságok adott zsák esetén függetlenek, de mindegyik tulajdonság feltételes valószínű-ség eloszlása függ a zsaktól. A következő paramétereink vannak:  $\theta$  annak az a priori valószínűsége, hogy a cukorka az 1. zsákból származik;  $\theta_{I1}$ , illetve  $\theta_{I2}$  annak valószínű-sége, hogy a cukorka megyízű, feltéve, hogy az 1., illetve a 2. zsákból származik;  $\theta_{Cs1}$ , illetve  $\theta_{Cs2}$  azokat a valószínűségeket jelölik, hogy a csomagolás piros;  $\theta_{Ly1}$ , illetve  $\theta_{Ly2}$  pedig azokat a valószínűségeket, hogy a cukorka lyukas. Vegyük észre, hogy az egész modell egy kevert modell. (Valójában a kevert Gauss-eloszlásokat is modellezhetjük Bayes-hálóval, ahogy azt a 20.10. (b) ábra mutatja.) Az ábrán a zsák egy rejtett változó, mivel attól a pillanattól, hogy összekevertük a zsákok tartalmát, többet nem tudjuk megmondani egy cukorkáról, hogy melyik zsákból származott. Vissza tudjuk ebben az esetben nyerni a zsákok leírását, ha csupán az összekevert cukorkákat tudjuk megfigyelni?



**20.10. ábra.** (a) A cukorkák kevert modellje. A különböző ízű cukorkák, a csomagolások, valamint a lyukas cukrok aránya a zsáktól függ, amelyet nem figyelünk meg. (b) Egy kevert Gaussra vonatkozó Bayes-háló. Az  $X$  megfigyelhető változó átlaga és kovarienciája a  $C$  komponenstől függ.

Vigyük végig az EM-iterációt erre a problémára. Először nézzük az adatokat. 1000 mintát generáltunk a következő paraméterekkel rendelkező (valódi) modell alapján:

$$\theta = 0,5, \theta_{I1} = \theta_{Cs1} = \theta_{Ly1} = 0,8, \theta_{I2} = \theta_{Cs2} = \theta_{Ly2} = 0,3 \quad (20.7)$$

Azaz a cukorkák egyforma valószínűséggel származhatnak minden zsákból, az első zsák nagyobbrészt meggyízű cukrokat tartalmaz piros csomagolásban, és sok köztük a lyukas, a második inkább nem lyukas citromízűeket, zöld csomagolásban. A lehetséges nyolcféle cukor előfordulási számai a következők:

	$Cs = \text{piros}$		$Cs = \text{zöld}$	
	$Ly = 1$	$Ly = 0$	$Ly = 1$	$Ly = 0$
$I = \text{meggy}$	273	93	104	90
$I = \text{citrom}$	79	100	94	167

A paraméterek alapértékre állításával kezdünk. A könnyű számíthatóság kedvéért a következő értékeket választjuk:<sup>8</sup>

$$\theta^{(0)} = 0,6, \theta_{I1}^{(0)} = \theta_{Cs1}^{(0)} = \theta_{Ly1}^{(0)} = 0,6, \theta_{I2}^{(0)} = \theta_{Cs2}^{(0)} = \theta_{Ly2}^{(0)} = 0,4 \quad (20.8)$$

Először nézzük a  $\theta$  paramétert. A teljesen megfigyelhető esetben ezt közvetlenül az 1. zsákból, illetve 2. zsákból származó cukorkák megfigyelt számaiból becsülhénk. Mivel a zsák rejtegett változó, ehelyett a várható számértékekkel számolunk. A várható cukorkaszám  $\hat{N}(Zsák = 1)$  annak valószínűsége, az összes cukorkára összegezve, hogy a cukorka az 1. zsákból származik:

<sup>8</sup> A gyakorlatban jobb véletlen értékeket választani, hogy elkerüljük a szimmetria által okozott lokális maximumokat.

$$\theta^{(1)} = \hat{N}(Zsák = 1)/N = \sum_{j=1}^N P(Zsák = 1 | Iz_j, csomagoló_j, lyukak_j) / N$$

Ezek a valószínűségek a Bayes-hálóra kifejlesztett tetszőleges következetési algoritmus-sal számíthatók. Naiv Bayes-hálóra, mint amilyen példánkban is szerepel, a következetés „kézzel” elvégezhető, a Bayes-szabály és a feltételes függetlenség felhasználásával:

$$\theta^{(1)} = \frac{1}{N} \sum_{j=1}^N \frac{P(Iz_j | Zsák = 1) P(csomagoló_j | Zsák = 1) P(lukak_j | Zsák = 1) P(Zsák = 1)}{\sum_i P(Iz_j | Zsák = i) P(csomagoló_j | Zsák = i) P(lukak_j | Zsák = i) P(Zsák = i)}$$

(Figyeljük meg, hogy a normalizált konstans szintén a paraméterektől függ.) Alkal-mazva ezt a képletet, mondjuk, a 273 pirosan csomagolt meggyízű cukorkára, amelyekben lyuk van, a következőt kapjuk:

$$\frac{273}{1000} \cdot \frac{\theta_{I1}^{(0)} \theta_{Cs1}^{(0)} \theta_{Ly1}^{(0)}}{\theta_{I1}^{(0)} \theta_{Cs1}^{(0)} \theta_{Ly1}^{(0)} + \theta_{I2}^{(0)} \theta_{Cs2}^{(0)} \theta_{Ly2}^{(0)} (1 - \theta^{(0)})} \approx 0,22797$$

A számok táblázatában található másik hét cukorkafajtával folytatva,  $\theta^{(1)} = 0,6124$ -et kapunk.

Nézzük a további paramétereket, mint pl.  $\theta_{I1}$ -et. A teljesen megfigyelhető esetben ezt közvetlenül a megfigyelt, 1. zsákból származó meggy- és citromízű cukorkák számából becsülnénk. Az 1. zsákból származó meggycukorkák várható száma:

$$\sum_{j: Iz_j = \text{meggy}} P(Zsák = 1 | Iz_j = \text{meggy}, csomagoló_j, lyukak_j)$$

Ezek a valószínűségek ismét bármely Bayes-háló algoritmussal kiszámíthatók. Befejevve ezt az eljárást, megkapjuk az összes paraméter új értékét:

$$\begin{aligned} \theta^{(1)} &= 0,6124, \quad \theta_{I1}^{(1)} = 0,6684, \quad \theta_{Cs1}^{(1)} = 0,6483, \quad \theta_{Ly1}^{(1)} = 0,6558 \\ \theta_{I2}^{(1)} &= 0,3887, \quad \theta_{Cs2}^{(1)} = 0,3817, \quad \theta_{Ly2}^{(1)} = 0,3827 \end{aligned} \quad (20.9)$$

Az adatok log likelihood értéke a kezdeti kb. -2044-ről az első iteráció során kb. -2021-re nő, ahogy a 20.9. (b) ábrán látható. Eszerint a frissítés magát a likelihoodot kb.  $e^{23} \approx 10^{10}$  szorzófaktorral növelte. A tizedik iterációra a megtanult modell jobban illeszkedik, mint a valódi ( $L = -1982,214$ ). Ezek után a javulás nagyon lassúvá válik. Ez nem szokatlan az EM algoritmusnál, ezért sok gyakorlatban használt rendszer a tanulás végső fázisában gradiensalapú algoritmusokkal – mint pl. a Newton–Raphson-módszer (lásd 4. fejezet) – kombinálja az EM algoritmust.

Az ebből a példából levonható általános tanulság az, hogy a rejtett változókkal történő Bayes-háló tanulásparamétereinek frissítése közvetlenül rendelkezésünkre áll az egyes példákon történő következetés eredményeképpen. Ráadásul csak lokális a posteriori valószínűségekre van szükségünk az egyes paraméterekhez. Az általános esetben, amikor az egyes  $X_i$  változók feltételes valószínűségi paramétereit – azaz a  $\theta_{ijk} = P(X_i = X_{ij}|Pa_i = pa_{ik})$  értékeit – tanuljuk, adott szülőcsomópontok esetén, a frissítést a normalizált várható számértékek adják:



$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, Pa_i = pa_{ik}) / \hat{N}(Pa_i = pa_{ik})$$

A várható számértékek a példák összegzésével nyerhetők, kiszámítva a  $P(X_i = x_{ij}, Pa_i = pa_{ik})$  valószínűségeket egy tetszőleges Bayes-háló következtetési algoritmussal. A pontos algoritmusok esetén – beleértve a változó eliminációt is – ezek a valószínűségek minden a következtetés melléktermékeként közvetlenül nyerhetők, a tanuláshoz nincs szükség semmilyen speciális exira számításra. Sőt minden egyes paraméter tanulásához szükséges információ *lokálisan* rendelkezésre áll.

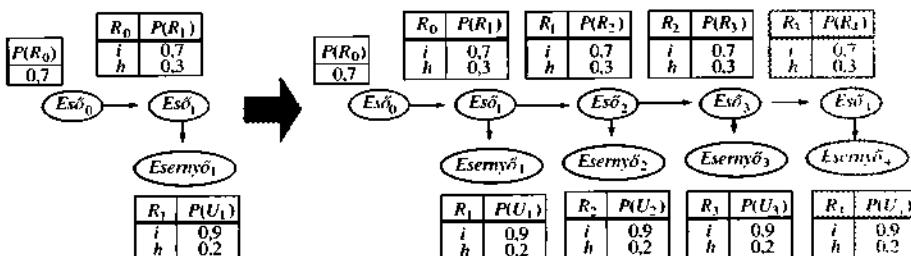
## Rejtett Markov-modellek tanulása

Az EM algoritmus utolsó, itt bemutatott alkalmazása a rejtett Markov-modellek (RMM) (hidden Markov model, HMM) állapotátmenet-valószínűségeinek tanulása. Idézzük fel a 15. fejezetből, hogy a rejtett Markov-modellek olyan dinamikus Bayes-hálóval reprezentálhatók, amelyeknek egyetlen, diszkrét állapotváltozója van, amint ez a 20.11. ábrán is látható. minden egyes adatpont egy véges hosszúságú megfigyelés-sorozatnak felel meg, tehát a megoldandó probléma az, hogy megfigyelési sorozatok halmaza (vagy esetleg egyetlen hosszú sorozat) alapján állapotátmenet-valószínűségeket kell megtanulnunk.

Már kimunkáltuk, hogy mi módon tudunk Bayes-hálókat megtanulni, de van egy nehézség: a Bayes-hálóknál minden egyes paraméter elkülönül, a rejtett Markov-modellben viszont időben *ismétlődnek* az egyes  $\theta_{ijt} = P(X_{t+1} = j | X_t = i)$  állapotátmenet-valószínűségek (annak valószínűsége, hogy a  $t$  időpillanatban az  $i$ -edik állapotból a  $j$ -edikbe jutunk), azaz  $\theta_{ijt} = \theta_{ij}$  minden  $t$ -re. Ahhoz, hogy megbecsüljük az  $i$ -edik állapotból a  $j$ -edik állapotba való átmenet valószínűségét, egyszerűen kiszámítjuk azon esetek várható arányát, amikor a rendszer az  $i$ -edik állapotban van és a  $j$ -edik állapotba megy:

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_t \hat{N}(X_t = i)$$

Ismét tetszőleges RMM következtetési algoritmust használhatunk a várható számértékek kiszámítására. A 15.4. ábrán bemutatott előre-hátra (forward-backward) algoritmus könnyen módosítható úgy, hogy kiszámítsa a szükséges valószínűségeket. Egy fontos szempont, hogy azokra a valószínűségekre van szükségünk, amelyeket inkább simítás-



20.11. ábra. Rejtett Markov-modellt reprezentáló kiterített dinamikus Bayes-háló (a 15.14. ábra megismétlése)

sal (*smoothing*) nyerünk, nem pedig szűréssel (*filtering*): azaz nekünk utólagosan kell értékelnünk a bizonyítékokat az adott állapotámenet bekövetkezési valószínűségének becslésekor. Mint a 15. fejezetben mondta, a gyilkosság bizonyítékát általában a gyilkosság bekövetkezte (azaz az  $i$  állapotból a  $j$  állapotba való átmenetet) után nyerjük.

## Az EM algoritmus általános alakja

Számos példát láttunk az EM algoritmusra. Mindegyik tartalmazza azt a lépést, hogy ki-számítottuk a rejtett változók várható értékét minden példára, majd ezeket a várható értékeket úgy használtuk, mintha megfigyelt értékek lettek volna, és segítségükkel újraszámoltuk a paramétereket. Legyen  $\mathbf{x}$  az összes példában az összes megfigyelt érték, jelölje  $\mathbf{Z}$  az összes példában az összes rejtett változót, és legyen  $\theta$  a valószínűségi modell összes paramétere. Ekkor az EM algoritmus:

$$\theta^{(i+1)} = \operatorname{argmax}_{\theta} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z} | \theta)$$

Az EM algoritmus dióhéjban ez az egyenlet. Az E-lépés az összegzés kiszámítása, ami a „teljessé tett” adatokra, a log likelihoodnak a  $P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)})$  eloszlás szerint számított várható értéke. Ez a valószínűség a rejtett változóknak az adatok felhasználásával nyert a posteriori valószínűsége. Az M-lépés viszont nem más, mint ennek a log likelihoodnak a maximalizálása a paraméterek szerint. Kevert Gauss-jelekre a rejtett változók a  $Z_{ij}$ -k, ahol  $Z_{ij} = 1$ , ha a  $j$ -edik mintát az  $i$ -edik komponens generálta. Bayes-hálóknál az egyes példák nem megfigyelt változó értékei a rejtett változók. RMM esetben az  $i \rightarrow j$  átmenetek a rejtett változók. Ha azonosítottuk a megfelelő rejtett változókat az adott alkalmazásban, akkor az általános formából kiindulva levezethető az EM algoritmus.

Amint megrégettük az általános elgondolást, amin az EM alapul, az összes variáns és javítás könnyen levezethető. Például: sok esetben az E-lépés – a rejtett változók posteriorjainak kiszámítása – kezelhetetlen problémát okoz, mint például a nagy Bayes-hálókban. Kiderült, hogy alkalmazhatunk egy közelítő E-lépést, és még ezzel is hatékony algoritmust kapunk. Egy mintavételi algoritmussal – mint amilyen például az MCMC (lásd 14.5. alfejezet) – a tanulási folyamat nagyon szemléletes: minden egyes, az MCMC által bejárt állapotot (a rejtett és megfigyelt változók konfigurációját) úgy kezelünk, mintha teljes megfigyelés volna. Ennek megfelelően a paraméterek minden egyes MCMC-átmenet után frissíthetők. A közelítő következetés más formái – mint például a variációs és hurkos módszerek – szintén hatékonyak bizonyultak nagyon nagy hálók tanulásánál.

## Bayes-hálóstruktúra tanulása rejtett változók esetén

A 20.2. alfejezetben a Bayes-hálóstruktúra tanulásának problémáját tárgyaltuk teljes adatok esetén. Amikor rejtett változókat is figyelembe veszünk, a dolgok nehezebbé válnak. A legegyszerűbb esetben a rejtett változók a megfigyelésekkel együtt fel vannak sorolva, bár az értékük nem ismert, de a tanuló algoritmus tud a létezésükről, és helyet kell nekik

találjon a hálóstruktúrában. Például egy algoritmus tanulhatja úgy a 20.7. (a) ábrán látható struktúrát, hogy megadjuk neki, hogy egy SzívBetegség nevű (hárommértékű) változót be kell építsen a struktúrába. Ha a tanuló algoritmus nem kapja meg ezt az információt, akkor két lehetőség van: vagy úgy viselkedik, mintha az adatok teljesek volnának (ami arra kényszeríti, hogy a 20.7. (b) sokparaméteres modellt tanulja meg), vagy *kitalál* új rejttet váltoozókat a modell egyszerűsítése érdekében. Ez utóbbi megközelítés megvalósítható úgy, hogy új módosítási lehetőségeket vezetünk be a struktúrakeresésbe: a kapcsolatok módosításán túl az algoritmus hozzáadhat vagy törlhet egy rejttet változót, illetve megváltoztathatja argumentumszámát. Természetesen az algoritmus nem fogja tudni, hogy a bevezetett új változó neve SzívBetegség; hasonlóképpen nem tud értelmes nevet adni az értékeinek. Szerencsére általában a bevezetett új rejttet változó előzetesen már létező változókkal van kapcsolatban, így egy szakember rendszerint meg tudja vizsgálni az új változót is tartalmazó lokális feltételeles eloszlásokat, és meg tudja állapítani a jelentését.

Éppúgy, mint a teljes adatokra vonatkozó esetben, egy tisztán maximum-likelihood struktúratanulás teljesen összekötött hálót eredményez (ráadásul rejttet változók nélkülit), így használnunk kell a komplexitásbüntetés valamelyen formáját. Használhatjuk az MCMC-t is a Bayes-tanulás approximációjára. Például: ismeretlen komponenzzámu kevert Gauss-jelek tanulásakor túlmintavételezhetjük a számot; ekkor a közelítő Gauss-jelek számának közelítő a posteriori eloszlását az MCMC-folyamat mintavételi frekvenciája adja.

Az eddigiekben az általunk vizsgált eljárásnak volt egy külső hurka, amely a struktúrakeresés folyamata, és egy belső hurka, ami egy paraméteres optimalizálási eljárás. A teljes adatokra vonatkozó esetben a belső hurok nagyon gyors – csupán ki kell nyerni az adathalmazból a feltételeles gyakoriságokat. Ha rejttet változók is vannak, akkor a belső hurok az EM algoritmussal történő számos iterációt vagy egy gradiensalapú algoritmust is tartalmazhat. minden iterációs ciklusban ki kell számítani a Bayes-háló a posteriori eloszlásait, ami önmagában NP-teljes probléma. Ez a megközelítés bonyolult modellek tanulása esetén napjainkig nem bizonyult a gyakorlatban is alkalmazhatónak. Egy lehetséges előrelépés a **strukturális EM (structural EM)**, ami nagyon hasonlóan működik, mint a normál (parametrikus) EM, leszámítva azt, hogy nemcsak a paraméreket, hanem a struktúrát is frissíti. Éppúgy, mint ahogy a normál EM algoritmus arra használja az aktuális paramétereket, hogy a várható számértékeket meghatározza az E-lépés során, majd alkalmazza ezeket a számértékeket az M-lépésben új paraméterek választására, a strukturális EM az aktuális struktúrát használja a várható számértékek meghatározására, majd arra alkalmazza ezeket a számértékeket az M-lépésben, hogy a potenciális új struktúrák likelihoodját meghatározza. (Ellentétben a külső hurok/belső hurok módszerrel, amelynél minden potenciális struktúrára kiszámítjuk a várható számértékeket.) Ezen az úton a strukturális EM számos strukturális változtatást tud végrehajtani a hálón anélkül, hogy egyszer is újraszámolná a várható számértékeket. Így a módszer képessé válik nemtriviális Bayes-hálóstruktúrák tanulására. Mindamellett sok munkát kell még befektetni, míg azt mondhatjuk majd, hogy a struktúratanulás problémája megoldott.

## 20.4. PÉLDÁNYALAPÚ TANULÁS

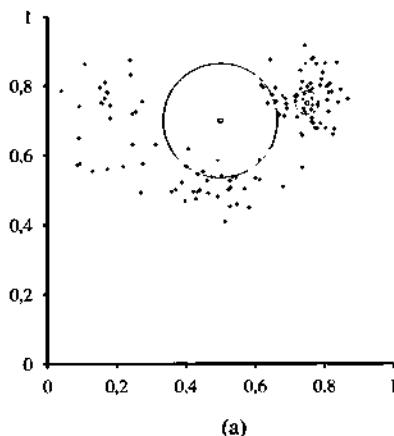
Eddigiekben a statisztikai tanulás tárgyalásában arra koncentráltunk, hogy valószínűségi modellek egy *behatárolt* családjának paramétereit egy *struktúrájában nem kötött* adathalmazra illesszük. Például a kevert Gaussokat használó, nem ellenőrzött tanulás feltételezi, hogy az adatok megmagyarázhatók *rögzített* számú *Gauss-eloszlás összegeként*. Az ilyen módszereket **paraméteres tanulásnak** (*parametric learning*) nevezzük. A paraméteres tanuló eljárások gyakran nagyon egyszerűek és hatékonyak, de egy meghatározott behatárolt modellcsalád feltételezése sokszor túlzott egyszerűsítése annak, ami az adatokat szolgáltató valós világban történik. Nos, az természetesen igaz, hogy ha nagyon kevés adatunk van, akkor nem reménykedhetünk egy bonyolult és részletes modell megtanulásában, de butaságnak tűnik a hipotézis komplexitását akkor is rögzítve tartani, amikor az adathalmaz nagyon nagyra nő.

A paraméteres módszerekkel ellentétben a **nemparaméteres tanulási** (*nonparametric learning*) módszerek lehetővé teszik, hogy a hipotézis bonyolultsága együtt növekedjék az adatokkal. Minél több adatunk van, annál körmönfontabb lehet a hipotézis. Két, nagyon egyszerű nemparaméteres **példányalapú tanulást** használó módszercsaládot fogunk vizsgálni, amelyeket azért nevezünk így, mert közvetlenül a tanító példányokból hoznak létre hipotéziseket.

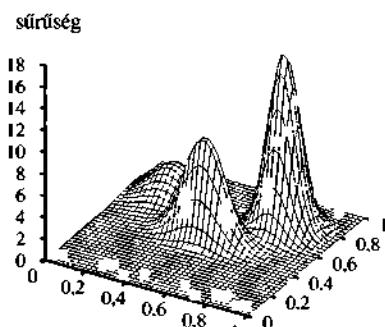
### Legközelebbi-szomszéd modellek

A **legközelebbi-szomszéd** (*nearest-neighbor*) modellek alapötlete az, hogy bármely egyedi  $x$  bemeneti pont tulajdonságai valószínűleg hasonlók az  $x$  pont környezetébe eső pontok tulajdonságaihoz. Például ha **sűrűségbecslést** (*density estimation*) kívánunk végezni – azaz meg akarjuk becsülni egy ismeretlen valószínűség sűrűségsűrűség-értekét  $x$ -ben –, akkor egyszerűen mérhetjük azt, hogy  $x$  környezete milyen sűrű van beszórva pontokkal. Ez nagyon egyszerűen hangzik, amíg rá nem jövünk, hogy meg kell határoznunk, mit is értünk „szomszédság” alatt. Ha a szomszédság túl kicsi, akkor egyáltalán nem fog pontokat tartalmazni. Ha túl nagy, akkor esetleg az összes adatpontot tartalmazni fogja, és olyan sűrűséget eredményez, ami mindenütt ugyanakkora. Egy megoldási lehetőség, ha a szomszédságot úgy definiáljuk, hogy legyen éppen elég tág ahoz, hogy  $k$  pontot tartalmazzon, ahol  $k$  elég nagy ahoz, hogy ésszerű becslést végezhessünk. Rögzített  $k$  esetén a szomszédság mérete változó, ahol ritkán vannak az adatok, ott a szomszédság tág térrész, ahol az adatok sűrűn helyezkednek el, ott a szomszédság szűk. A 20.12. (a) ábra egy példát mutat: két dimenzióban szétszórt adatpontokat. A 20.13. ábra mutatja az ezen adatokra elvégzett  $k$ -legközelebbi-szomszéd sűrűségbecslést  $k = 3, 10$  és  $40$  esetére.  $k = 3$  esetében a sűrűségbecslést minden egyes pontban csak 3 szomszédos pontra alapozzuk, és ezért erősen változó eredményt kapunk.  $k = 10$  esetén a 20.12. (b) ábrán látható valódi sűrűség jó becslését kapjuk.  $k = 40$  esetén a szomszédság túl tágággá válik, és az adatok struktúráját mindenestől elveszti. Gyakorlatban – kis dimenziószám esetén – egy 5 és 10 közé eső  $k$  a legtöbb esetben jó eredményt ad. Keresztpeliszálásával is jó értéket kaphatunk a  $k$ -ra.

Ahoz, hogy egy kérdéses pontban a legközelebbi szomszédokat meg tudjuk határozní, egy távolságmetrikára –  $D(x_1, x_2)$ -re – van szükségünk. A 20.12. ábra kétdimenziós

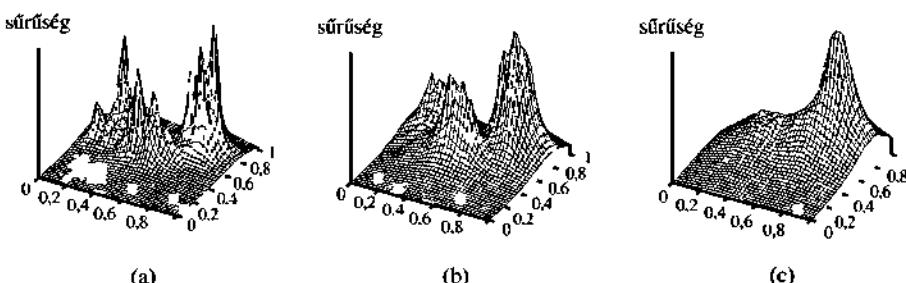


(a)



(b)

**20.12. ábra.** (a) A 20.8. (a) ábra adatainak 128 pontos részhalmaza, két kiválasztott ponttal, és 10 legközelebbi szomszédjukkal. (b) Az adatok előállításánál használt kevert Gauss-eloszlás 3D ábrája.



**20.13. ábra.** A 20.12. (a) ábrán látható adatokra végzett sűrűségbecsülés,  $k = 3$  (a);  $k = 10$  (b) és  $k = 40$  (c) esetén

példájában euklideszi távolságot használtunk. Ez nem felel meg, ha a tér minden dimenziójában valami másat mérünk – például magasságot és súlyt –, mivel egy dimenzió skálájának megváltoztatása megváltoztatja a legközelebbi szomszédok halmazát. Az egyik lehetőség, hogy minden egyes dimenzió skáláját normalizáljuk. Ehhez először megmérjük az összes tulajdonság varianciáját a teljes adathalmazon, majd a tulajdonságértéket úgy fejezzük ki, hogy megadjuk: a tulajdonságérték az adott tulajdonság varianciájának hányoszora. (Ez a Mahalanobis-távolság [Mahalanobis distance] – amely a tulajdonságok kovarianciamátrixát veszi figyelembe – speciális esete.) Végül diszkrét esetben alkalmazhatjuk a Hamming-távolságot (Hamming distance), amely úgy definiálja  $D(x_1, x_2)$ -t, mint azon tulajdonságok számát, amelyben  $x_1$  és  $x_2$  eltér.

A 20.13. ábrán láthatóhoz hasonló sűrűségbecsülések a bemeneti tér feletti együttes eloszlásokat határoznak meg. A Bayes-hálóktól eltérően a példányalapú reprezentációk nem tartalmaznak rejtett változókat, ami azt jelenti, hogy nem alkalmazhatunk nem ellenőrzött osztályozást, mint ahogy a kevert Gauss-modellnél tettük. Továbbra is alkalmazhatjuk a kívánt  $y$  értéknek az  $x$  bemeneti tulajdonságok alapján történő jóslá-

sára a sűrűségbecslést, ha kiszámítjuk  $P(y|x) = P(y, x)/P(x)$ -et, feltéve, hogy a tanító adatok a kívánt értékre nézve is tartalmaznak értékeket.

A legközelebbi-szomszéd tanulási algoritmus használható közvetlen ellenőrzött tanulásra is. Ha adott egy tesztpélda  $x$  bemeneti értékkal, akkor az  $y = h(x)$  az  $x$   $k$ -legközelebbi-szomszédjának értékéiből nyerhető. Diszkrét esetben többségi szavazzal kaphatunk egyetlen jóslt értéket. Folytonos esetben átlagolhatjuk a  $k$  darab értéket, vagy lokális lineáris regressziót végezhetünk a  $k$  pontra illesztve, ez utóbbi esetben a jóslást a kialakuló hipersík  $x$ -beli értéke adja.

A  $k$ -legközelebbi-szomszéd tanulási algoritmus nagyon egyszerűen megvalósítható, kevés hangolást igényel, és gyakran elég jól működik. Jó dolog, ha először egy új tanulási problémán próbáljuk ki. Mindazonáltal nagy adathalmazokban hatékony mechanizmust kell találnunk arra, hogy a kérdéses  $x$  pont legközelebbi szomszédjait megtaláljuk, hiszen túl hosszú lenne minden pontra kiszámítani a távolságot. Egy sor szellemes, a tanító adatok előfeldolgozásán alapuló módszert javasoltak ennek a lépésnek a hatékonytájékoztatására. Szerencsétlen módon a legtöbb ilyen módszer nem viselkedik jól a tér dimenziójának növekedésével (azaz a tulajdonságok számának növekedésével).

A sokdimenziós terek még egy további problémát jelentenek, nevezetesen azt, hogy egy ilyen téren a legközelebbi szomszédok rendszerint nagyon messze vannak. Vagyunk egy  $N$  elemű,  $d$  dimenziós egységekkában elhelyezkedő adathalmazt, és tegyük fel, hogy a szomszédságok  $b$  oldalú hiperkockák, amelyek térfogata  $b^d$ . (Ugyanez a megfontolás minden hipergömbök feltételezése esetén is, de a hipergömb térfogatképlete bonyolultabb.) Ahhoz, hogy  $k$  pontot tartalmazzon, az átlagos szomszédság a teljes térfogat – amit egységegyinek tekintünk –  $k/N$ -ed részét kell elfoglalnia. Ennek megfelelően  $b^d = k/N$ , vagyis  $b = (k/N)^{1/d}$ . Eddig jó. Legyen most  $d = 100$ ,  $k = 10$  és  $N = 1\,000\,000$  értékű. Akkor  $b \approx 0,89$  – azaz a szomszédság majdnem az egész bemeneti teret elfoglalja! Ez arra utal, hogy sokdimenziós esetben a legközelebbi-szomszéd alapú módszerek nem megbízhatók. Alacsony dimenziószám esetén nincs probléma,  $d = 2$  esetén  $b = 0,003$ .

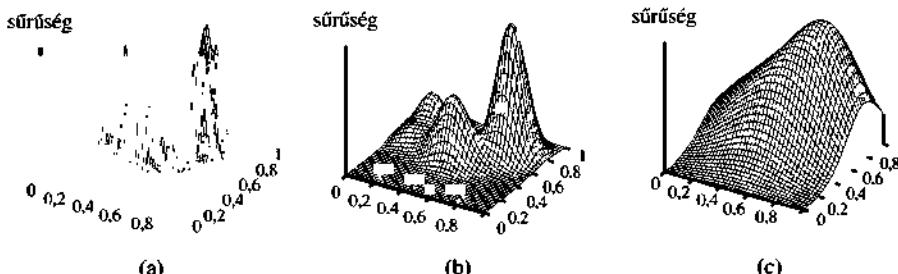
## Kernelmódszerek

A **kernelmodellben** (**kernel model**) úgy tekintünk minden tanító példányra, mintha egy kis saját sűrűségtüggvényt – **kernelfüggvény** (**kernel function**) – generálna. Az eredő sűrűségtüggvény becslés nem más, mint a kis kernelfüggvények súlyozott összege. Egy  $x_i$  tanító példány egy  $K(x, x_i)$  kernelfüggvényt generál, amely a tér minden  $x$  pontjához egy valószínűséget rendel. Így a sűrűségbecslés:

$$P(x) = \frac{1}{N} \sum_{i=1}^N K(x, x_i)$$

Normális esetben a kernelfüggvény csak az  $x$  és  $x_i$  közötti  $D(x, x_i)$  távolságtól függ. A legnépszerűbb kernelfüggvény (természetesen) a Gauss-függvény. Az egyszerűség kedvéért egy gömbszimmetrikus Gauss-függvényt feltételezzük, amelynek minden tengely mentén  $w$  a szórása, azaz:

$$K(x, x_i) = \frac{1}{(w^2 \sqrt{2\pi})^d} e^{-\frac{D(x, x_i)^2}{2w^2}}$$



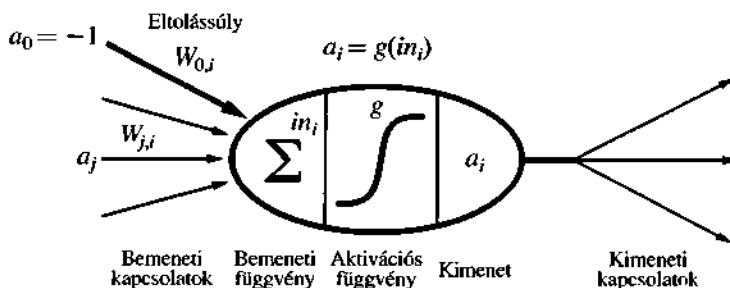
20.14. ábra. A 20.12. (a) ábrán látható adatok sűrűségbecslése Gauss-kernelekkel,  $w = 0,02$  (a);  $w = 0,07$  (b) és  $w = 0,20$  (c) esetén.

ahol  $d$  az  $x$  dimenziószáma. Hátra van még a megfelelő  $w$  érték választásának problémája. Éppúgy, mint az előzőökben egy túl kis szomszédság nagyon hepehupássá teszi a becslést – mint a 20.14. (a) ábrán látható. A (b) ábrán látható, hogy egy közepes  $w$  érték nagyon jó becslést ad. A (c) ábrán bemutattuk, hogy a túl nagy szomszédság eredményeképp teljesen elveszítjük a struktúrát. Jó  $w$  értéket választunk keresztvalidációval.

Ellenőrzött tanulást kernelekkel úgy végezhetünk, hogy a tanító példányokból származtatott összes jóslás *súlyozott* kombinációját vesszük. (Összehasonlíta a  $k$ -legközelebbi-szomszéd módszerrel, ott a  $k$ -legközelebbi-példány súlyozatlan kombinációját használjuk.) Az  $i$ -edik példánynak a kérdéses  $x$  pontra vett súlyát a  $K(x, x_i)$  kernelfüggvényérték adja. Diszkrét jóslás esetén súlyozott szavazást vehetünk, folytonos esetben súlyozott átlagot vagy súlyozott lineáris regressziót. Figyeljük meg, hogy a kernelekkel végzett jóslás azt igényli, hogy az összes tanító példány figyelembe kell vennünk. A kernelek összekombinálhatók a legközelebbi-szomszéd indexelési sémaival azért, hogy csupán a szomszédos példányok alapján előállított súlyozott jóslást alkalmazzuk.

## 20.5. NEURÁLIS HÁLÓK

A **neuron** egy olyan agysejt, amelynek alapfeladata elektromos jelek összegyűjtése, feldolgozása és szetterjesztése. A 1.2. oldalon az 1.2. ábra egy tipikus neuron sematikus rajzát mutatja. Azt gondoljuk, hogy az agy információfeldolgozó kapacitása elsősorban ilyen neuronok *hálózatából* alakul ki. Ezért a korai MI néhány kutatása mesterséges **neurális hálók** (*neural networks*) létrehozására irányult. (A terület más, szintén használt elnevezései: **konnektionizmus** [*connectionism*], **párhuzamos elosztott feldolgozás** [*parallel distributed processing*] és **neurális számítástechnika** [*neural computation*].) A 20.15. ábra a neuron egyszerű matematikai modelljét mutatja, ahogy McCulloch és Pitts (McCulloch és Pitts, 1943) megalkották. Elnagyolva az mondható, hogy a neuron akkor „tüzel”, amikor a bemeneti értékek súlyozott összege meghalad egy küszöböt. 1943 óta sokkal részletesebb és valósághűbb modellek alkottak minden a neuronra, minden az agy nagyobb rendszereire, ez vezetett a **számítógépes idegháló-modellezés** (**computational neuroscience**) modern tudományterületének megjelenéséhez. Másrészről az MI és a statisztika kutatóinak érdeklődését felkelgették a neurális hálózatok absztraktabb tulajdonságai, mint például az elosztott számítás elvégzésére, a bemeneti



**20.15. ábra.** A neuron egyszerű matematikai modellje. Az egység kimeneti aktivációja  $a_i = g\left(\sum_{j=0}^n W_{j,i} a_j\right)$ , ahol  $a_j$  a  $j$ -edik egység kimeneti aktivációja és  $W_{j,i}$  a  $j$ -től  $i$ -ig vezető összeköttetés súlya.

zajjal szembeni érzéketlenségre és a tanulásra való képesség. Bár ma már tudjuk, hogy más rendszerek, például a Bayes-hálók is rendelkeznek ezekkel a tulajdonságokkal, de a neurális háló maradt a tanuló rendszerek egyik leghatékonyabb és legnépszerűbb formája, ezért megéri külön tárgyalni.

## A neurális háló egységei

A neurális hálók irányított kapcsolatokkal (**link**) összekötött csomópontokból vagy egységekből (**unit**) állnak. A  $j$ -edik egységtől az  $i$ -edik felé vezető kapcsolat hivatott az  $a_i$  aktivációtól  $j$ -től az  $i$ -ig terjeszteni. minden egyes kapcsolat rendelkezik egy hozzá azszociált  $W_{j,i}$  numerikus súlytal (weight), ami meghatározza a kapcsolat erősséget és előjelét. minden egyes  $i$  egység először a bemeneteinek egy súlyozott összegét számítja ki:

$$in_i = \sum_{j=0}^n W_{j,i} a_j$$

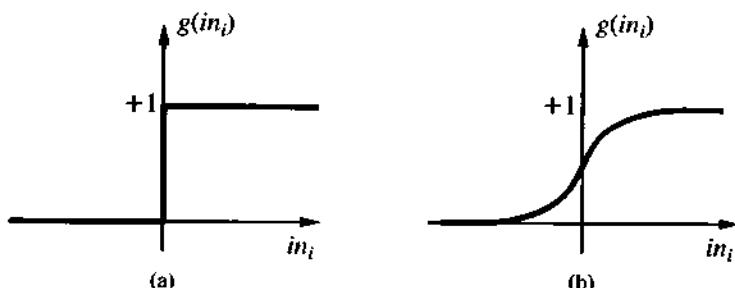
A kimenetét úgy kapja, hogy ezek után egy **aktivációs függvényt** (activation function) alkalmaz a kapott összegre:

$$a_i = g(in_i) = g\left(\sum_{j=0}^n W_{j,i} a_j\right) \quad (20.10)$$

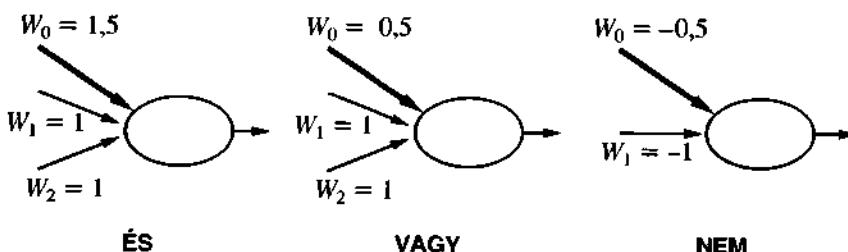
Figyeljük meg, hogy használtunk egy **eltolássúlyt** (bias weight)  $W_{0,i}$ -t, amelyet egy rögzített értékű  $a_0 = -1$  bemenetre kapcsolunk. Rövidesen megmagyarázzuk, hogy mi a jelentősége.

Az aktivációs függvénytel szemben két elvárásunk van. Először az, hogy az egység legyen „aktív” (+1 körüli kimenet), ha a „helyes” bemeneteket kapja, és „inaktív” (0 körüli kimenet), ha „rossz” bemeneteket kap. Másodszor az, hogy az aktiváció legyen **nemlineáris**, különben az egész neurális háló egy egyszerű lineáris függvényé fajul (lásd 20.17. feladat). A 20.16. ábra kétféle aktivációs függvényt mutat be: a **küsziob-függvényt** (threshold function), illetve a **szigmoid függvényt** (sigmoid function) (mely utóbbit **logisztikus függvényként** [logistic function] is ismert). A sigmoid függvény előnye, hogy differenciálható, ami – mint később látni fogjuk – fontos a súlytanulási algoritmus szempontjából. Vegyük észre, hogy minden függvénynek van egy

küszöbpontja (akár kemény, akár lágy) a nullánál; az eltolássúly állítja be az egység *aktuális* kiüszöbpontját. Ez azt jelenti, hogy az egység akkor aktiválódik, ha a „valódi” beámenetek súlyozott összege  $\sum_{j=1}^n W_{j,i} a_j$  meghaladja  $W_{0,i}$ -t.



20.16. ábra. (a) A küszöb aktivációs függvény, amely 1-ét ad a kimenetben, ha a bemenet pozitív, különben pedig 0-t. (Néha az előjelfüggvényt is használják ehelyett, amely  $\pm 1$ -et ad a bemenet előjelétől függően.) (b) A szigmoid függvény  $1/(1+e^{-x})$ .



20.17. ábra. Megfelelő bemeneti és eltolássúlyokkal rendelkező, küszöbaktivációjú egységek képesek logikai kapuként működni

Némi fogalmunk alakulhat ki az egyes egységek működéséről, ha összehasonlítjuk őket a logikai kapukkal. Az egyes egységek tervezésének eredeti motivációi között (McCulloch és Pitts, 1943) szerepelt az, hogy képesek az alapvető logikai függvények reprezentálására. A 20.17. ábra bemutatja, hogy az ÉS, VAGY és NEM logikai függvények hogyan reprezentálhatók egy megfelelő súlyokkal rendelkező küszöbegység segítségével. Ez azért fontos, mert azt jelenti, hogy ezen egységek felhasználásával tetszőleges logikai függvény kiszámítására tudunk hálózatot építeni.

## Hálóstruktúrák

A neurális hálóstruktúrák két fő csoportja: a **hurokmentes** vagy **előrecsatolt hálók** (**feed-forward network**) és a visszacsatolt vagy **rekurrens hálók** (**recurrent network**). Az előrecsatolt háló a pillanatnyi bemenet függvényét reprezentálja, azaz nincs semmilyen más belső állapota, csak maguk a súlyok. A rekurrens háló viszont a kiemelteit visszacsatolja a bemeneteire. Ez azt jelenti, hogy a háló aktivációs szintjei

dinamikus rendszert alkotnak, elérhetnek stabil állapotot, de mutathatnak oszcillációt, sőt kaotikus viselkedést is. Ezenfelül a háló egy adott bemenetre adott válasza a kezdeti állapotától függ, amely a korábbi bemenetektől függhet. Ennél fogva a rekurrens hálók (ellentétben az előre csatolt hálókkal) rövid távú memóriát is biztosíthatnak. Ezáltal érdekesebbé válnak mint agymodellek, de egyben nehezebben is érhetők. Ez a rész az előre csatolt hálókra koncentrál, a fejezet végén néhány hivatkozást adunk, segítve a rekurrens hálók további tanulmányozását.

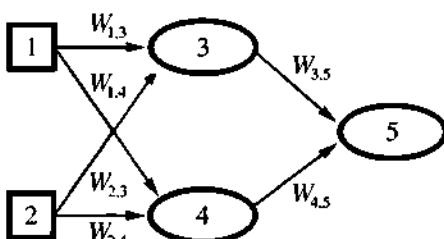
Nézzük meg közelebbről azt az állítást, hogy az előre csatolt háló a bemeneteinek függvényét reprezentálja. Vizsgáljuk a 20.18. ábra egyszerű hálózatát, amelynek két bemeneti egysége, két **rejtett egysége (hidden unit)** és egy kimeneti egysége van. (Az egyszerűség kedvéért ebben a példában elhagytuk az eltolásegységeket.) Adott  $\mathbf{x} = (x_1, x_2)$  bemeneti vektor esetén a bemeneti egységek aktivációja  $(a_1, a_2) = (x_1, x_2)$ , és a hálózat a következő számítást végez:

$$\begin{aligned} a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\ &= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2)) \end{aligned} \quad (20.11)$$

Azaz kifejezve a rejtett egységek kimenetét, mint a saját bemeneteik függvényét, megmutattuk, hogy az egész háló  $a_5$  végső kimenete a háló bemeneteinek függvénye. Továbbá azt látjuk, hogy a háló súlyai ennek a függvénynek a paramétereiként szolgálnak; ha  $\mathbf{W}$ -vel jelöljük a paramétereiket, akkor a háló a  $h_{\mathbf{W}}(\mathbf{x})$  függvényt számítja ki. Ha változtatjuk a súlyokat, akkor változik a háló által reprezentált függvény. Ez a módja a neurális hálók tanulásának.

A neurális hálót osztályozásra vagy regresszióra használhatjuk. Ha folytonos kimenete van a hálónak (pl. szigmoid egységekkel), akkor logikai osztályozás esetén hagyományosan egy kimeneti egységet használunk, és ha ennek aktivációs értéke 0,5 feletti, azt az egyik, ha 0,5 alatti, akkor a másik osztályba tartozásként interpretáljuk. Egy  $k$  osztályos osztályozási feladatról feloszthatjuk az egyetlen kimeneti egység értéktartományát  $k$  részre, de megszokottabb, hogy ehelyett  $k$  elkülönült kimeneti egységet használunk, ahol mindeniknek az aktivációs értéke a bemenet adott osztályba tartozásának valószínűségét reprezentálja.

Az előre csatolt hálókat rendszerint **rétegekbe (layer)** szervezzük oly módon, hogy minden egyes egység csak a közvetlenül megelőző réteg egységeitől kap bemeneti jelet. A következő két alfejezetben egyrészt az egyrétegű hálózatokkal foglalkozunk, amelyeknek nincsenek rejtett egységei, másrészt a többrétegű hálókkal, amelyek egy vagy több rejtett réteggel rendelkeznek.



20.18. ábra. Egy nagyon egyszerű, két bemeneti egységgel, egyetlen – két egységből álló – rejtett réteggel és egy kimeneti egységgel rendelkező háló

## Egyrétegű előrecsatolt neurális hálók (perceptronok)

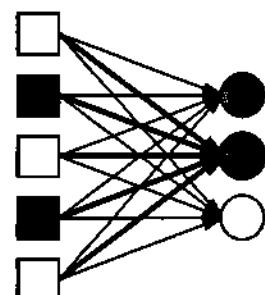
Azt a hálót, amelyben az összes bemenet közvetlenül a kimenetekre kapcsolódik egyrétegű **neurális hálónak** (single layer neural network) vagy **perceptron** (perceptron) hálónak nevezünk. Mivel mindenki kimeneti egység független a többiből – mindenki súly csak egyetlen kimenetre van hatással – vizsgálatainkat korlátozhatjuk az egy-kimenetű perceptronra, mint azt a 20.19. (a) ábra magyarázza.

Kezdjük annak a hipotézistérnek a tanulmányozásával, amelyet egyetlen perceptron reprezentálhat. Küszöbaktivációs függvény esetén úgy tekinthetjük a perceptron, mint ami logikai függvényt reprezentál. Az elemi logikai függvényeken (ÉS, VAGY és NEM, lásd 20.17. ábra) túl a perceptron képes néhány egészen „bonyolult” logikai függvényt is nagyon tömören reprezentálni. Például a többségfüggvényt (**majority function**), amely csak akkor ad ki 1-et, ha  $n$  bemeneteinek több mint fele 1, egy olyan perceptron reprezentál, amelynek minden stílya  $W_j = 1$  és a küszöb  $W_0 = n/2$ . Egy döntési fának  $O(2^n)$  csomópontra lenne szüksége ennek a függvénynek a reprezentálásához.

Sajnos számos olyan logikai függvény van, amelyet a küszöbfüggvényt használó perceptron nem tud reprezentálni. A (20.10) egyenletet vizsgálva látjuk, hogy a küszöbérték-perceptron akkor és csak akkor ad 1-et, ha bemeneteinek (beleértve az eltolás-bemenetet is) súlyozott összege pozitív:

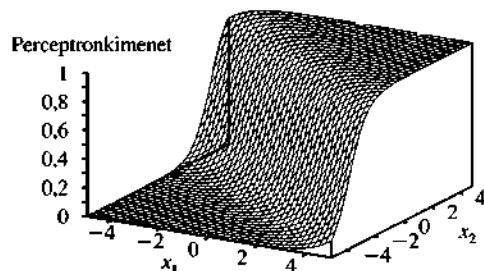
$$\sum_{j=0}^n W_j x_j > 0 \text{ vagy } \mathbf{W} \cdot \mathbf{x} > 0$$

A  $\mathbf{W} \cdot \mathbf{x} = 0$  egyenlet egy hipersíket határoz meg a bemeneti térből, tehát a perceptron akkor és csak akkor ad 1-et, ha a bemenet ennek a hipersíknak az egyik oldalán van. Ezért a küszöbperceptron **lineáris szeparátornak** (**linear separator**) is nevezik. A 20.20. (a) és (b) ábra mutatja a szeparáló hipersíket (ami két dimenzióban egy egyenes) a kétbemenetű ÉS, illetve VAGY függvények perceptronreprezentációja



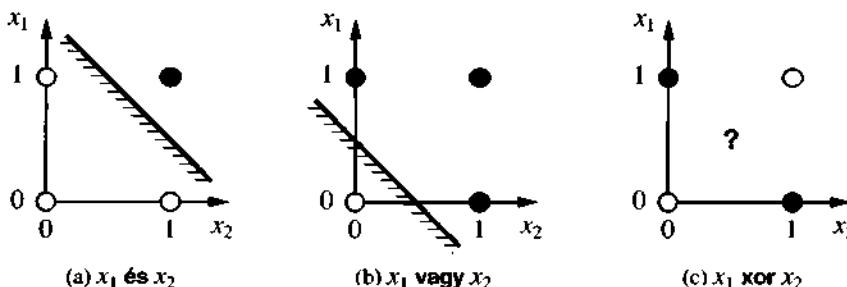
Bemeneti  
egységek       $W_{ij}$       Kimeneti  
egységek

(a)



(b)

**20.19. ábra.** (a) Egy perceptronhálózat három kimeneti egységből, amelyeknek öt közös bemenete van. Ha kiválasztunk egy kimeneti egységet (mondjuk a másodikat, vastag vonallal kiemelv), akkor azt látjuk, hogy bemeneti összeköttetései nincsenek semmilyen hatással a többi kimeneti egységre. (b) Egy kétbemenetű szigmoid aktivációs függvényű perceptronegység kimenetének ábrázolása.



**20.20. ábra.** Lineáris szeparálhatóság kúszöbperceptronok esetén. Fekete pötty jelzi a bemeneti tér olyan pontjait, amelyekre a függvény értéke 1, fehér pötty pedig az olyan pontokat, amelyre 0 ez az érték. A perceptronkimenet az egyenes nem árnyékolt oldalán 1. A (c) esetben nincs olyan egyenes, amely jól osztályozná a bemeneteket.

esetén. Fekete pötty jelzi a bemeneti tér olyan pontjait, amelyekre a függvény értéke 1, fehér pötty pedig az olyan pontokat, amelyre 0 ez az érték. A perceptron azért képes reprezentálni ezt a függvényt, mert létezik olyan egyenes, ami az összes fehér pontot az összes feketétől elválasztja. Az ilyen függvényeket **lineárisan szeparálhatónak (linearly separable)** nevezik. A 20.20. (c) ábra egy olyan függvényre mutat példát, amely *nem* szeparálható lineárisan – ez az XOR függvény. Nyilvánvalóan nincs lehetőség arra, hogy egy kúszöbperceptron megtanulja ezt a függvényt. Általánosságban elmondható, hogy *egy kúszöbperceptron csak lineárisan szeparálható függvények reprezentációjára* képes. Ez a függvényeknek csak kis töredékét jelenti; a 20.14. feladat azt kéri, hogy fejezze ki számszerűen, mekkora is ez a töredék. A szigmoidal felépített perceptronok hasonlóképpen korlátozott képességek abban az értelemben, hogy csupán „lág” lineáris szeparátorokat reprezentálnak. (Lásd 20.19. (b) ábra.)

Korlátozott kifejezőképessége ellenére a kúszöbperceptronoknak vannak előnyei is. Különösen is fontos, hogy *létezik olyan egyszerű tanuló algoritmus, amely a kúszöbperceptron tetszőleges lineárisan szeparálható adathalmazra képes illeszteni*. Mégsem mutatjuk most be ezt, inkább *levezetünk* egy ehhez közel álló tanuló algoritmust a sigmoid perceptronokra.

Ennek az algoritmusnak az az alapgondolata (valójában a neuronháló tanuló algoritmusok legtöbbjének is ez), hogy a tanító mintahalmazon mért hiba valamelyen mértékének minimalizálása érdekében módosítjuk a neuronháló súlyait. Tehát a tanulást formálisan a **súlyterben (weight space)** végzett optimalizálási keresésként fogalmazzuk meg.<sup>9</sup> A hiba „klasszikus” mértéke a **négyzetes hibaösszeg**, amit a lineáris regresszióról a 826–827. oldalon is használtunk. Egyetlen,  $\mathbf{x}$  bemenettel és  $y$  kívánt kimenettel megadott példa négyzetes hibája a következőképpen írható fel:

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_{\mathbf{W}}(\mathbf{x}))^2$$

ahol  $h_{\mathbf{W}}(\mathbf{x})$  a perceptron kimeneti értéke.



<sup>9</sup> A folytonos terekben használható általános optimalizálási technikákat lásd a 4.4. alfejezetben.

A négyzetes hiba csökkentésére gradiensalapú optimalizálási eljárást használhatunk, ehhez az  $E$  minden egyes súlyra vonatkozó parciális deriváltját meg kell határoznunk.

Tehát:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\ &= Err \times \frac{\partial}{\partial W_j} \left( y - g \left( \sum_{j=0}^n W_j x_j \right) \right) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

ahol  $g'$  az aktivációs függvény deriváltja.<sup>10</sup> Amikor a gradiensalapú algoritmusban csökkenteni akarjuk  $E$ -t, a súlyfrissítés összefüggése a következő:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j \quad (20.12)$$

ahol  $\alpha$  a bátorsági faktor (vagy tanulási faktor, learning rate). Intuitíve ez nagyon ésszerű. Ha a hiba  $Err = y - h_W(x)$  pozitív, akkor a háló kimenete túl kicsi, tehát a pozitív bemenetekhez tartozó súlyokat növeljük, a negatívakhoz tartozókat pedig csökkentjük. Ha a hiba negatív, akkor éppen ennek ellenkezője történik.<sup>11</sup>

A teljes algoritmus a 20.21. ábrán látható. Egyesével sorban végigfuttatja a mintákat a hálón, és minden egyes példa után a hiba csökkentése érdekében kissé módosítja a súlyokat. A mintahalmaz egyszeri végigfuttatását epochnak (epoch) nevezzük. Az epochokat addig ismétljük, amíg valamilyen leállási feltétel nem teljesül – tipikus, hogy akkor állunk le, amikor a súlyváltozások már nagyon kicsivé válnak. Más módszerek esetén eredő gradienst számítunk az egész tanító halmazra, egyszerűen összeadva az egyes példák nál a (20.12) származó gradienseket, és az eredő gradiens alapján frissítjük a súlyokat. A sztochasztikus gradiens (stochastic gradient) módszer nem ciklikusan veszi a tanító halmaz mintáit, hanem inkább véletlenszerűen választ mintákat a tanító halmazból.

A 20.22. ábra két különböző problémára bemutatja a perceptron tanulási görbüjét. A bal oldali ábrán a 11 logikai bemenetre vonatkozó többségfüggvény (tehát a kimenet akkor 1, ha 6 vagy több bemenet 1) tanulási görbéje látható. Várakozásunknak megfelelően a perceptron meglehetősen gyorsan tanulja a függvényt, mivel a többségfüggvény lineárisan szeparálható. Másrészt viszont a döntési fa tanuló nem nagyon halad a tanulással, mert a többségfüggvényt nagyon nehéz (bár nem lehetetlen) döntési fában reprezentálni. A jobb oldali ábrán az étterem példa látható. A probléma megoldása könnyen ábrázolható döntési fával, de lineárisan nem szeparálható. Az adatokra felvett legjobb szeparáló sík csak 65%-ot osztályoz helyesen.

Az eddigiekben úgy tekintettük a perceptronokat, mint olyan determinisztikus függvényeket, amelyeknek valószínűleg hibával terhelt a kimenete. Lehetőségünk van, hogy a szigmoid perceptron kimenetét valószínűségeként interpretáljuk – annak valószínűsé-

<sup>10</sup> Sigmoid függvény esetén ez a derivált  $g' = g(1 - g)$ .

<sup>11</sup> Küszöbperceptronokra, abol  $g'(in)$  nem definiált, a Rosenblatt (Rosenblatt, 1957) által kidolgozott eredeti perceptron tanulási szabály (perceptron learning rule) megegyezik a (20.12) egyenettel, leszámítva, hogy a  $g'(in)$  kimarad. Mivel  $g'(in)$  minden súlyra azonos, elhagyása az egyes mintáknál összességében csak a súlyfrissítés nagyságát változtatja meg, az irányát nem.

```

function PERCEPTRON-TANULÁS(példák, háló) returns egy perceptronhipotézis
  inputs: példák, egy mintahalmaz, minden egyikhez  $x = x_1, \dots, x_n$  bemenet és  $y$  kimenet tartozik
          háló, egy  $W_j, j = 0 \dots n$  súlyokkal és  $g$  aktivációs függvényvel rendelkező perceptron

  repeat
    for each  $p$  in példák do
       $in \leftarrow \sum_{j=0}^n W_j x_j[p]$ 
       $Err \leftarrow y[p] - g(in)$ 
       $W_j \leftarrow W_j + \alpha Err \times g'(in) \times x_j[p]$ 
    until valamelyen megállási feltétel teljesül
  return NEURÁLIS-HÁLÓ-HIPOTÉZIS(háló)

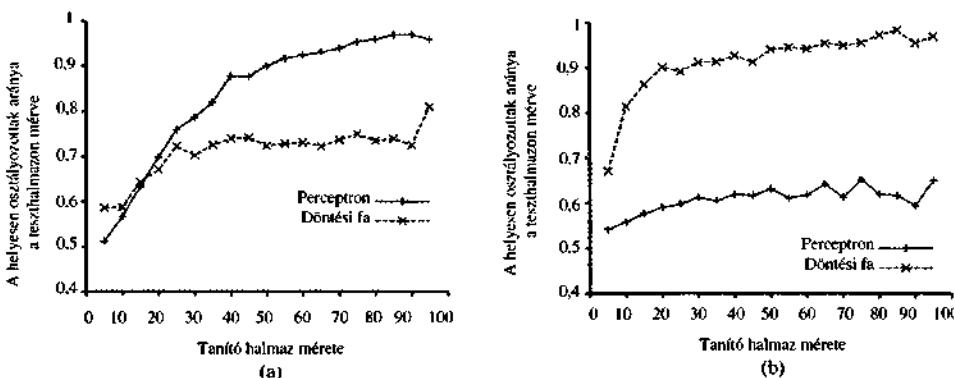
```

**20.21. ábra.** A gradiensalapú perceptron tanulási algoritmus, feltételezve, hogy a  $g$  aktivációs függvény differenciálható. Küszöbperceptronokra elhagyjuk a  $g'(in)$  faktort a szűlyfrissítésből. A NEURÁLIS-HÁLÓ-HIPOTÉZIS függvény egy olyan hipotézist ad vissza, amely bármely bemenetre kiszámítja a háló válaszát.

geként, hogy adott bemenetek esetén a valós kimenet 1. Ezzel az interpretációval úgy használhatjuk a szigmoidot, mint a Bayes-hálók feltételes eloszlásainak kanonikus reprezentációját (lásd 14.3. alfejezet). Levezethetünk egy tanuló algoritmust is, a standard módszernek megfelelően maximalizálva az adatok (feltételes) log likelihood értékét, amint ezt a fejezet korábbi részében ismertettük. Lássuk, hogyan is működik ez.

Vegyük egyetlen tanító példát, amelynek kívánt kimeneti értéke  $T$ , és legyen erre a példára a perceptron válasza  $p$ . Ha  $T = 1$ , akkor az adat feltételes valószínűsége  $p$ , ha  $T = 0$ , akkor a feltételes valószínűség  $(1 - p)$ . Egy egyszerű trükk alkalmazásával a log likelihoodot differenciálható formában írhatjuk fel. A trükk az, hogy ha a 0/1 változót egy kifejezés kifejezőjébe írjuk, akkor indikátorváltozóként (**indicator variable**) viselkedik:  $p^T$  akkor  $p$ , ha  $T = 1$ , egyébként 1; hasonlóképpen  $(1 - p)^{1-T}$  akkor  $(1 - p)$ , ha  $T = 0$ , különben 1. Ezek szerint az adat log likelihood értékét felírhatjuk, mint:

$$L = \log p^T (1 - p)^{1-T} = T \log p + (1 - T) \log(1 - p) \quad (20.13)$$



**20.22. ábra.** A perceptronok és döntési fák teljesítményének összehasonlítása. (a) A perceptronok jobbak a 11 bemenetű többségszáműség függvény tanulásában. (b) A döntési fák jobbak az éterem példában a VárunkE predikátum tanulásában.

A szigmoid függvény tulajdonságainak hála, a gradiens rendkívül egyszerű formára hozható (20.16. feladat):

$$\frac{\partial L}{\partial W_j} = Err \cdot x_j$$



Vegyük észre, hogy *sigmoid perceptronok esetén a maximum-likelihood tanulás súlyfrissítési vektorát megadó egyenlete alapvetően azonos a négyzetes hiba minimalizálásán alapuló frissítés vektorával*. Tehát azt mondhatjuk, hogy a perceptronnak még akkor is van valószínűségi interpretációja, ha a tanulási szabályt determinisztikus megközelítéssel származtattuk.

## Többrétegű előrecsatolt neurális hálók

Vizsgáljuk most a rejtett neuronokkal rendelkező hálókat. A legelterjedtebb esetben egy rejtett réteget<sup>12</sup> szoktak használni, ezt mutatja 20.24. ábra. A rejtett réteg hozzáadásának az az előnye, hogy kiterjeszti a háló által reprezentálható hipotézisek terét. Gondolunk minden egyes rejtett neuronra úgy, mint ami egy lágy küszöbfüggvényt reprezentál a bemeneti térben – lásd a 20.19. (b) ábrát. Ezek után gondolunk úgy egy kimeneti neuronra, mint ami számos ilyen függvény lineáris kombinációjának lágy küszöbfüggvénye. Például összeadva két – egymással szemben álló – lágy küszöbfüggvényt, és küszöböözve az eredményt, a 20.23. (a) ábrán látható „hegyerinc” függvényt kapjuk. Egymásra derékszögben álló két ilyen hegyerinc kombinálásával (tehát 4 rejtett neuron kimenetének kombinálásával) a 20.23. (b) ábrán látható „dudort” kapjuk.

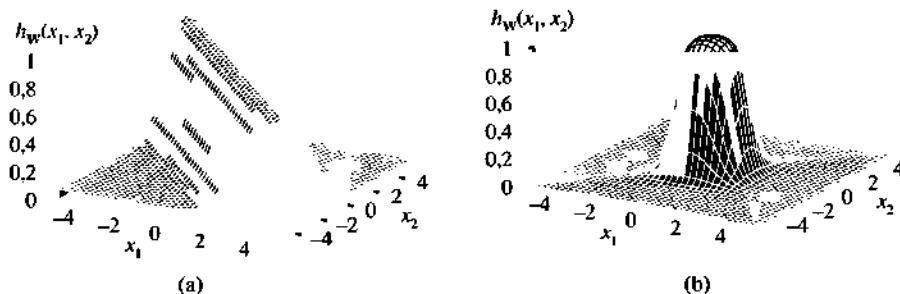
Több rejtett neuronnal különböző helyeken több, eltérő méretű dudort tudunk létrehozni. Valójában egyetlen, megfelelően nagy rejtett réteggel a bemenetek tetszőleges folytonos függvénye tetszőleges pontossággal reprezentálható, sőt két réteggel még nemfolytonos függvények is reprezentálhatók.<sup>13</sup> Sajnos egy *egyedi* neurális struktúra esetén nemigen lehet megmondani, hogy milyen függvények reprezentálhatók, és mi-lyenek nem.

Tegyük fel, hogy az étterem problémára akarunk konstruálni egy egy-rejtett-réteggel rendelkező hálót. minden példát 10 attribútum ír le, tehát egy 10 bemenetű hálóra lesz szükség. Hány rejtett neuron kell? A 20.24. ábrán bemutatunk egy négy rejtett neuronral felépített hálót. Az derült ki, hogy ez nagyjából megfelelő ehhez a problémához. A rejtett neuronok számának előzetes meghatározása még napjainkban sem jól megoldott probléma. (Lásd 856–857. oldal.)

A többrétegű háló tanuló algoritmusa hasonló a 20.21. ábrán bemutatott perceptron-tanulási algoritmushoz. Egy kisebb különbség, hogy több kimenet is lehet, így nem egy skalár kimeneti értékünk, hanem egy  $h_w(x)$  kimeneti vektorunk van, és minden egyes példához is egy  $y$  kívánt kimeneti vektor tartozik. Fontosabb különbség,

<sup>12</sup> Egyesek ezt háromrétegű hálózatnak nevezik, mások kétrétegűnek (mivel a bemenetek nem „igazi” neuronok). A zűrzavar elkerülése érdekében mi „egy-rejtett-rétegű háló”-nak nevezzük.

<sup>13</sup> A bizonyítás bonyolult, de legfőbb pontja az, hogy a bemenetek számaval exponenciálisan nő a szükséges rejtett neuronok száma. Például  $n$  bemenetű logikai függvények kódolására  $2^n/n$  rejtett neuron kell.



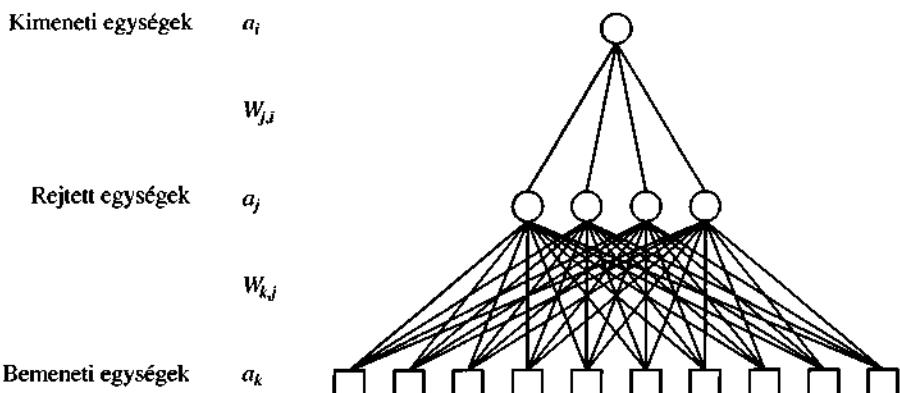
20.23. ábra. (a) Két – egymással szemben álló – lágy külsőfüggvény kombinációjának eredménye: egy hegyerinc. (b) Két hegyerinc kombinációjának eredménye a dudor.

ség, hogy míg a kimeneti rétegben a hiba nyilvánvalóan  $y - h_W$ , addig a rejtett rétegben a hiba misztikusnak tűnik, hiszen a tanító minták nem mutatják, hogy mimnek kellene lennie a rejtett csomópontok értékének. Kiderül, hogy a hibát **visszaterjeszthetjük** (back-propagate) a kimeneti rétegről a rejtett rétegekre. A hiba-visszaterjesztési eljárás (back-propagation) közvetlenül kiadódik a teljes hibagradiens levezetéséből. Először egy intuitív bizonyítással mutatjuk be az eljárást, majd megmutatjuk a levezetést is.

A kimeneti rétegre a súlyfrissítési szabály azonos a (20.12) egyenlettel. Több kimenetünk van, legyen  $Err_i$  az  $y - h_W$  hibavektor  $i$ -edik komponense. Hasznos lesz, ha bevezetjük a módosított hibát,  $\Delta_i = Err_i \times g'(in_i)$ -t, így a súlyfrissítési szabály:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad (20.14)$$

Ahhoz, hogy a bemeneti neuronok és a rejtett neuronok közti összeköttetések frissítését megoldjuk, a kimeneti csomópontok hibájához hasonló mennyiséget kell definálnunk. Ez az a pont, ahol a hiba-visszaterjesztést végezzük. A gondolat az, hogy a  $j$  rejtett csomópont valamelyen arányban „felelős” minden egyes – vele összeköttetésben lévő – kimeneti csomópont  $\Delta_j$  hibájáért. Így a  $\Delta_j$  értékeket a rejtett csomópont



20.24. ábra. Többrétegű neurális háló egy rejtett réteggel és 10 bemeneti egységgel, alkalmas az étterem problémához.

és a kimeneti csomópont közötti összeköttetés erőssége alapján osztjuk és visszaterjesztjük, hogy megkapjuk a rejtett réteg  $\Delta_j$  értékeit. A  $\Delta$  értékek terjesztési szabálya a következő:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \quad (20.15)$$

Ezek után a bemenetek és a rejtett réteg közötti súlyok frissítési szabálya szinte azonos a kimeneti réteg frissítési szabályával:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

A hiba-visszaterjesztési algoritmus a következő módon foglalható össze:

- Számítsuk ki a kimeneti neuronokra a  $\Delta$  értékeket a megfigyelt hiba alapján.
- A kimeneti réteggel kezdve ismételjük a következő lépéseket minden rétegre, amíg a legelső rejtett réteget el nem érjük:
  - Terjesszük vissza a  $\Delta$  értékeket a megelőző rétegre.
  - Frissítsük a két réteg közötti súlyokat.

Az algoritmust a 20.25. ábra mutatja be részleteiben.

A matematika iránt vonzalmat érzők kedvéért most az alapegyenletekből vezetjük a hiba-visszaterjesztési algoritmust. Egyetlen mintára a négyzetes hiba definíciója:

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

```

function HIBA-VISSZATERJESZTÉS-TANULÁS(példák, háló) returns egy neurális háló
  inputs: példák, minták halmaza, mindegyikhez x bemeneti és y kimeneti vektor
          háló, többrétegű háló, L réteggel, Wj,i súlyokkal és g aktivációs függvénnyel

  repeat
    for each p in példák do
      for each j csomópontra a bemeneti rétegen do aj  $\leftarrow x_j[p]
      for ℓ = 2 to L do
        ini  $\leftarrow \sum_j W_{j,i} a_j
        ai  $\leftarrow g(in_i)
        for each i csomópontra a kimeneti rétegen do
          Δi  $\leftarrow g'(in_i) \times (y_i[p] - a_i)
        for ℓ = L – 1 to 1 do
          for each j csomópontra az ℓ-edik rétegen do
            Δj  $\leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i
          for each i csomópontra az (ℓ + 1)-edik rétegen do
            Wj,i  $\leftarrow W_{j,i} + \alpha \times a_j \times \Delta_j
        until valamelyen megállási feltétel teljesül
        return NEURÁLIS-HÁLÓ-HIPOTÉZIS(háló)$$$$$$ 
```

20.25. ábra. A többrétegű hálókra kidolgozott hiba-visszaterjesztéses tanulási algoritmus

ahol az összegzés a kimeneti réteg csomópontjaira vonatkozik. Egy bizonyos  $W_{j,i}$  súlyra vett gradiens kiszámításához csak az  $a_i$  aktivációt kell deriválnunk, mivel az összeg összes többi tagja független  $W_{j,i}$ -től:

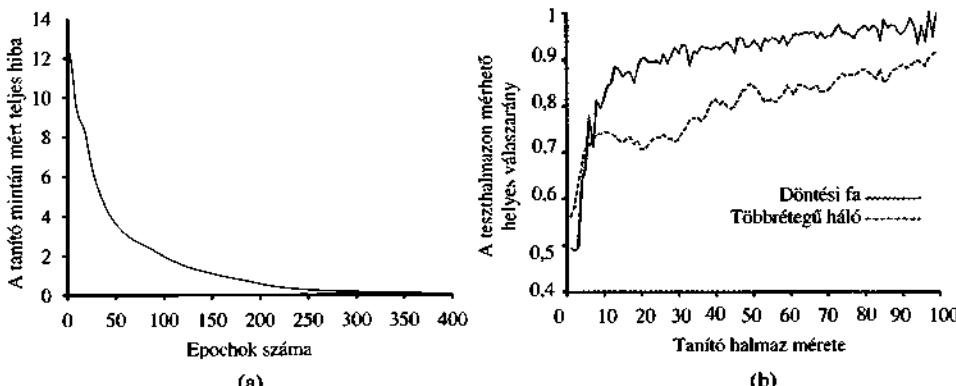
$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left( \sum_j W_{j,i} a_j \right) \\ &= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i\end{aligned}$$

ahol  $\Delta_i$ -t az előzőkel megegyezően definiáltuk. Ahhoz, hogy a bemeneti réteget és a rejttett réteget összekötő  $W_{k,j}$  súlyokra vonatkozó gradienst megkapjuk, meg kell tartanunk az  $i$  feletti teljes összegzést, hiszen az összes kimeneti  $a_i$  értékre hatással lehetnek  $W_{k,j}$  változásai. Az  $a_i$  aktivációkat szintén mindenki kell fejenünk. Aprólékosan bemutatjuk a levezetést, mivel érdekes megfigyelni, ahogy a differenciáló operátor visszaterjed hálón keresztül:

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left( \sum_j W_{j,i} a_j \right) \\ &= -\sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left( \sum_k W_{k,j} a_k \right) \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j\end{aligned}$$

ahol a  $\Delta_j$ -t az előzőkel megegyezően definiáltuk. Tehát megkaptuk azt a frissítési szabályt, amelyet már korábban megkaptunk intuitív úton is. Az is nyilvánvaló, hogy az eljárás folytatható több mint egy rejttett rétegű hálókra is, ami igazolja a 20.25. ábrán bemutatott általános algoritmust.

Keresztlüverekedvén magunkat (vagy átugorva) a matematikai levezetésen lássuk most, hogy milyen teljesítményt nyújt egy egy-rejttett-rétegű háló az étterem problémán. A 20.26. ábrán két görbét mutatunk be. Az első a **tanítási görbe** (*training curve*), amely a súlyfrissítés során az átlagos négyzetes hiba alakulását mutatja egy adott 100 elemű étterem példahalmazon mérve. Ez jól demonstrálja, hogy a háló valóban konvergál a tanító mintákra való tökéletes illeszkedéshez. A második görbe az étteremadatok standard tanulási görbéje. A neurális háló jól tanul, bár nem annyira gyorsan, mint a döntési fa tanulás. Ez talán nem meglepő, hiszen az adatokat egy egyszerű döntési fával generáltuk.



**20.26. ábra.** (a) Az éterem probléma egy adott példahalmazán felvett tanulási görbe, ami a súlyok számos epoch során történő módosításával elérte fokozatos hibacsökkenést mutatja. (b) Összehasonlító tanulási görbék, amelyek azt mutatják, hogy a döntési fa tanulás valamivel jobb teljesítményt ad, mint a többrétegű háló hiba-visszaterjesztéses tanulása.

A neurális hálók természetesen messze bonyolultabb tanulási feladatokra képesek, bár meg kell jegyeznünk, hogy szükség van némi babrálásra ahhoz, hogy megfelelő hálóstruktúrát kapjunk, és a súlyterben valahol a globális optimum közelébe konvergáljunk. A szó szoros értelmében tízezerszám vannak publikált neurális háló alkalmazások. A 20.7. alfejezet alaposabban bemutat egyet.

## Neurális hálóstruktúrák tanulása

Az eddigiekben adott hálóstruktúra mellett történő súlytanulással foglalkoztunk. A Bayes-hálóhoz hasonlóan azt is meg kell értenünk, hogy hogyan találhatjuk meg a legjobb hálóstruktúrát. Ha túl nagy hálót használunk, akkor az egy nagy táblázatot kialakítva képes lesz memorizálni az összes példát, de nem feltétlenül lesz képes olyan bemenetekre jól általánosítani, amelyeket nem látott korábban.<sup>14</sup> Más szavakkal a neurális háló – éppúgy, mint az összes statisztikus modell – hajlamos a túllilleszkedésre (*overfitting*), ha túl sok modellparaméter van. Ezt bemutattuk a 18.1. ábrán (752. oldal), ahol a (b) és (c) sok-paraméteres modell jól illeszkedett az adatokra, de nem volt képes olyan jó általánosításra, mint a kevés paraméterrel rendelkező (a) és (d) modellek.

Ha ragaszkodunk a teljesen összekötött hálókhöz, akkor egyedül a rejtett rétegek számára és méretére korlátozódnak a választási lehetőségeink. A szokásos megközelítés, hogy sok struktúrát kipróbálunk, és megtartjuk a legjobbat. A 18. fejezetben ismertetett **keresztvalidációs** (*cross-validation*) technikára van szükségünk, ha el akarjuk kerülni a teszthalmazra való kükucska káltást (*peeking*). Azaz azt a hálóarchitektúrát választjuk, amely a validációs halmazon a legnagyobb jóslási pontosságot adja.

<sup>14</sup> Azt figyelték meg, hogy a nagyon nagy hálók *mindaddig* jól általánosítanak, *amíg a súlyakat kis értéken tartjuk*. Ez a megszorítás az aktivációs értékeket a  $g(x)$  szigmoid függvény lineáris tartományában tartja, ahol  $x$  közel nulla. Ez viszont azt jelenti, hogy a háló úgy viselkedik, mint egy sokkal kevesebb paraméterrel rendelkező lineáris függvény (lásd 20.17. feladat).

Ha figyelembe akarunk venni nem teljesen összekötött hálókat is, akkor szükségünk van egy hatékony módszerre, amely a lehetséges összeköttetési topológiák nagyon nagy terében keres. Az **optimális agykárosodás** (**optimal brain damage**) módszere egy teljesen összekötött hálóból indul ki, és összeköttetéseket távolít el belőle. Miután a hálót első menetben tanítottuk, egy információelméleti megközelítés segítségével meghatározzuk az eltávolítható összeköttetések optimális készletét. A hálót ezek után újratanítjuk, és ha teljesítménye nem csökkent, az eljárást megismételjük. Az összeköttetések eltávolításán felül olyan neuronokat is eltávolíthatunk, amelyek nem sokkal járulnak hozzá a megoldáshoz.

Rengeteg algoritmust javasoltak arra, hogy egy kisebb hálóból nagyobbat növesszenek. Egyikük, a **csempézés** (**tiling**), a döntési fa tanulásra emlékeztet. Az ötlet az, hogy kezdjünk egyetlen neuronnal, amely a legjobbját nyújtja, hogy annyi tanító mintára adjon helyes választ, ahányra csak lehet. További neuronokat adunk hozzá, hogy megoldjuk azokat a példákat, amelyekre az első neuron rossz választ adott. Az algoritmus csak annyi neuront ad hozzá, amennyi az összes minta megoldásához szükséges.

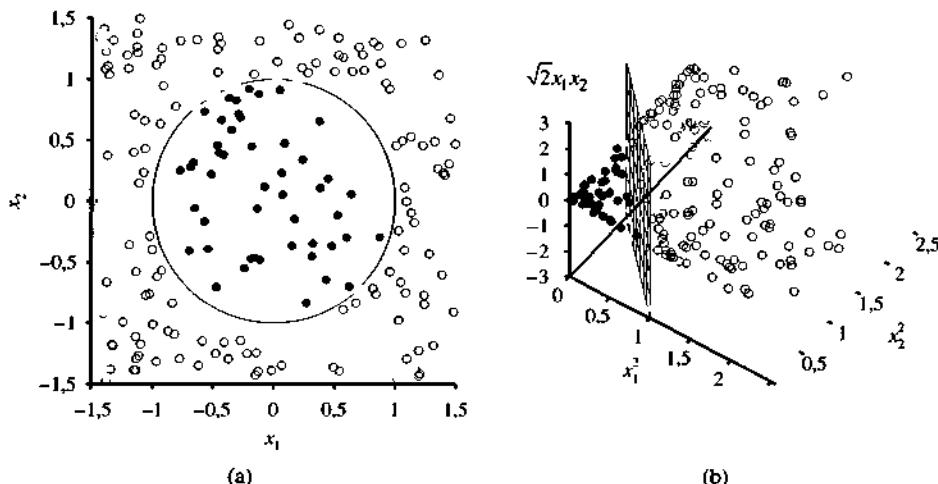
## 20.6. KERNELGÉPEK

A neurális hálók tárgyalása után maradt egy dilemmánk. Az egyrétegű hálóknak nagyon egyszerű és hatékony tanulási algoritmusuk van, de nagyon korlátozott a kifejezőképességük, csupán lineáris döntési határokat képesek megtanulni a bemeneti térből. Másrészt a többrétegű hálók sokkal kifejezőbbek – általános nemlineáris függvényeket képesek reprezentálni –, de a rengeteg lokális minimum jelenléte, illetve a sokdimenziós súlytér miatt nagyon nehéz a tanításuk. Ebben az alfejezetben egy relatíve új tanulómódszer családot fedezünk fel, az úgynevezett **szupport vektor gépeket** (**support vector machines**, SVM), vagy általánosabban a **kernelgépeket** (**kernel machines**). Bizonyos fokig a kernelgépek a két oldal legjobb tulajdonságait egyesítik. Azaz ezek a módszerek hatékony tanulási algoritmusokat alkalmaznak, *ugyanakkor* képesek bonyolult, nemlineáris függvények reprezentálására.

A kernelgépek teljes mélységű tárgyalása meghaladja ennek a könyvnek a kereteit, de a fő gondolatot egy példán keresztül illusztráljuk. A 20.27. (a) ábra egy kétdimenziós bemeneti teret mutat, amelyet az  $x = (x_1, x_2)$  attribútumok írnak le. A pozitív példák ( $y = +1$ ) egy kör alakú rész belsőjében, a negatív példák ( $y = -1$ ) azon kívül helyezkednek el. Nyilvánvaló, hogy a probléma megoldására nem létezik lineáris szeparátor. Tegyük fel, hogy valamilyen számított tulajdonságok segítségével új formára hozzuk a példákat – azaz az összes bemeneti  $x$  vektort leképezzük a tulajdonságértékekből formált új,  $F(x)$  vektorra. A példában használjuk a következő három tulajdonságot:

$$f_1 = x_1^2 \quad f_2 = x_2^2 \quad f_3 = \sqrt{2} x_1 x_2 \quad (20.16)$$

Rövidesen látni fogjuk, hogy honnan vesszük ezeket a tulajdonságokat, de most csak nézzük meg azt, hogy mi is történt. A 20.27. (b) ábra mutatja az adatokat az új, a tulajdonságok által definiált háromdimenziós térből: ebben a térből *lineárisan szeparálhatók!* Ez a jelenség eléggé általános: ha az adatokat megfelelően sokdimenziós térből képezzük le, akkor minden lineárisan szeparálható lesznek. Itt mi csak három dimen-



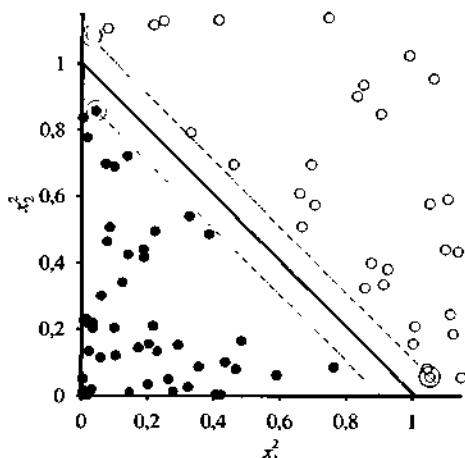
**20.27. ábra.** (a) Egy kétdimenziós tanító halmaz, amelyben a pozitív példákat fekete, a negatívakat fehér körök jelölik. Az  $x_1^2 + x_2^2 \leq 1$  valódi elválasztó határt is bejelöltük. (b) Ugyanazok az adatok az  $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$  háromdimenziós térbe való leképezés után. Az (a) ábrán látható kör alakú döntési határ a háromdimenziós téren lineáris döntési felületbe ment át.

zót használtunk,<sup>15</sup> de ha  $N$  pontunk van – akkor speciális esetek kivételével –, egy  $N - 1$  vagy ennél magasabb dimenziós téren a pontok minden lineárisan szeparálhatók lesznek (lásd 20.21. feladat).

Ennyi az egész? Egyszerűen létrehozunk egy nagy halom számított tulajdonságot, és a megfelelő sokdimenziós téren megkeressük a lineáris szeparátort? Sajnálatos módon nem ilyen egyszerű. Emlékezzünk, hogy a  $d$  dimenziós téren a lineáris szeparátort egy  $d$  paraméteres egyenlet határozza meg, így aztán az a veszély fenyeget, hogy ha  $d \approx N$  (ahol  $N$  az adatpontok száma), akkor könnyen túlilleszkedhetünk az adatokra. (Ez ahhoz hasonló, mint amikor egy magas fokszámú polinommal túlillesztünk adatokat, ahogy ezt a 18. fejezetben tárgyalunk.) Ezen okból a kernelgépek rendszerint az *optimális* lineáris szeparátort találják meg. Azt nevezzük optimálisnak, amelynek legnagyobb a **tartaléka (margin)**: a lineáris szeparátor és a pozitív példák között az egyik oldalon, illetve a lineáris szeparátor és a negatív példák között a másikon. (Lásd 20.28. ábra.) A számítógépes tanulás elmélet módszereit (lásd 18.5. alfejezet) használva megmutatható, hogy ez a szeparátor az új példák robusztus általánosítására nézve nagyon jó tulajdonságokkal rendelkezik.

Hogyan találjuk meg ezt a szeparátort? Kiderül, hogy ez egy **kvadratikus programozással (quadratic programming)** megoldható optimalizálási feladat. Tegyük fel, hogy  $\mathbf{x}_i$  példáink vannak, az osztálybasorolásuk  $y_i = \pm 1$ , és a bemeneti téren optimális szeparátort akarunk találni. Ekkor a megoldandó kvadratikus programozási feladat azon paraméterértékek megtalálása, amelyek az  $\alpha_i \geq 0$  és  $\sum_i \alpha_i y_i = 0$  korlátozó feltételek mellett maximálják a következő kifejezést:

<sup>15</sup> Az olvasó felfedezheti, hogy elég lett volna csupán  $f_1$  és  $f_2$  használata, de a 3D leképezés jobban illusztrálja az ötletet.



**20.28. ábra.** Az első két dimenzióra vetített közelkép: a 20.27. (b) ábra optimális szeparátora. A szeparátor vastag vonallal jelöltük, a hozzá legközelebbi pontokat – a szupport vektorokat – bekarakáltuk. A tartalek nem más, mint a pozitív és negatív példák közti elválasztó sáv szélessége.

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (20.17)$$

Bár ennek a kifejezésnek a deriválása nem nagyon fontos pontja a történetnek, de azért van két lényeges tulajdonsága. Először is a kifejezésnek egyetlen, globális maximuma van, ami hatékonyan megtalálható. Másodszor az adatok kizárolag pontpárok skalárszorzataiként jelennek meg a kifejezésben. Ez a második tulajdonság magára a szeparátorra is igaz, ha az optimális  $\alpha_i$ -ket kiszámítottuk, akkor:

$$h(\mathbf{x}) = \text{sign} \left( \sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) \right) \quad (20.18)$$

Az ezen egyenlettel definiált optimális szeparátor utolsó fontos tulajdonsága az, hogy az egyes adatpontokkal asszociált  $\alpha_i$  súlyok minden nullák, kivéve a szeparátorhoz legközelebb eső pontokat – ezeket nevezzük szupport vektoroknak (support vector). (Azért nevezzük így őket, mert ők „tartják” a szeparáló síkot.) Mivel rendszerint jóval kevesebb szupport vektor van, mint adatpont, ezért az optimális szeparátort meghatározó tényleges paraméterszám rendszerint jóval kisebb  $N$ -nél.

Rendszerint nem várhatjuk el, hogy lineáris szeparátorral találunk az  $\mathbf{x}$  bemeneti térben, de könnyen belátható, hogy a sokdimenziós  $F(\mathbf{x})$  tulajdonságterben találhatunk lineáris szeparátorokat. Ennek érdekében a (20.17) egyenletben  $\mathbf{x}_i \cdot \mathbf{x}_j$ -t egyszerűen kicseréljük  $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$ -re. Ez önmagában nem túlzottan figyelemre méltó –  $\mathbf{x}$  kicserélése  $F(\mathbf{x})$ -re bármely tanuló algoritmusban elérné a kívánt hatást –, de a skalárszorzatnak van néhány érdekes tulajdonsága. Az  $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$  gyakran kiszámítható anélkül, hogy először kiszámítanánk minden pontra  $F$ -et. A (20.16) egyenlettel definiált háromdimenziós tulajdonsgáter példánkban némi algebrai átalakításokkal megmutatható, hogy:

$$F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

Az  $(x_i \cdot x_j)^2$  kifejezést **kernelfüggvénynek (kernel function)** nevezik, és  $K(x_i, x_j)$ -vel jelöljük. A kernelgépek szempontjából ez egy olyan függvény, amely pontpárokra alkalmazható avégett, hogy valamilyen tulajdonságterben kiszámítuk a skalárszorzatukat. Ennek megfelelően újrafogalmazhatjuk állításunkat: a (20.17) egyenletben  $x_i \cdot x_j$ -t egyszerűen kicsérélve a  $K(x_i, x_j)$  kernelfüggvényre, a sokdimenziós  $F(x)$  tulajdonságterben találhatunk lineáris szeparátorokat. Így a tanulást a sokdimenziós térben végezhetjük, de csupán kernelfüggvények értékét kell kiszámítanunk, nem kell az összes pontra a tulajdonságok teljes készletét kiszámítani.

A következő lépés – aminek most már kézenfekvőnek kell lennie – az, hogy meglás-suk, a  $K(x_i, x_j) = (x_i \cdot x_j)^2$  kernelben nincs semmi különleges. Ez egy bizonyos sokdimenziós tulajdonságternek felel meg, de más kernelfüggvények más tulajdonságerekkel vannak kapcsolatban. A **Mercer-tétel** (1909) azt mondja ki, hogy bármely „ésszerű”<sup>16</sup> kernelfüggény megfelel *valamelyen* tulajdonságternek. Ez a tulajdonságter még egész ártatlannak látszó kernelek esetén is nagyon nagy lehet. Például a  $K(x_i, x_j) = (1 + x_i \cdot x_j)^d$  polinomiális kernel (**polynomial kernel**) egy olyan tulajdonságternek felel meg, amelynek dimenziója  $d$ -ben exponenciális. Ha a (20.17) egyenletben ilyen kerneleket használunk, akkor *hatékonyan* kereshetünk *lineáris szeparátorokat sok milliárd* (vagy *egyes esetekben végtelen*) dimenziós terekben. Az eredményként kapott lineáris szeparátorokat visszavetítve az eredeti bemeneti térbe, a pozitív és negatív példákat elválasztó tétszölegenek tekervényes, neilineáris határfelületeket kaphatunk.

Említettük az előző részben, hogy a kernelgépek kiemelkedően teljesítenek a kézirásos számjegyek felismerésében, de gyorsan felhasználják őket más alkalmazásokhoz is, különösen olyanokhoz, ahol sok bemeneti tulajdonság van. Ennek a folyamatnak részeként számos új kernelt dolgoztak ki, amelyek karakterfüzérekre, fákra és más nem-numerikus adattípusokra alkalmazhatók. Az is megfigyelhető, hogy a kernelmódszer nemcsak optimális szeparátorok keresésére alkalmas, hanem bármely olyan algoritmusra, amely úgy átalakítható, hogy csak adatpontpárok skalárszorzatát használja, mint a (20.17), illetve (20.18) egyenlet. Amint ezt sikerült megtennünk, a skalárszorzat kicserélhető egy kernelfüggvényre, és elkészítettük az algoritmus **kernelesített (kernelized)** változatát. Ez az átalakítás többek közt a  $k$ -legközelebbi-szomszéd algoritmusra és a perceptron tanulásra is könnyen elvégezhető.

## 20.7. ESETTANULMÁNY: KÉZZEL ÍROTT SZÁMJEGYEK FELISMERÉSE

A kézzel írott számjegyek felismerése számos alkalmazásban felmerülő fontos probléma. Néhány ezek közül: postai levelek irányítószám szerinti automatikus osztályozása, csekkek és adó-visszatérítési számlák automatikus leolvasása, kézi számítógépek adat-bevitelle. Ez olyan terület, ahol gyors volt a fejlődés, részben a jobb tanuló algoritmusoknak, részben a jobb tanító adatbázisoknak köszönhetően. Az Egyesült Államok Mérés- és Szabványügyi Hivatala (National Institute of Standards and Technology, NIST) egy 60 000 megcímkézett számjegyből álló archívumot hozott létre, amelyben minden számjegy egy  $20 \times 20 = 400$  pixeles, 8 bites szürke árnyalatú képen jelenik

<sup>16</sup> Itt ésszerű alatt azt értjük, hogy a  $K_{ij} = K(x_i, x_j)$  mátrix pozitív definit; lásd A) függelék.

meg. Ez az új tanuló algoritmusok összehasonlításának egyik standard mércéjévé (benchmark) vált. A 20.29. ábra bemutat néhány számjegyet.

Számos különböző tanuló algoritmust kipróbáltak. Az elsők egyike, és egyben valószínűleg a legegyszerűbb, a **3-legközelebbi-szomszéd** (**3-nearest-neighbor**) osztályozó, amelynek további nagy előnye, hogy nem igényel tanítási időt. Ugyanakkor memóriaalapú algoritmusként minden a 60 000 képet tárolnia kell, és a futási idejű teljesítmény lassú. A teszthalmazon elérte hibaarány 2,4%.

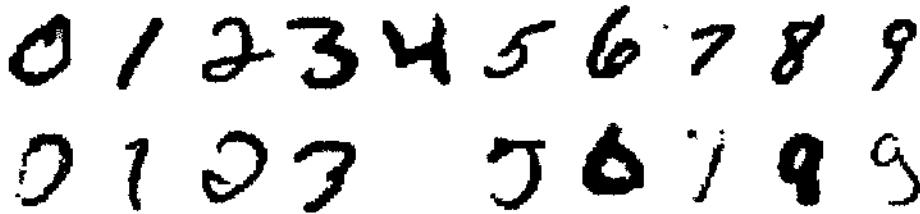
Kifejlesztettek egy **egy-rejtett-rétegű neurális hálót** (**single-hidden-layer neural network**) a feladatra, 400 bemenettel (egy bemenet pixelenként) és 10 kimenettel (osztályonként egy). Keresztsvalidációt használva azt találták, hogy durván 300 rejtett neuron adta a legjobb teljesítményt. A rétegek között teljes összeköttetést valósítottak meg, így összesen 123 300 súlyt használtak. Ez a háló 1,6% hibaárányt ért el.

LeNet néven egy sor **speciális neurális hálót** (**specialized neural network**) szerkesztettek egybe azért, hogy a problémában rejlö struktúrát kihasználják. Az egyik strukturális adottság, hogy a bemeneti pixelek kétdimenziós tömböt alkotnak, a másik, hogy egy ábra kismértékű pozíció- vagy dőléssírány-változása lényegtelen. minden egyes háló  $32 \times 32$  egységből álló bemeneti réteget tartalmaz, amelyre a  $20 \times 20$  pixelt úgy kapcsolták, hogy minden egyes bemeneti egység egy lokális szomszédságot reprezentált. Ezt három rejtett réteg követte. Mindegyik réteg számos  $n \times n$ -es mezőből áll, ahol  $n$  kisebb, mint a megelőző réteg, tehát a háló a bemenetek alul-mintavezetését valósítja meg. Egy mezőn belül minden egyik egységre azonos súlyokat kényszerítünk, így a mező tulajdonságdetektorként működik: olyan tulajdonságokat érzékel, mint egy hosszú függőleges vonal vagy egy rövid félkörív jelenléte. A kimeneti réteg 10 neuronból áll. Ennek az architektúrának számos változatát kipróbálták: egyik tipikus képviselőjük a rejtett rétegeiben rendre 768, 192, illetve 30 neuront tartalmazó háló volt. A tanító halmazt kiegészítették további mintákkal, ezeket a valósakból kismértékű affin transzformációkkal nyerték, amelyek a következők voltak: eltolás, kismértékű elforgatás, nagyítás/kicsinyítés. (Természetesen a transzformációknak kismértékűeknek kell lenniük, különben a 6-ot 9-cé transzformáljuk!) A LeNet által elérte legkisebb hibaárány 0,9% volt.

Egy **turbózott neurális háló** (**boosted neural network**) három LeNet-architektúrát kombinált: a másodikat egy olyan mintakeverékkel tanították, amelyen az első 50%-os hibát produkált, a harmadikat pedig olyan mintákkal, amelyekre az első kettő különböző eredményt adott. A tesztelés során a három háló a tíz lehetséges számjegy mindenikére egy-egy súlyal szavazott, ezeket összeadva határozták meg a győztest. A teszthibaárány 0,7% volt.

Egy 25 000 szupport vektort használó **szupport vektor gép** (**support vector machine**) (lásd 20.6. alfejezet) 1,1% hibaárányt ért el. Ez azért figyelemre méltó, mert az SVM-technika – éppúgy mint az egyszerű legközelebbi-szomszéd megközelítés – szinte nem igényelt gondolkodást vagy iteratív kísérletezést a fejlesztőtől, mégis az évekig fejlesztett LeNet-et megközelítő eredményt ért el. A szupport vektor gépek valójában nem használják ki a probléma struktúráját, tehát ugyanolyan jól teljesítenének, ha a pixeleket valamilyen permutált sorrendben használnánk.

A **virtuális szupport vektor gépet** (**virtual support vector machine**) egy standard SVM-ként indítjuk, majd egy olyan technikával javítjuk, amelyet a probléma struktúrájának kihasználására hoztak létre. Ahelyett hogy az összes lehetséges pixelpárból képzett kernet megengednénk, ez olyan kernelekre koncentrál, amelyek szomszédos pixelpárok formá-



20.29. ábra. Néhány példa az NIST kézzel írt számjegyekre létrehozott adatbázisából. A felső sorban a 0–9 karakterek könnyebben felismerhető példáit mutatjuk be, az alsó sorban pedig ugyanezen számjegyekre nehezebben felismerhető példákat látunk.

lódtak. A LeNethez hasonlóan itt is a példák transzformáltjaival egészítjük ki a tanító halmazt. Egy virtuális SVM érte el az eddigi legjobb hibaárányt, 0,56%-ot.

Az alakillesztés (**shape matching**) a számítógépes látás egyik eljárása, amelyet arra hoztak létre, hogy tárgyak két különböző képének megfelelő részeit összerendeljék. (Lásd 24. fejezet.) Az alapötlet, hogy vegyük egy-egy pontalmazt minden két képről, és számítjuk ki az első kép minden egyes pontjára, hogy a második kép mely pontja felel meg neki. Ebből a megfeleltetésből aztán számítjuk ki a két képet összekapcsoló transzformációt. Ez a transzformáció a képek közt egy távolság mértékét határozza meg. Ez a távolság mérték indokoltabb, mint egyszerűen az egymástól különböző pixelek megszámlálása. Kiderült, hogy az erre a távolság mértékre alapozott 3-legközelebbi-szomszéd algoritmus nagyon jól működik. Csupán 20 000-et használva a 60 000 számjegyből, és egy Canny éldetektor képének csupán 100 pixelét felhasználva az alakillesztés eljárás 0,63% teszthibaárányt ér el.

Becslések szerint az **emberek** (**human**) kb. 0,2% hibaárányt érnak el ennél a problémánál. Ez a szám nemiképpen gyanús, mert az embereket nem tesztelték olyan alaposan, mint a gépi tanulás algoritmusait. Az Egyesült Államok Postájának (United States Postal Service) egy hasonló – számjegyekből képzett – adathalmazán az emberek által elkövetett hibaárány 2,5% volt.

A következő ábra összegzi a tárgyalt hét eljárás esetén a hibaárányt, a futási időt, a memóriaigényt és a szükséges tanítási időt. Még egy mértéket ad ezekhez, a számjegyek azon arányát, amelyet vissza kell utasítanunk, hogy a 0,5% hibaárányt elérjük. Ha például egy SVM-nek megengedjük, hogy bemeneti képeinek 1,8%-át visszautasítsa – azaz átpasszolhatja valakinek, hogy az hozza meg a végső döntést –, akkor a képek többi 98,2%-án a hibaárány 1,1%-ról 0,5%-ra csökken.

A következő táblázat a hét tárgyalt technikára összegzi a hibaárányt és néhány további jellemzőt.

	3 L-Sz	300 rejtett	LeNet	Turbózott LeNet	SVM	Virtuális SVM	Alak- illesztés
Hibaárány (%)	2,4	1,6	0,9	0,7	1,1	0,56	0,63
Futási idő (millisec/számjegy)	1000	10	30	50	2000	200	
Memóriaigény (MB)	12	0,49	0,012	0,21	11		
Tanítás ideje (nap)	0	7	14	30	10		
A 0,5% hiba eléréséhez a visszautasított %	8,1	3,2	1,8	0,5	1,8		

## 20.8. ÖSSZEFOGLALÁS

A statisztikai tanulás módszerei széles skálán helyezkednek el: az egyszerű átlagszámításoktól a bonyolult modellek – mint például a Bayes-hálók vagy neurális hálók – konstruálásáig. Alkalmazási területük a számítógép-tudományra, a mérnöki alkalmazásokra, a neurobiológiára, a pszichológiára és a fizikára is kiterjed. Ebben a fejezetben bemutattunk néhány alapelvet, és ízelítőt adtunk a matematikai tárgyalásból. A következő fő pontok szerepeltek:

- A **Bayes-tanulási (Bayesian learning)** módszerek a tanulást valószínűségi következetésként fogalmazzák meg, a megfigyelések alapján frissítve a hipotézisek a priori eloszlásait. Ez a meghközelítés jó eszköz az Ockham borotvája elv megvalósítására, de bonyolult hipotézisterek esetén hamar kezelhetetlenné válik.
- A **maximum a posteriori (MAP)** tanulás az adatok alapján választ ki egyetlen, a legvalószínűbb hipotézist. A hipotézis priorit is használjuk, ez a módszer gyakran jobban kezelhető, mint a tiszta Bayes-tanulás.
- A **maximum-likelihood** tanulás egyszerűen azt a hipotézist választja, amely maximálja az adatok likelihood értékét. Megfelel egy egyenletes prior mellett végrehajtott MAP-tanulásnak. Egyszerű esetekben, mint a lineáris regresszió és a teljesen megfigyelhető Bayes-hálók, a maximum-likelihood megoldás könnyen előállítható zárt alakban. A **naiv Bayes- (naive Bayes)** tanulás különösen hatékony technika, amely különböző bonyolultságú feladatokra is jól illeszthető.
- Amikor a változók között van néhány rejtett (nem megfigyelhető), akkor az EM algoritmussal lokális maximum-likelihood megoldásokat találhatunk. Az alkalmazások kiterjednek a kevert Gauss-jelek osztályozására, a Bayes-hálók tanulására és a rejtett Markov-modellek tanulására.
- A **modellválasztásra (model selection)** ad példát a Bayes-hálók struktúrájának tanulása. Ez rendszerint egy, a struktúrák terében végzett diszkrét keresést foglal magában. Szükség van valamilyen módszerre a modell bonyolultsága és az illeszkedés mértéke közti kompromisszum létrehozásához.
- A **példányalapú modellek (instance-based model)** a tanító példányok gyűjteményének eloszlását reprezentálják. Így a paraméterek száma a tanító halmaz méretével nő. A **legközelebbi-szomszéd (nearest neighbor)** módszerek a kérdéses mintapont közelében lévő példányokat nézik, míg a **kernelmódszerek** az összes példány távol-sággal súlyozott kombinációját.
- A **neurális hálók (neural networks)** nem mások, mint sok paraméterrel rendelkező, komplex nemlineáris függvények. Paramétereiket zajos adatok alapján tanulhatják meg. Több ezer alkalmazásban használták már őket.
- A **perceptron** egy előrekesztőt neurális háló, amelynek nincs rejtett rétege, és csak **lineárisan szeparálható (linearly separable)** függvények reprezentálására alkalmas. Ha az adatok lineárisan szeparálhatók, akkor egy egyszerű súlyfrissítési szabály alkalmazásával az adatokra való pontos illeszkedést tudunk elérni.
- A **többrétegű előrekesztő (multilayer feed-forward)** neurális hálók – ha kellő számú neuronjuk van – tetszőleges függvények reprezentálására képesek. A **hiba-visszaterjesztési (back-propagation)** algoritmus a kimeneti hiba minimalizálása érdekében gradiensalapú csökkentést valósít meg a paramétertérben.

A statisztikai tanulás továbbra is igen aktív kutatási terület. Mind az elmélet, mind a gyakorlat hatalmas lépésekkel haladt, míg elérünk addig a pontig, hogy szinte bármely modell megtanulható, ha megvalósítható rá egzakt vagy közelítő következetés.

## Irodalmi és történeti megjegyzések

Az MI korai éveiben a statisztikai tanuláselmélet a kutatás aktívan művelt területe volt (Duda és Hart, 1973), de elkülönült az MI fősodrától, ahogy ez utóbbi egyre inkább a szimbolikus módszerekre koncentrált. Különböző formákban folytatódott – egyesek explicit módon valószínűségek, mások nem – olyan területeken, mint az alakzatfelismerés (pattern recognition) (Devroye és társai, 1996) és az információkeresés (information retrieval) (Salton és McGill, 1983). Nem sokkal a Bayes-háló modellek bevezetése után, az 1980-as évek végén az érdeklődés ismét erősen ráirányult, nagyjából ugyan ebben az időben jelent meg a neurális hálók statisztikai megközelítése. Az 1990-es évek végén a gépi tanulás, a statisziika és a neuronhálók területén is az érdeklődés középpontjába kerültek az adatok alapján nagy valószínűségi modelleket létrehozó módszerek.

A naiv Bayes-modell egyike a legrégebbi és legegyszerűbb Bayes-hálóknak, megjelenése egészen az 1950-es évekig követhető vissza. Eredetüket megemlíttetük a 13. fejezet záró megjegyzéseiben. Részleges magyarázat található Domingos és Pazzani publikációjában (Domingos és Pazzani, 1997). A naiv Bayes-tanulás turbózott változata nyerte az első KDD Cup adatbányászati versenyt (Elkan, 1997). Heckerman kitűnő bevezetését adja a Bayes-háló tanulás általános problematikájának (Heckerman, 1998). Spiegelhalter és társai a Bayes-hálók Bayesi paramétertanulását tárgyalta Dirichlet-priorok esetére (Spiegelhalter és társai, 1993). A BUGS szoftvercsomag (Gilks és társai, 1994) számos gondolatot megtestesített ezek közül, nagyon hatékony eszközt biztosított az összetett valószínűségi modellek felállítására és tanulására. A Bayes-hálóstruktúra tanulásának első algoritmusai feltételes függetlenségi teszteket használtak (Pearl, 1988; Pearl és Verma, 1991). Spirtes és társai hasonló elvek alapján dolgozták ki átfogó megközelítéstük, valamint a TETRAD csomagot Bayes-hálóstruktúra tanulás céljaira (Spirtes és társai, 1993). Az azóta végrehajtott algoritmikus javítások a 2001-es KDD Cup adatbányászati versenyen egy Bayes-háló tanulási algoritmus (Cheng és társai, 2002) meggyőző győzelméhez vezettek. (Itt a speciális megoldandó feladat egy 139 351 tulajdonsággal leírt bioinformatikai probléma volt!) Cooper és Herskovits egy likelihood maximalizáláson alapuló struktúratanulási megközelítést fejlesztett ki (Cooper és Herskovits, 1992), ezt Heckerman és társai fejlesztették tovább (Heckerman és társai, 1994). Friedman és Goldszmidt mutatták ki a lokális feltételes eloszlások reprezentációjának a megtanult struktúrára gyakorolt hatását (Friedman és Goldszmidt, 1996).

A rejtejtő változókkal és a hiányzó adatokkal való valószínűségi modell tanulás általános problematikáját az EM algoritmussal kíséreltek meg kezelní (Dempster és társai, 1977). Ezt számos meglévő módszerből absztrahálták, amelyek között található a rejtejtő Markov-modell (HMM) tanulásra szolgáló Baum–Welch-algoritmus is (Baum és Petrie, 1966). (Maga Dempster az EM algoritmust inkább sémának tekinti, nem algoritmusnak, mivel jó adag elméleti matematikai munkára lehet szükség mielőtt egy új eloszláscsaládra alkalmazható lenne.) Manapság az EM egyike a tudományos kutatásban legelterjedtebb használt algoritmusoknak, McLachlan és Krishnan egy teljes könyvet szenteltek neki

és tulajdonságainak (McLachlan és Krishnan, 1997). A kevert modellek – beleértve a kevert Gauss-modellek – tanulásának speciális problémát Titterington és társai tárgyalják (Titterington és társai, 1985). Az AUTOCLASS volt az első sikeres rendszer az MI-n belül, amely az EM-et alkalmazta kevert modellezésre (Cheeseman és társai, 1988; Cheeseman és Stutz, 1996). Az AUTOCLASS-t egy sor valós tudományos osztályozási feladatra alkalmazták; ezek közül kettő: spektrális tulajdonságok alapján új csillagtípusok felfedezése (Goebel és társai, 1989); új fehérje- és intronosztályok felfedezése DNS/féhéjeszekvencia adatbázisokban (Hunter és States, 1992).

A rejtejt változókkal rendelkező Bayes-hálók tanulására kifejlesztett EM algoritmus Lauritzen munkája (Lauritzen, 1995). Mind a Bayes-hálók, mind a dinamikus Bayes-hálók esetén a gradiensalapú eljárások is hatékonyak bizonyultak (Russell és társai, 1995; Binder és társai, 1997a). A strukturális EM algoritmus kifejlesztése Friedman nevéhez fűződik (Friedman, 1998). A Bayes-hálók struktúrájának megtanulhatósága szoros kapcsolatban van a *kauzális* kapcsolatok adatokból történő visszanyerésének kérdésével. Azaz lehetséges-e Bayes-hálókat úgy megtanulni, hogy az előállított hálóstruktúra valós kauzális hatásokat jelezzen? A statisztikusok hosszú évek óta elkerülték ezt a kérdést, azt hitték, hogy a megfigyelt adatok (ellenértében a kísérletek során előálltakkal) csak korrelációs információt hordoznak. Végül is bármely két változóra, amelyek egymással kapcsolatban állónak tűnnek, lehet, hogy valójában inkább egy harmadik – mindenkorra kauzális hatást gyakorló – ismeretlen változó hatása alatt állnak, nem pedig egymásra gyakorolnak közvetlen hatást. Ennek ellenkezőjére Pearl adott megegyőző érveket (Pearl, 2000). Megmutatta, hogy valójában számos eset van, amikor a kauzalitás kideríthető, és **kauzális háló (causal network)** formalizmus alakítható ki az oksági kapcsolatok, a beavatkozás hatásának, valamint a szokásos feltételes valószínűségek beépítésére.

A legközelebbi-szomszéd modellek legalább Fix és Hodges (Fix és Hodges, 1951) munkájáig nyúlnak vissza, és azóta a statisztika és alakfelismerés standard eszközei. Az MI-n belül Stanfill és Waltz népszerűsítették ezeket a modelleket (Stanfill és Waltz, 1986), ők a távolságmetrika adatokhoz történő adaptálási módszereivel foglalkoztak. Hastie és Tibshirani kifejlesztettek egy módszert, amellyel a tér egyes pontjaihoz kötötték az ezen pont körülü adateloszlástól függő metrikát (Hastie és Tibshirani, 1996). A legközelebbi-szomszédok hatékony indexelési sémával történő megtalálásával az algoritmusokat kutató közösség foglalkozott (pl. Indyk, 2000). A kernelsűrűség-becslést, amelyet **Parzen ablak (Parzen window)** sűrűségbecslésnek is neveznek, kezdetben Rosenblatt és Parzen tanulmányozta (Rosenblatt, 1956; Parzen, 1962). Azóta óriási az irodalma a különböző becslők tulajdonságai vizsgálatának. Devroye alapos bevezetést nyújt ehhez a témahez (Devroye, 1987).

A neurális hálók irodalma túl nagy ahhoz (napjainkig kb. 100 000 publikáció), hogy részletesen bemutathassuk. A korai fejleményekről Cowan és Sharp készített összefoglalót (Cowan és Sharp, 1988b; 1988a), McCulloch és Pitts munkásságával kezdve az áttekintést (McCulloch és Pitts, 1943). Norbert Wiener – a kibernetika és a szabályozáselmélet egyik úttörője (Wiener, 1948) – együttes dolgozott McCullochkal és Pittssel, és nagy hatást gyakorolt egy sor fiatal kutatóra, például Marvin Minskyre, aki valószínűleg elsőként fejlesztett ki működő neurális háló-hardvert 1951-ben (Minsky és Papert, 1988, pp. ix–x.) Ezalatt Nagy-Britanniában W. Ross Ashby (szintén a kibernetika egyik úttörője; Ashby, 1940), Alan Turing, Grey Walter és mások megalakították a Ráció Klubot (Ratio Club) azok számára, akik „rájöttek Wiener gondolataira, még

mielőtt Wiener könyve megjelent". Ashby *Az agy felépítése (Design for a Brain, 1948, 1952)* című könyvében vetette fel, hogy stabil adaptív viselkedést létrehozó alkalmas visszacsatoló hurkokkal rendelkező **homeostatikus**<sup>17</sup> (*homeostatic*) eszközök segítségével intelligenciát lehetne létrehozni. Turing egy kutatási jelentést írt *Intelligens Gépek (Intelligent Machinery)* címen (Turing, 1948), amely a következő mondattal kezdődik: „Javaslom megvizsgálni azt a kérdést, hogy vajon lehetséges-e az, hogy a gépek intelligens viselkedést mutassanak”, majd leírja a rekurrens neurális hálózatokat, amelyeket „B típusú nem szervezett gépek” néven vezet be, és megadja tanításuknak egy lehetséges megközelítését. Sajnos ezt a jelentést 1969-ig nem is publikálták, és napjainkig lényegében figyelmen kívül hagyták.

Frank Rosenblatt (Rosenblatt, 1957) nevéhez fűződik a modern „perceptron” felfedezése, és ő bizonyította be a perceptronkonvergencia tételet (Rosenblatt, 1960), bár ezt már a neurális hálók területén kívül eső, tisztán matematikai munkák is előrevetítették (Agmon, 1954; Motzkin és Schoenberg, 1954). Volt némi, a többrétegű hálózatokra irányuló korai kutatás is, amelynek eredményei például a **Gamba-perceptronok** (Gamba és társai, 1961) és a **madaline**-ok (Widrow, 1962). A *Learning Machines* (Nilsson, 1965) c. könyv áttekintést ad a korai kutatás legnagyobb részéről. A korai perceptronkutatások halálát siettette – a szerzők későbbi állítása szerint csak magyarázza – a *Perceptrons* c. könyv (Minsky és Papert, 1969), amelyben a terület matematikai precizitásának hiányát panaszolták fel. A könyv rámutatott, hogy egyrétegű perceptronokkal csak lineárisan szeparálható helyzetek reprezentálhatók, és felhívta a figyelmet a többrétegű hálók hatékony tanuló algoritmusainak hiányára.

A San Diegóban, 1979-ben tartott konferencia publikációira alapozott kiadvány (Hinton és Anderson, 1981) tekinthető a konnektionizmus újjáéledése jelének. Nagy figyelmet keltett a kétrészes „PDP” (Párhuzamos elosztott feldolgozás – Parallel Distributed Processing) antológia (Rumelhart és társai, 1986a), illetve a *Nature*-ben megjelent rövid cikk (Rumelhart és társai, 1986b). A neurális hálókkal foglalkozó cikkek száma az 1980–1984 közötti publikációk számáról 200-szorosára nőtt 1990–1994-re. A mágneses spin üvegek fizikai elméletének felhasználásával elvégzett neurális háló analízis (Amit és társai, 1985) nem csupán szorosabb kapcsolatot hozott a statisztikus mechanika és neurális hálók elmélete közt, hanem *tekintélyt* is adott a területnek. A hiba-visszaterjesztés (back-propagation) technikáját viszonylag hamar kitalálták (Bryson és Ho, 1969), de több alkalommal újra felfedezték (Werbos, 1974; Parker, 1985).

Az 1990-es években megjelent szupport vektor gépeknek (Cortes és Vapnik, 1995) napjainkban gyorsan növekvő az irodalmuk, amely olyan tankönyveket is magában foglal, mint Cristianini és Shawe-Taylor könyve (Cristianini és Shawe-Taylor, 2000). Nagyon népszerűnek és bizonyos feladatokra nagyon hatékonynak bizonyultak, ilyenek például a szövegkategorizálás (Joachims, 2001), a bioinformatikai kutatás (Brown és társai, 2000), a természetes nyelvű szöveg feldolgozása, mint a kézzel írt számjegyek DeCoste és Schölkopf által megvalósított felismerése (DeCoste és Schölkopf, 2002). A szavazó perceptron szintén egy olyan technika, amely a kerneltrükköt alkalmazza az exponenciális tulajdonságtér implicit reprezentációjára (Collins és Duffy, 2002).

<sup>17</sup> *Homeosztázis*: egy szervezet különböző, de egymással kölcsönhatásban álló elemei vagy elemek csoportjai között fennálló stabil egyensúly vagy egyensúly felé való törekvés. Gyakran biológiai összefüggésben használt kifejezés. (A ford.)

Számos forrás adható meg a neurális hálók valószínűségi interpretációjára, például (Baum és Wilczek, 1988), valamint (Bridle, 1990). A szigmoid függvény szerepét Jordan tárgyalja (Jordan, 1995). MacKay javasolta a neurális hálók Bayes-i paraméterbecslését (MacKay, 1992), amelyet Neal fejlesztett tovább (Neal, 1996). A neurális hálók függvény-reprezentációs képességeit Cybenko vizsgálta (Cybenko, 1988; 1989), aki megmutatta, hogy két rejtett réteg elegendő tetszőleges függvény reprezentációjához, tetszőleges folytonos függvény reprezentációjához pedig elég egy réteg. A haszontalan összekötetések eltávolítását célzó „optimális agykárosodás” módszer LeCun és társai eredménye (LeCun és társai, 1989), míg Sietsma és Dow mutatták meg, hogyan kell a felesleges neuronokat eltávolítani (Sietsma és Dow, 1988). A nagyobb struktúrák növesztéssel való előállítására szolgáló csempézési algoritmus Mézard és Nadal munkája (Mézard és Nadal, 1989). A kézzel írt számjegyek felismerésével foglalkozó algoritmusokról LeCun és társai írtak áttekintő publikációt (LeCun és társai, 1995). Azóta jobb hibaárányt értek el Belongie és társai (2002) az alakillesztési eljárás (Belongie és társai, 2002), valamint DeCoste és Schölkopf a virtuális szupport gép alkalmazásával (DeCoste és Schölkopf, 2002).

A számítógépes tanulás elmélete területén tevékenykedő kutatók foglalkoztak a neurális háló tanulás komplexitásával. Az első számítási eredményeket Judd kapta (Judd, 1990), aki megmutatta, hogy egy példahalmazzal konzisztens súlyhalmaz megtalálásának általános problémája – még nagyon erősen korlátozó feltételek esetén is – NP-teljes. A mintakomplexitásra vonatkozó eredmények közül néhány Baum és Haussler munkájához fűződik, akik megmutatták, hogy  $W$  súly esetén a hatékony tanításhoz szükséges mintaszám  $W \log W$  arányában nő (Baum és Haussler, 1989).<sup>18</sup> Azóta Anthony és Bartlett egy sokkal fejlettebb elméletet fejlesztettek ki (Anthony és Bartlett, 1999), amely magában foglalja azt a fontos eredményt, hogy a háló reprezentációs képessége a súlyok *nagy-ságrendjétől* és számától egyaránt függ.

Az általunk nem tárgyalt legnépszerűbb neurális háló a **radiális bázisfüggvény** (RBF) (**radial basis function**) háló. A radiális bázisfüggvény kernelek súlyozott kombinációját (természetesen rendszerint Gauss-kernelekét) alkalmazza függvényapproximációra. Az RBF-hálókat két fázisban taníthatjuk: először egy nem ellenőrzött osztályozással tanítjuk a Gauss-függvények paramétereit (az átlagokat és varianciákat), mint a 20.3. alfejezetben láttuk. A második fázisban a Gauss-függvények relatív súlyát határozzuk meg. Ez egy lineáris egyenletrendszer megoldása, amiről tudjuk, hogy közvetlenül hogyan oldható meg. Így az RBF-tanítás minden fázisának vonzó tulajdonságai vannak: az első fázis nem ellenőrzött, tehát nincs szükségünk hozzá címkezett mintákra, a második ugyan feliügyelt, de hatékonyan elvégezhető. A részleteket lásd Bishop publikációjában (Bishop, 1995).

A **rekurrens hálókat** (**recurrent network**), amelyekben a neuronok hurkokba vannak kapcsolva, említettük ugyan a fejezetben, de nem részleteztük. A legjobban meg-értekk rekurrens háló osztályt valószínűleg a **Hopfield-hálók** (**Hopfield networks**) (Hopfield, 1982) alkotják. A Hopfield-háló kétirányú kapcsolatokat használ *szimmetrikus* súlyokkal (azaz  $W_{i,j} = W_{j,i}$ ), minden neuron bemeneti és egyben kimeneti egység is, a  $g$  aktivációs függvény az előjelfüggvény, az aktivációs szint pedig csak a  $\pm 1$  lehet. A Hopfield-háló **asszociatív memóriaként** működik: miután egy mintahalmazon taní-

<sup>18</sup> Ez nagyából igazolta „Bernie bácsi” törvényét: a szabályt Bernie Widrowról nevezték el, aki azt javasolta, hogy nagyából tízszer annyi mintát használjanak, mint a súlyok száma.

tottuk, egy új bemeneti stimulus hatására egy olyan aktivációs mintára áll be, amely a tanító példák közül a *leginkább emlékezett* az új bemenetre. Ha például a tanító halmaz egy halom fénykép, az új bemenet pedig az egyik fénykép egy kis darabja, akkor aktivációs szintjeivel a háló vissza fogja állítani az adott darabból a fényképet. Figyeljük meg, hogy a fényképek nincsenek elkitünlően tárolva a hálóban: minden egyik súly az összes fénykép egy részleges kódja. Érdekes elméleti eredmény, hogy az  $N$  neuronból álló Hopfield-háló legfeljebb  $0,138N$  tanító mintát képes megbízhatóan tárolni.

A Boltzmann-gépek is szimmetrikus súlyokat használnak, de tartalmaznak rejttett egységeket is (Hinton és Sejnowski, 1983; 1986). Ráadásul aktivációs függvényük *sztocchasztikus*: a kimenet 1 értékének valószínűsége a teljes súlyozott bemenetnek valamilyen függvénye. A Boltzmann-gépek ennek megfelelően a szimulált lehűtés keresési eljárásra (lásd 4. fejezet) emlékezetető állapotámenetek során jutnak el a tanító halmazt a legjobban közelítő konfigurációhoz. Kimutatható, hogy a Boltzmann-gépek nagyon közeli rokonságban vannak bizonyos speciális Bayes-hálókkal, amelyeket sztochasztikus szimulációs algoritmussal értékelünk ki. (Lásd 14.5. alfejezet.)

A kernelgépek alapötleteinek első alkalmazása Aizerman és társai munkájához fűződik (Aizerman és társai, 1964, de a szupport vektor gépek felé mutató, teljes elméleti kidolgozás Vlagyimir Vapniknak és kollégáinak köszönhető (Boser és társai, 1992; Vapnik, 1998). A precíz elméleti tárgyalás megtalálható Cristianini és Shawe-Taylor, illetve Schölkopf és Smola könyvében (Cristianini és Shawe-Taylor, 2000; Schölkopf és Smola, 2002), valamint egy barátságosabb kifejtés található Cristianini és Schölkopf *AI Magazine*-ban megjelent cikkében.

Ennek a fejezetnek az anyaga a statisztika, alakfelismerés és neurális hálók kutatásában elért eredményeket integrálja, tehát a történetet sokszor, sokféle módon elmondták már. A Bayes-statisztikáról jó publikációkat találunk, ilyenek például a (DeGroot, 1970), a (Berger, 1985), továbbá a (Gelman és társai, 1995). A statisztikus tanulás elméletének kiváló tárgyalása található a (Hastie és társai, 2001) munkában. Az alakosztályozás téma körében Duda és Hart könyve számít klasszikusnak (Duda és Hart, 1973), melynek nemrég új kiadása is megjelent (Duda és társai, 2001). A neurális hálókról a (Bishop, 1995) és a (Ripley, 1996) a legfontosabb könyvek. A számítógépes neurális tudományt a (Dayan és Abbott, 2001) könyv tárgyalja. A neurális hálók és hozzájuk kapcsolódó témaik legfontosabb konferenciái az évente megrendezett NIPS- (Neural Information Processing Conference) konferenciák, melyek kiadványait az *Advances in Neural Information Processing Systems* sorozatban jelentetik meg. A Bayes-hálók tanulásáról szóló írások az *Uncertainty in AI* és a *Machine Learning* konferenciákon jelennek meg, továbbá számos statisztikával foglalkozó konferencián. A neurális hálókkal foglalkozó újságok között érdemes megemlíteni a *Neural Computation*, a *Neural Networks*öt és az *IEEE Transaction on Neural Networks*öt.

## Feladatok

- 20.1.** A 20.1. ábrán használt adatokat úgy tekinthetjük, mintha  $h_5$  generálta volna őket. A másik négy hipotézis mindegyikére generáljon egy-egy 100 hosszúságú adathalmazt, rajzolja fel a megfelelő  $P(h_i|d_1, \dots, d_m)$  és  $P(D_{m+1} = \text{citrom}|d_1, \dots, d_m)$  görbékét! Értékelje az eredményeket.

- 20.2.** Ismételje meg a 20.1. feladatot úgy, hogy a  $P(D_{m+1} = \text{citrom}|h_{\text{MAP}}, \dots, d_m)$ , illetve  $P(D_{m+1} = \text{citrom}|h_{\text{ML}})$  görbéket rajzolja fel.
- 20.3.** Tegyük fel, hogy Anna számára a meggycukorkák hasznossága  $c_A$ , a citromcukorkáké  $c_B$ , míg Béla számára a hasznosságok rendre  $c_B$  és  $c_A$ . (De ha Anna kibontott egy cukorkát, akkor azt Béla már nem fogja megvásárolni tőle.) Ha Béla sokkal jobban szereti a citromízű cukorkát, mint Anna, akkor Anna feltehetőleg bölcsen teszi, ha eladja a cukros zacskóját, amikor már kellően biztos a citromízűek arányában. Másrészt viszont, ha Anna túl sok cukrot kibontott már az arány eldöntése során, akkor a zacska értéktelennek válik. Vizsgálja meg a cukroszacskó-eladás optimális pillanatának meghatározását. Határozza meg az optimális eljárás várható hasznosságát, a 20.1. alfejezet a priori eloszlásainak feltételezésével.
- 20.4.** Két statisztikus elmegy az orvoshoz, és mindenkorról ugyanazt a diagnózist állítja fel az orvos: 40% eséllyel a halálos  $A$  betegségben, 60% eséllyel a szintén végzetes  $B$  betegségben szenvednek. Szerencsére van az  $A$  és  $B$  betegségnek is olcsó, 100%-ban hatásos, mellékhatás nélküli gyógyszere. A statisztikusoknak lehetőségük van az  $A$  elleni, a  $B$  elleni vagy minden gyógyszert szedni, de döntethetnek úgy is, hogy egyiket sem szedik. Mit fog az első statisztikus – aki elszánt Bayes-hívő – választani? Mit tesz a második, aki minden a maximum-likelihood hipotézist választja?
- Az orvos némi kutatás után felfedezi, hogy a  $B$  betegségnek két változata van, a dextro- $B$  és a levo- $B$ , amelyek egyforma valószínűséggel lépnek fel, és egyformán jól kezelhetők az anti- $B$  gyógyszerrel. Mit fog csinálni a két statisztikus most, hogy három hipotézis van?
- 20.5.** Magyarázza meg, hogyan alkalmazható a 18. fejezetben tárgyalt turbózás módszere a naiv Bayes-tanulásra. Tesztelje a kapott algoritmust az étterem tanulási problémán.
- 20.6.** Vegyen  $m$  darab  $(x_j, y_j)$  adatpontot, az  $x_j$ -ből a (20.5) egyenlet alapján generálja  $y_j$ -t. Keresse meg azokat a  $\theta_1$ ,  $\theta_2$  és  $\sigma$  értékeket, amelyek maximálják az adatok feltételes log likelihood értékét.
- 20.7.** Vizsgálja a láz 14.3. alfejezetben bemutatott zajos-VAGY modellt. Magyarázza meg, hogyan alkalmazható a maximum-likelihood tanulás arra, hogy egy teljes adathalmazra illesszük egy ilyen modell paramétereit. (Segítség: a parciális deriváltakra használja a láncszabályt.)
- 20.8.** Ebben a feladatban a (20.6) egyenlettel definiált béta-eloszlások tulajdonságait vizsgáljuk.
- A  $[0, 1]$  tartomány feletti integrálás alapján mutassa meg, hogy a béta $[a,b]$  eloszlás normalizáló faktorát  $\alpha = \Gamma(a+b)/\Gamma(a)\Gamma(b)$  adja, ahol  $\Gamma(x)$  az úgynevezett **gamma-függvény**, amelynek definíciója:  $\Gamma(x+1) = x \times \Gamma(x)$  és  $\Gamma(1) = 1$ . (Egész  $x$ -ekre  $\Gamma(x+1) = x!$ )

- (b) Mutassa meg, hogy az átlagérték  $a/(a + b)$ .  
 (c) Állítsa elő a modus(okat) (a legvalószínűbb  $\theta$  értéke(ke)t).  
 (d) Jellemesse a béta[ $\epsilon$ ,  $\epsilon$ ] eloszlást nagyon kis  $\epsilon$  esetén. Mi történik, ha egy ilyen eloszlást frissítünk?
- 20.9.** Vegyen egy tetszőleges Bayes-hálót, egy ehhez a hálóhoz tartozó teljes adathalmazt és az adathalmaznak a háló által megadott likelihood-értékét. Adjon egy-szerű bizonyítást arra, hogy az adathalmaz likelihood-értéke nem csökkenhet, ha a hálóhoz hozzáunk egy új kapcsolatot, majd újraszámoljuk a maximum-likelihood paramétereket.
- 20.10.** Vizsgálja meg az EM alkalmazását arra a problémára, amikor a 20.10. (a) ábrán látható háló paramétereit akarjuk tanulni. miközben az igazi paramétereket a (20.7) egyenlet adja.  
 (a) Magyarázza meg, hogy miért nem működne az EM algoritmus, ha csak két attribútum volna, és nem három.  
 (b) Végezze el az első iterációra vonatkozó számítást a (20.8) egyenletből kiindulva.  
 (c) Mi történik, ha induláskor az összes paramétert ugyanarra a  $p$  értékre állítjuk? (*Segítség:* hasznos lehet, ha empirikusan megvizsgálja, mielőtt az általános eredményt megpróbálná levezetni.)  
 (d) Írja fel a 836. oldalon található adattáblázatra a log likelihoodnak a paraméterekkel kifejezett összefüggését. Számítsa ki minden egyes paramétere a parciális derivált értékét! Vizsgálja meg a (c) részben rögzített pont jellegét.
- 20.11.** Hozzon létre manuálisan egy olyan neurális hálót, amely két bemenetének az XOR függvényét számítja ki. Bizonyosodjon meg arról, helyesen specifikálta-e azt, hogy milyen egységeket kell használni.
- 20.12.** Hozzon létre egy olyan szupport vektor gépet, amely az XOR függvényt számítja ki. Kényelmesebb lesz, ha mind a bemeneteknél, mind a kimenetnél  $-1$ -et és  $+1$ -et használ  $0$  és  $+1$  helyett. Ennek megfelelően egy minta például  $([-1, 1], 1)$  vagy  $([-1, -1], -1)$  lesz. Tipikus, hogy az  $x$  bemenetet egy ötdimenziós térré képezzük le, ahol kettő az eredeti  $x_1$  és  $x_2$  dimenzió, a másik három pedig  $x_1^2$ ,  $x_2^2$ , illetve  $x_1 x_2$  kombinációi. Ebben a feladatban viszont csak két dimenziót használunk:  $x_1$ -et és  $x_1 x_2$ -t. Rajzolja fel ebben a térben a négy bemeneti pontot és a maximális tartalékkal rendelkező osztályozót. Mekkora a tartalék? Rajzolja meg a határoló vonalat az eredeti, euklideszi térben.
- 20.13.** Egy egyszerű perceptron nem képes az XOR függvény (vagy általánosabban a paritásfüggvény) ábrázolására. Mutassa be, hogy hogyan alakulnak egy négy-bemenetű, ugrásfüggvényt használó perceptron súlyai, amikor sorban a paritásfüggvényből származó minták érkeznek (kezdetben minden súly 0,1 értékű volt).

**20.14.** Idézzük fel a 18. fejezetből, hogy  $n$  bemenet esetén  $2^n$  különböző logikai függvény létezik. Hányat tud reprezentálni ezek közül az ugrásfüggvényt használó perceptron?

**20.15.** Vegyük a következő – hat bemeneti értéket ( $I_1 \dots I_6$ ) és egy, ezekhez tartozó kiáltványt kimeneti értéket ( $T$ ) megadó – mintahalmazt.

$I_1$	1	1	1	1	1	1	0	0	0	0	0	0	0
$I_2$	0	0	0	1	1	0	0	1	1	0	1	0	1
$I_3$	1	1	1	0	1	0	0	1	1	0	0	0	1
$I_4$	0	1	0	0	1	0	0	1	0	1	1	1	0
$I_5$	0	0	1	1	0	1	1	0	1	1	0	0	1
$I_6$	0	0	0	1	0	1	0	1	1	0	1	0	1
$T$	1	1	1	1	1	1	0	1	0	0	0	0	0

- (a) Futassa a perceptron tanulási szabályt erre az adathalmazra, és adja meg a végső súlyokat.
- (b) Futassa a döntési fa tanulási algoritmust, és adja meg az eredményül kapott döntési fát.
- (c) Értékelje az eredményeket.

**20.16.** A (20.13) egyenletből kiindulva mutassa meg, hogy  $\partial L / \partial W_j = Err \cdot x_j$ .

**20.17.** Tegyük fel, hogy egy lineáris aktivációs függvényeket használó neurális hálón van. Azaz minden neuron kimenete a bemenetek súlyozott összegének  $c$ -szerese, ahol  $c$  egy konstans.

- (a) Tegyük fel, hogy a hálónak egy rejtett rétege van. Írja fel a súlyok egy adott  $W$  értékkészletére azokat az egyenleteket, amelyek megadják a kiemeneti réteg neuronjainak kimeneti értékeit  $W$  és a bemeneti réteg  $I$  értékeinek függvényében – anélkül hogy explicit módon megjelennének az egyenletekben a rejtett neuronok kimeneti értékei. Mutassa meg, hogy létezik olyan háló, amelynek nincs rejtett rétege, de ugyanezt a függvényt valósítja meg.
- (b) Ismételje meg az (a) rész alatti feladatot, de tetszőleges számú rejtett réteg esetére. Milyen következtetést vonhat le a lineáris aktivációs függvényekre?

**20.18.** Hozzon létre egy adatstruktúrát rétegekbe szervezett, előrecsatolt neurális hálók leírására, amely tartalmazza az előreterjesztéshez és a hiba-visszaterjesztéshez szükséges információt is. Ezt használva írjon egy NEURÁLIS-HÁLÓ-KIMENET nevű programot, amely egy mintát és egy neurális hálót kap bemenetként, és kiszámítja a neurális hálónak a mintára adott kimeneti válaszát.

**20.19.** Tegyük fel, hogy egy tanító halmaz csupán egyetlen példát tartalmaz, de azt 100-szor. A 100 esetből 80-ban az egyetlen kimeneti érték 1; a másik 20-ban 0.

Mit ad erre a példára egy hiba-visszaterjesztéssel tanított háló, ha tanítottuk, és elérte a globális optimumot? (*Segítség: a globális optimum megtalálásához differenciálja a hibafüggvényt, és keresse a nullahelyét.*)

- 20.20.** A 20.24. ábrán látható hálónak négy rejtett neuronja van. Ezt a neuronszámot némi képpen ötletszerűen választottuk. Végezzen szisztematikus kísérleteket, hogy különböző számú rejtett neuronnal rendelkező hálókra lemérje a tanulási görbét. Mi az optimális neuronszám? Lehetséges lenne keresztavalidációs módszerrel megtalálni a legjobb hálót még a kísérletek előtt?
- 20.21.** Vegyük azt a problémát, amikor  $N$  adatpontot akarunk lineáris osztályozóval szétválasztani pozitív és negatív példákra. Nyilvánvaló, hogy  $N = 2$  pont esetén ez egy  $d = 1$  dimenziós egyenesen minden megtehető, függetlenül attól, hogy a pontok hol helyezkednek el, és hogy vannak címkézve (kivéve, ha ugyanazon a helyen vannak).
- Mutassa meg, hogy a szeparálás minden elvégezhető  $N = 3$  pontra egy  $d = 2$  dimenziós síkon, ha a pontok nem egy egyenesre esnek.
  - Mutassa meg, hogy nem lehet minden elvégezni a szeparálást  $N = 4$  pontra egy  $d = 2$  dimenziós síkon.
  - Mutassa meg, hogy a szeparálás minden elvégezhető  $N = 4$  pontra egy  $d = 3$  dimenziós térben, ha a pontok nem esnek egy síkra.
  - Mutassa meg, hogy nem lehet minden elvégezni a szeparálást  $N = 5$  pontra egy  $d = 3$  dimenziós térben.
  - Egy törekvő diák kitűzheti maga elé, hogy bebizonyítja:  $N - 1$  dimenziós térben általános helyzetben található  $N$  pont lineárisan szeparálható (de  $N + 1$  nem). Ebből következik, hogy az  $N - 1$  dimenziós lineáris félterek **VC-dimenziója** (lásd 18. fejezet)  $N$ .

# 21. MEGERŐSÍTÉSES TANULÁS

Ebben a fejezetben azt vizsgáljuk, hogy egy ágens hogyan képes tanulni a sikereiből és a kudarcaiból, illetve a jutalomból és a büntetésből.

## 21.1. BEVEZETÉS

A 18. és 20. fejezetben olyan tanuló módszereket mutattunk be, amelyek példák alapján függvényeket vagy valószínűségi modelleket sajátítanak el. Ebben a fejezetben azonban fogalkozunk, hogy az ágens mi módon tanulhatja meg azt, hogy *mit tegyen*, különösen akkor, amikor nincs tanár, aki minden előforduló körfülmény esetére elárulná a helyes cselekvést.

Vegyük példaként a sakkot. Tudjuk, hogy az ágens képes megtanulni sakkozni felügyelt tanulással, vagyis ha példaként sakkállásokat mutatunk neki, megnevezve az adott állásban a legjobb lépést. De mit tud csinálni az ágens akkor, ha nincs kéznél egy barátságos tanító, aki példákat mutatna? Véletlen lépések kipróbálásával végül is az ágens felépítheti környezetének prediktív modelljét: milyen lesz az állás, ha meglép egy adott lépést, esetleg még azt is modellezheti, hogy az ellenfél váratlanul mit fog válaszul lépni. A következő azonban a probléma: *visszacsatolás nélkül – tehát anélkül, hogy tudná, mi a jó és mi a rossz – az ágensnek semmilyen alapja sem lesz eldöntenı azt, hogy melyik lépést húzza meg*. Az ágensnek tudnia kell, hogy valami jó történt, amikor nyert, illetve valami rossz történt, amikor vesztett. Az ilyen típusú visszacsatolást nevezzük **jutalomnak (reward)** vagy **megerősítésnek (reinforcement)**. A sakkhoz hasonló játékban a megerősítés csak a játék végén jelenik meg. Más környezetben a jutalom gyakrabban jön, például a pingpongban mindenkor pont jutalomnak tekinthető éppúgy, mint a mászás tanulásakor bármely előrefelé haladás. Az általunk használt tárgyalásmódban az ágens a jutalmat is az észlelés részeként kezeli, de valahogy bele kell lennie „huzalozva”, hogy ez a rész jutalom, és nem csupán egy észlelésfajta. Úgy tűnik, hogy az állatokba be van építve, hogy a fájdalom és az éhség negatív „jutalom”, míg a gyönyör és az élelem pozitív jutalom. A megerősítéses tanulást az állatok pszichológiáját kutatók 60 éve intenzíven tanulmányozzák.

A jutalom fogalmát a 17. fejezetben vezettük be, ahol a **Markov döntési folyamatok** (**Markov decision process, MDP**) optimalitásának definiálására szolgált. Optimális stratégiának neveztük azt a stratégiát, amely maximálja a várható összjutalmat. A **megerősítéses tanulás (reinforcement learning)** feladata az, hogy optimális (vagy közel optimális) stratégiát tanuljunk az adott környezethez. Míg a 17. fejezetben az ágensnek a teljes környezeti modell és a jutalomfüggvény a rendelkezésére állt, addig a jelen fejezetben egyikről sem feltételezünk semmilyen előzetes tudást. Képzelje el, hogy egy új, ismeretlen játékokat játszik, amelynek nem ismeri a szabályait. Nagyjából száz lépés után az ellenfele bejelenti: ön vesztett. Dióhéjban ez jellemzi a megerősítéses tanulást.



Számos bonyolult problématerületen a megerősítéses tanulás az egyetlen lehetséges út arra, hogy egy programnak magas szintű működést tanítsunk. Például a játékok terén nagyon nehéz lenne a szakember dolga: a lehetséges nagyszámú állás pontos és következetes értékelését kellene elvégezniue annak érdekében, hogy a példák alapján közvetlenül meg-tanítsa egy kiértékelő függvényt. Ehelyett a programnak megadható az az információ, hogy nyert vagy vesztett, és ezt felhasználhatja egy olyan kiértékelő függvény tanulására, amely elfogadható pontossággal képes megbecsülni bármely adott állásban a nyerés valószínűségét. Hasonlóképpen rendkívül nehéz megtanítani egy ágenst helikopter vezetésére, de ha megfelelő negatív jutalmat kap minden balesetnél, bukdácsoló mozgásnál vagy a beállított útvonaltól való eltérésnél, akkor az ágens önállóan megtanulhat repülni.

Bizonyos értelemben a megerősítéses tanulás az egész ML-problémát átfogja. Egy ágenst belehelyezünk valamelyen környezetbe, és meg kell tanulnia ott sikeresen viselkedni. Ahhoz, hogy ezt a fejezetet áttekinthető bonyolultsági szinten tartsuk, csak egyszerű helyzetekre és egyszerű ágensekre fogunk koncentrálni. A fejezet legnagyobb részében teljesen megfigyelhető környezetet feltételezzük, tehát a megfigyelések informálnak az aktuális állapotról. Másrészt viszont azt feltételezzük, hogy az ágens sem azt nem tudja, hogyan működik a környezet, sem azt, hogy cselekedeteinek mi a hatása, továbbá megengedjük a cselekedetek valószínűségi (nemdeterminisztikusan meghatározott) kimenetelét. Három olyan ágensfelépítést fogunk áttekinteni, amelyeket először a 2. fejezetben vezettünk be:

- **A hasznosságalapú ágens (utility-based agent)** az állapotokra alapozott hasznosságfüggvényt tanul, és ennek alapján választja ki azokat a cselekvéseit, amelyekkel maximálja az elérhető hasznosság várható értékét.
- **A Q-tanuló (Q-learning) ágens** egy **cselekvésérték (action-value)** függvényt – vagy más néven *Q*-függvényt – tanul, valamilyen várható hasznöt tulajdonítva egy adott helyzetben egy adott cselekvésnek.
- **Egy reflexszerű ágens (reflex agent)** olyan stratégiát tanul, amely közvetlenül képezi le az állapotokat cselekvésekre.

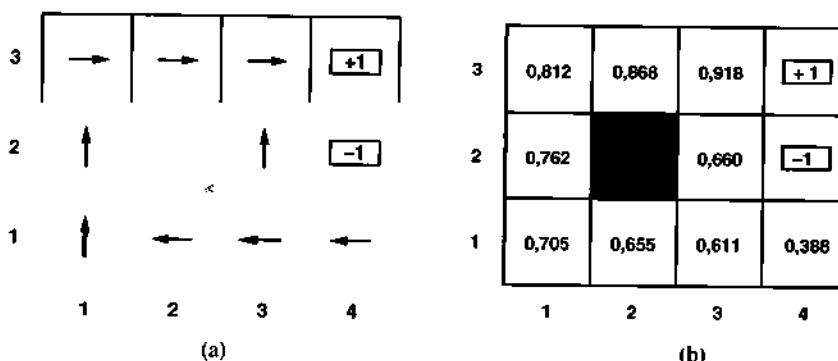
Egy hasznosságalapú ágensnek a környezet valamelyen modelljével is rendelkeznie kell ahhoz, hogy tudja, cselekvése milyen állapotba vezet. Például ahhoz, hogy az ostáblajáték hasznosságfüggvényének hasznát vegyük, az ostáblajátékot játszó programnak tudnia kell, hogy melyek a lehetséges szabályos lépések, és ezek *hogyan hatnak az állásra*. Csak ilyen módon van arra lehetősége, hogy a lépése eredményeképpen kapott állásra alkalmazza a hasznosságfüggvényt. Egy *Q*-tanuló ágens viszont össze tudja hasonlítani a lehetséges választásait anélkül, hogy tudná, mire vezetnek, így nincs szüksége a környezet modelljére. Másrészt viszont, mivel nem tudják, hogy cselekvéseik hova vezetnek, a *Q*-tanuló ágensek nem képesek előre nézni. Ez viszont, mint látni is fogjuk, súlyosan korlátozhatja tanulási képességeiket.

A 21.2. alfejezetben a **passzív tanulással (passive learning)** kezdünk foglalkozni, amelyben az ágensnek rögzített stratégiája van, és az állapotok hasznosságának (vagy az állapot-cselekvés párok hasznosságának) megtanulása a feladat. Ez magában foglalhatja a környezet modelljének megtanulását is. A 21.3. alfejezet az **aktív tanulással (active learning)** foglalkozik, amikor is az ágensnek azt is meg kell tanulnia, hogy mit tegyen. Az alapvető elv a **selffedezés (exploration)**, az ágensnek a lehető legtöbbet meg kell tapasztalnia környezetéről ahhoz, hogy megtanulja, mi a célszerű viselkedés benne. A 21.4. alfejezet bemutatja, hogy az ágens hogyan tudja felhasználni az induktív

tanulást arra, hogy sokkal gyorsabban tanuljon a tapasztalataiból. A 21.5. alfejezet a közvetlen stratégiareprezentáció tanulásának módszereivel foglalkozik reflexszerű ágenseknél. A Markov döntési folyamatok (lásd 17. fejezet) megértése döntő fontosságú ennek a fejezetnek a tárgyalása szempontjából.

## 21.2. PASSZÍV MEGERŐSÍTÉSES TANULÁS

Az egyszerűség kedvéért egy ismert, teljes mértékben hozzáférhető környezet állapot-reprezentációját használó passzív ágens esetét tárgyaljuk először. Passzív tanulás esetén az ágens  $\pi$  stratégiája rögzített, az  $s$  állapotban minden a  $\pi(s)$  cselekvést hajtja végre. A cél egyszerűen a stratégia jóságának – tehát az  $U^\pi(s)$  hasznosságfüggvénynek – a megtanulása. Példaként a 17. fejezetben bevezetett  $4 \times 3$ -as világot fogjuk használni. A 21.1. ábra mutatja ezt a világot, és a megfelelő hasznosságokat. Nyilvánvaló, hogy a passzív tanulás hasonló a **stratégiakiértékelési** (policy evaluation) feladathoz, amely része a 17.3. alfejezetben ismertetett **stratégiaiterációs** (policy iteration) algoritmusnak. A legfontosabb különbség, hogy a passzív ágens nem ismeri az **állapotátmenet-modellt** (transition model), a  $T(s, a, s')$ -t, amely annak a valószínűséget adja meg, hogy az  $a$  cselekvés hatására az  $s$  állapotból az  $s'$  állapotba jutunk, továbbá nem ismeri a **jutalomfüggvényt** (reward function),  $R(s)$ -et, amely minden állapothoz megadja az ott elnyerhető jutalmat.



**21.1. ábra.** (a) A  $4 \times 3$ -as világban alkalmazott  $\pi$  stratégia. Ez optimális stratégia, ha nem alkalmazunk leírtékelést, és minden állapotban, amely nem végállapot,  $R(s) = -0,04$ . (b) A  $4 \times 3$ -as világ állapotainak hasznossága a  $\pi$  stratégia esetén.

Az ágens –  $\pi$  stratégiája felhasználásával – egy sor **kísérletet** (trial) végez. minden egyes kísérletben az (1,1) állapotból indul, és az állapotátmenetek valamelyen sorozatát észleli, amíg el nem érkezik a (4,2), illetve a (4,3) végállapotok valamelyikébe. Észlelései alapján információt kap mind az aktuális állapotról, mind az ott nyert jutalomról. Például így nézhetnek ki a tipikus kísérletek:

$$\begin{aligned}
 &(1,1)_{-0,04} \rightarrow (1,2)_{-0,04} \rightarrow (1,3)_{-0,04} \rightarrow (1,2)_{-0,04} \rightarrow (1,3)_{-0,04} \rightarrow (2,3)_{-0,04} \rightarrow (3,3)_{-0,04} \rightarrow (4,3)_{+1} \\
 &(1,1)_{-0,04} \rightarrow (1,2)_{-0,04} \rightarrow (1,3)_{-0,04} \rightarrow (2,3)_{-0,04} \rightarrow (3,3)_{-0,04} \rightarrow (3,2)_{-0,04} \rightarrow (3,3)_{-0,04} \rightarrow (4,3)_{+1} \\
 &(1,1)_{-0,04} \rightarrow (2,1)_{-0,04} \rightarrow (3,1)_{-0,04} \rightarrow (3,2)_{-0,04} \rightarrow (4,2)_{-1}
 \end{aligned}$$

Vegyük észre, hogy mindegyik állapotészlelés mellett – alsó indexként – feltüntettük az elnyert jutalmat. Célunk a jutalmakban rejő információ felhasználása arra, hogy az egyes állapotokhoz – amelyek nem végállapotok – rendelhető várható hasznosságot,  $U^\pi(s)$ -t megtanuljuk. A hasznosságot úgy definiáljuk, mint a  $\pi$  stratégia követése esetén az összegzett (leértékelt) jutalom várható értéke. Ez – ahogy a (17.3) egyenletben a 713. oldalon – már felírtuk:

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s \right] \quad (21.1)$$

Mindegyik összefüggésünkben figyelembe veszünk egy  $\gamma$  leértékelési tényezőt (discount factor), de a  $4 \times 3$  világban  $\gamma = 1$ -et használunk.

## Közvetlen hasznosságbecslés

Widrow és Hoff az adaptív szabályozáselmélettel (adaptive control theory) foglalkozva egy egyszerű módszert fedeztek fel a közvetlen hasznosságbecslésre (direct utility estimation) (Widrow és Hoff, 1960). Az ötlet a következő: az állapot hasznossága nem más, mint az adott állapotból kiindulva az összegzett jutalom várható értéke, és minden egyes kísérlet ennek az értéknek egy mintájával szolgál minden bejárt állapotra. Például az előzőben bemutatott háromelemű tipikus kísérlethalmaz első kísérlete egy 0,72 értékű összegzett jutalommintát ad az (1,1) állapotra, két – 0,76 és 0,84 értékű – mintát az (1,2) állapotra, továbbá két – 0,80 és 0,88 értékű – mintát az (1,3) állapotra és így tovább. Ennek megfelelően az algoritmus minden kísérleti lépéssorozat végén kiszámítja az összes állapotra a megfigyelt hátralevő jutalmat, és ez alapján frissíti az állapot hasznosságát, egyszerűen minden egyes állapotra mozgóablak átlagolást végezve, és az eredményt egy táblában tárolva. Ha a kísérletek száma a végtelenhez tart, a minták átlaga a (21.1) egyenettel megadott valódi várható értékhez fog tartani.

Nyilvánvaló, hogy a közvetlen hasznosságbecslés nem más, mint egy példánya a felügyelt tanításnak. A tanító példapárok bemeneti része az állapot, kimeneti része pedig a megfigyelt hátralevő jutalom. Tehát a megerősítéses tanulást standard induktív tanulási problémára redukáltuk, a 18. fejezetben tárgyaltnak megfelelően. A 21.4. alfejezetben áttekintjük a hasznosságfüggvények hatékonyabb reprezentálására szolgáló eszközök használatát, például a neurális hálókét. Ezeknek a reprezentációknak a használata esetén a tanulási technika közvetlenül a megfigyelt adatokra alkalmazható.

A közvetlen hasznosságbecsléssel sikerült a megerősítéses tanulást standard induktív tanulási problémára redukálnunk, amelyről már nagyon sok minden tudunk. Sajnálatos módon egy nagyon fontos információforrás hiányzik, nevezetesen az, hogy az állapotok hasznossága nem független egymástól! *Minden egyes állapot hasznossága egyenlő az ebben az állapotban elnyerhető jutalom és az őt követő állapotok várható hasznosságának összegével*. Tehát a hasznosságértékek kielégítik a rögzített stratégiára vonatkozó Bellman-egyenletet (lásd még (17.10) egyenlet):

$$U^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi(s') \quad (21.2)$$

Azzal, hogy figyelmen kívül hagyja az állapotok közötti összefüggéseket, a közvetlen hasznosság becslése eljátssza a tanulás lehetőségeit. Például a bemutatott három kísérlet közül a második érinti a (3,2) állapotot, amelyet előzőleg az ágens nem érintett még. A következő átmenet után (3,3)-ba jut, amely az első kísérlet alapján nagy hasznosságot mutat. A Bellman-összefüggés azonnal rámutat, hogy valószínűleg akkor a (3,2)-nek is nagy a hasznossága, mivel a (3,3)-ba vezet, de a közvetlen hasznosságbecslés semmit se tanul meg ebből a kísérlet végéig. Általánosabban megfogalmazva azt mondhatjuk, hogy a közvetlen hasznosságbecslésre úgy tekinthetünk, mint ami jóval nagyobb  $U$  hipotézis-térben keres, mint amire szükség van. Ennek oka, hogy számos olyan függvényt is számon tart a hipotézisek között, amelyek sértik a Bellman-egyenletet. Emiatt az algoritmus gyakran csak nagyon lassan konvergál.

## Adaptív dinamikus programozás

Ahhoz, hogy használhassuk az állapotok közti kényszerekben rejlő információt, az ágensnek meg kell tanulnia, hogy az állapotok között milyen kapcsolatok állnak fenn. Egy **adaptív dinamikus programozás (ADP)** (*adaptive dynamic programming, ADP*) alapú ágens működésének lényege, hogy menet közben megtanulja a környezet állapot-átmenet-modelljét, és dinamikus programozási módszerrel megoldja a hozzá kapcsolható Markov döntési folyamatot. Egy passzív ágens számára ez azt jelenti, hogy a megtanult  $T(s, \pi(s), s')$  állapotátmenet-modellt és a megfigyelt  $R(s)$  jutalmakat behelyettesíti a 21.2. Bellman-egyenletbe, majd kiszámítja az állapotok hasznosságát. Mint a 17. fejezetben a stratégiaiterációnál rámutattunk, ezek az egyenletek lineárisak (nincs maximalizáció lépés), így bármely lineáris algebrai programcsomaggal megoldhatók. Egy másik alternatíva, ha a **módosított eljárásmód-iterációt (modified policy iteration)** választjuk (lásd 720. oldal), egy egyszerűsített értékiterációs eljárást használva arra, hogy a tanult modell minden változása után frissítük a hasznosságbecslés eredményét. Mivel a modell legtöbbször csak keveset változik az egyes megfigyelések után, az értékiterációs algoritmus kiinduló értékként felhasználhatja az előző hasznosságbecslést, és egész gyorsan konvergálhat.

Magának a modellnek egyszerű a megtanulása, mivel a környezet teljesen megfigyelhető. Ennek megfelelően egy ellenőrzött tanítási feladatunk van, amelyben a bemenet az állapot-cselekvés pár, a kimenet pedig a következő állapot. A legegyszerűbb esetben az állapotátmenet-modellt valószínűségek táblázatával reprezentálhatjuk. Számon tartjuk, hogy a cselekvések egyes eredményei milyen gyakran lépnek fel, és abból a gyakoriságból becsljük a  $T(s, a, s')$  állapotátmenet valószínűségét, hogy háromszor következett  $s'$ , amikor az  $s$  állapotban az a cselekvést választottuk.<sup>1</sup> Például ha a 875. oldal három útvonalában a *Jobbra* cselekvést háromszor választottuk az (1,3) állapotban, és a háromból kétszer a (2, 3) lett a következő állapot, akkor  $T((1, 3), \text{Jobbra}, (2, 3))$  becsült értéke 2/3 lesz.

A passzív ADP-ágens teljes programja a 21.2. ábrán látható. A 21.3. ábra mutatja a  $4 \times 3$ -as világban elérte teljesítményét. Az értékbecsléseinek javulási sebessége szem-

<sup>1</sup> Ez – mint a 20. fejezetben is tárgyalunk – a maximum-likelihood becslés. Egy Bayes-frissítés Dirichlet-priorral jobb is lehet.

pontjából az ADP-ágens a lehetőségekhez képest a legjobban működik, ahogy csak az állapotátmenet-modell tanulási képességének megfelelően haladni tud. Ebben az értelemben más megerősítéses tanulási algoritmusok mérésének standardjaként szolgálhat. Ugyanakkor nagy állapotterek esetén nemiképp kezelhetetlenne válhat. Például az osztáblajátékban nagyjából  $10^{50}$  egyenletet kell megoldania  $10^{50}$  ismeretlenre nézve.

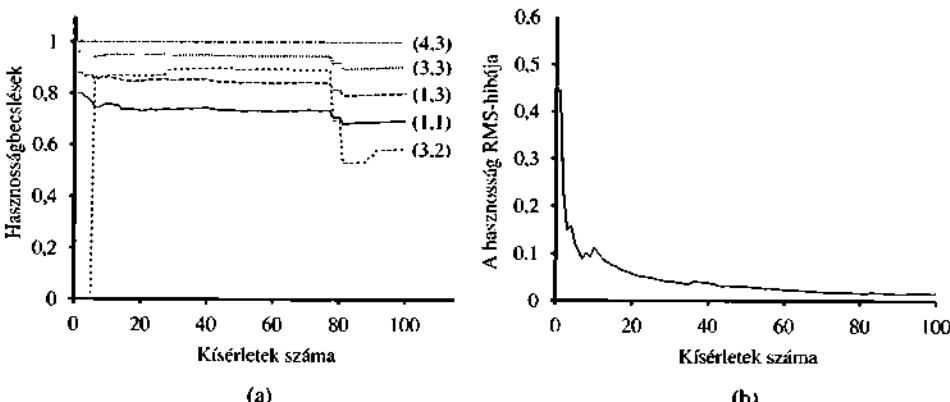
```

function PASSZÍV-ADP-ÁGENS(észlelés) returns egy cselekvés
  inputs: észlelés, egy észlelés, amely a pillanatnyi  $s'$  állapotot és az  $r'$  jutalomjelet adja
  static:  $\pi$ , egy rögzített stratégia
     $mdp$ , egy MDP, amely a  $T$  modellből, az  $R$  jatalomból és a  $\gamma$  leértékelésből áll
     $U$ , egy hasznosságtábla, kezdetben üres
     $N_{sa}$ , az állapot-cselekvés gyakoriságátábla, kezdetben nulla
     $N_{sas'}$ , az állapot-cselekvés-állapot hármasok gyakoriságátábla, kezdetben nulla
     $s, a$ , az előző állapot, illetve cselekvés, kezdetben üres

  if  $s'$  új then do  $U[s'] \leftarrow r'; R[s'] \leftarrow r'$ 
  if  $s$  nem üres then do
    inkrementálja  $N_{sa}[s, a]$ -t és  $N_{sas'}[s, a, s']$ -t
    for each  $t$  amelyre  $N_{sas'}[s, a, t]$  nem nulla do
       $T[s, a, t] \leftarrow N_{sas'}[s, a, t] / N_{sa}[s, a]$ 
     $U \leftarrow$  ÉRTÉK-MEGHATÁROZÁS( $\pi, U, mdp$ )
  if VÉGÁLLAPOT?[ $s'$ ] then  $s, a \leftarrow$  üres else  $s, a \leftarrow s', \pi[s']$ 
  return  $a$ 

```

**21.2. ábra.** Egy adaptív lineáris programozás alapon tanuló passzív megerősítéses ágens. A kód egyszerűsítése érdekében feltétük, hogy minden egyes észlelés szétbontható egy észlelt állapotra és egy jutalomjelre.



**21.3. ábra.** A passzív ADP-tanuló tanulási görbék a  $4 \times 3$ -as világ tanulása során, a 21.1. ábrán látható optimális stratégia használata esetén. (a) Az állapotok egy kiválasztott részhalmazára kiszámított hasznosságbecslések a kísérletek számának függvényében. Figyeljük meg a 78-adik kísérlet környékén a jelentős változásokat – ez az első eset, amikor az ágens a (4,2) végállapotba jut, melyben –1 jutalmat kap. (b) Az  $U(1,1)$  hibájának RMS-értéke 20 – egyenként 100 kísérletből álló – futásra átlagolva.

## Az időbeli különbség tanulása

Lehetőségünk van a két világ (szinte) legjobb vonásait egyesíteni: közelíthetjük az állapotok közti kényszereket leíró egyenleteket anélkül, hogy megoldanánk az összes lehetőséges állapotra. Az alapötlet a következő: használjuk fel a megfigyelt állapotámeneteket a megfigyelt állapotok olyan módosítására, hogy azok megfeleljenek a korlátozó egyenleteknek. Vegyük például az (1,3)-ból (2,3)-ba való átmenetet a 875. oldal második kísérletében. Tegyük fel, hogy az első kísérlet eredményeként a hasznosságuk becsült értéke  $U^\pi(1,3) = 0,84$  és  $U^\pi(2,3) = 0,92$ . Ha ez az átmenet minden előáll, akkor azt várjuk, hogy a hasznosságoknak meg kell felelniük a következő kényszernek:

$$U^\pi(1,3) = -0,04 + U^\pi(2,3)$$

így  $U^\pi(1,3)$  eszerint 0,88 lenne. Tehát a jelenlegi becsült érték (0,84) egy kicsit alacsonyabb a kívántosnál, növelni kellene. Általánosabban megfogalmazva, ha egy  $s$ -ból  $s'$ -be történő átmenet lép fel, akkor a következő módon frissítjük az  $U^\pi(s)$  hasznosságot:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s)) \quad (21.3)$$

Ebben az egyenletben  $\alpha$  az úgynevezett **bátorsági faktor (learning rate)**. Mivel ez az egyenlet a hasznosságok egymást követő állapotokhoz tartozó értékeit használja, ezért gyakran **időbeli különbség vagy IK- (temporal-difference, TD) egyenletnek** nevezik.

Az időbeli különbségen alapuló módszerek alapötlete az, hogy először határozzuk meg azokat a feltételeket, amelyek korrekt becslési értékek esetén lokálisan fennállnak, majd állítsunk fel egy olyan értékfrissítésre szolgáló egyenletet, amely ezen ideális „egyensúlyi” egyenlet irányába viszi a becsléseket. Passzív tanulás esetén az egyensúlyi egyenletet a (21.2) szolgáltatja. A (21.3) egyenlet ténylegesen a (21.2) által megadott egyensúlyi helyzetbe viszi az ágenst, bár van némi finom trükk a dologban. Először vegyük észre, hogy a frissítés csak a megfigyelt következő állapotot,  $s'$ -t veszi figyelembe, míg a tényleges egyensúlyi feltétel az összes lehetséges következő állapotot. Azt hihetnénk, hogy ez túlzottan megváltoztatja  $U^\pi(s)$ -t, amikor egy ritka átmenet következik be. Valójában viszont ritka átmenetek ritkán következnek be, ezért az  $U^\pi(s)$  átlagos értéke a helyes értékhez fog konvergálni. Továbbá, ha  $\alpha$ -t megváltoztatjuk úgy, hogy ne egy rögzített paraméter legyen, hanem egy függvény, amely aszerint csökken, ahogy egy állapotnak a kísérletekben történő előfordulása nő, akkor maga az  $U(s)$  is a helyes értékhez fog tartani.<sup>2</sup> Ez a 21.4. ábrán látható ágensprogramhoz vezet. A 21.5. ábra bemutatja a passzív IK-ágens által a  $4 \times 3$  világban mutatott teljesítményt. Ugyan nem tanul olyan gyorsan, mint az ADP-ágens, és nagyobb változékonyságot mutat, de sokkal kevesebb számítást igényel megfigyeléseként. Vegyük észre, hogy az IK *nem igényel modellt a frissítés elvégzéséhez*. A környezet szolgáltatja a szomszédos állapotok közti kapcsolatokat, a megfigyelt átmenetek formájában.

Az ADP- és IK-megközelítés valójában szoros rokonságban van. Mindkettő lokális változtatásokat végez a hasznosságbecslésben annak érdekében, hogy minden egyes állapot „megfeleljen” a rákövetkezőknek. Ugyanakkor van egy különbség. Az IK úgy módosítja az állapotot, hogy az megfeleljen a megfigyelt rákövetkező állapotnak ((21.3)

<sup>2</sup> Technikailag azt kívánjuk meg, hogy  $\sum_{n=1}^{\infty} \alpha(n) = \infty$  és  $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$  fennálljon. Az  $\alpha(n) = 1/n$  csökkenés megfelel ennek a feltételnek. A 21.5. ábrán  $\alpha(n) = 60/(59+n)$ -t használtuk.

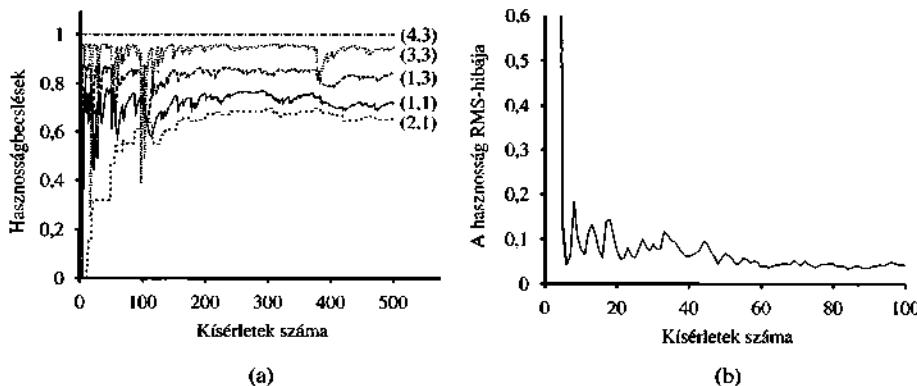
```

function PASSZÍV-IK-ÁGENS(észlelés) returns egy cselekvés
  inputs: észlelés, egy észlelés, amely a pillanatnyi s' állapotot és az r' jutalomjelet adja
  static:  $\pi$ , egy rögzített stratégia
    U, egy hasznosságtábla, kezdetben üres
     $N_s$ , az állapotgyakoriságok tábla, kezdetben nulla
    s, a, r, az előző állapot, cselekvés, illetve jutalom, kezdetben üres

  if s' tj then do U[s']  $\leftarrow r'$ 
  if s nem üres then do
    inkrementálja  $N_s[s]$ -t
     $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$ 
  if VÉGÁLLAPOT?[s'] then s, a, r  $\leftarrow$  nulla else s, a, r  $\leftarrow s', \pi[s'], r'$ 
  return a

```

21.4. ábra. Egy időbeli különbség alapon hasznosságot tanuló passzív megerősítéses ágens



21.5. ábra. Az IK-tanulás  $4 \times 3$ -as világban mért tanulási görbüi. (a) Néhány kiválasztott állapot hasznosságbecslése a kísérletszám függvényében. (b) Az  $U(1,1)$  becslésének RMS-hibája, amelyet 20 futás átlagából számoltunk, minden futás 500 kísérletet tartalmazott. Csak az első 100 kísérletből származó eredményt rajzoltuk fel, hogy a görbe a 21.3. ábrával összehasonlítható legyen.

egyenlet). Az ADP viszont úgy módosítja az aktuális állapotot, hogy az összes lehetséges következő állapotnak megfeleljen, de ezeket az állapotokat a valószínűségükkel súlyozza (21.2) egyenlet). Ez a különbség eltűnik, amikor az IK-módosítást nagyszámú átmenetre átlagoljuk, mivel az állapothalmaz következő állapotainak gyakorisága nagyjából arányos a valószínűségükkel. Egy, az előzőnél fontosabb különbség, hogy míg az IK minden megfigyelt állapotátmenet után egyetlen módosítást végez, addig az ADP annyit, amennyi csak szükséges ahhoz, hogy helyreállítsa az *U* hasznosságbecslések és a *T* környezetmodell közti konziszenciát. Bár a megfigyelt állapotátmenet csak lokálisan változtatja meg *T*-t, de szükség lehet arra, hogy ezt végigterjessük *U*-n. Ennek megfelelően úgy vehetjük, hogy az IK egy durva, de hatékony közelítése az ADP-módszernek.

Mindegyik, az ADP által elvégzett módosítás az IK szempontjából úgy is tekinthető, mint az aktuális környezeti modell szimulációja segítségével generált „pszeudokísérlet”

eredménye. Kiterjeszhetjük az IK-megközelítést úgy, hogy a környezeti modell segítségével számos pszeudokísérletet generáljon – létrehozva olyan átmeneteket, amelyekről az aktuális környezet modellje alapján az IK-ágens el tudja képzelni, hogy *megtörténhemek*. Az IK-ágens akár nagyszámú képzelt átmenetet is generálhat minden megfigyelt átmenetre. Ennek eredményeképpen a hasznosság becslései egyre inkább az ADP becsléseihez közelítenek, természetesen a fizetendő ár a megnövekedett számítási idő lesz.

Hasonló módon az ADP-eljárás hatékonyabb változatait hozhatjuk létre közvetlenül approximálva az értékiteráció és stratégiaiteráció algoritmusait. Ne felejtsük, hogy a teljes értékiteráció kezelhetetlenné válhat, ha az állapottér nagy. Ugyanakkor a módosítási lépések közül számos csak nagyon parányi. Elfogadhatóan jó válaszok gyors generálásához vezető lehetséges megközelítés, ha megkötjük a megfigyelt átmenetek után elvégzett módosítások számát. Azt is megtéhetjük, hogy valamilyen heurisztika segítségével rangsoroljuk a lehetséges módosításokat, és csak a legfontosabbakat végezzük el. A prioritásos végigszöprés (*prioritized sweeping*) heurisztikája preferálja azon állapotok módosítását, amelyek *valószínű* követő állapotainak hasznosságában éppen most *nagy* módosítás történt. Ehhez hasonló heurisztikák segítségével a közelítő ADP-algoritmusok rendszerint nagyjából ugyanolyan gyorsan képesek tanulni, mint a teljes ADP, ha pusztán a tanító szekvenciák számát nézzük. Viszont több nagyságrenddel hatékonnyabbak lehetnek, ha a számítási igényt vizsgáljuk (lásd 21.3. feladat). Ezáltal lehetővé válik számukra olyan állapotterek vizsgálata is, amelyek messze túl nagyok a teljes ADP-nek. A közelítő ADP-algoritmusoknak további előnye, hogy egy új környezet tanulásának korai fázisában a  $T$  környezeti modell gyakran olyan távol van a helyestől, hogy nincs sok értelme egy azzal pontosan konziszens hasznosságfüggvényt kiszámítani. A közelítő algoritmus használhat olyan minimális méretű módosítást, amely csökken annak mentén, ahogy a környezeti modell egyre pontosabbá válik. Ez elkerülhetővé teszi a nagyon hosszú értékiterációkat, amelyek a tanulás korai szakaszában azért lépnek fel, mert nagy változások történnek a modellben.

## 21.3. AKTÍV MEGERŐSÍTÉSES TANULÁS

A passzív tanuló ágensnek rögzített stratégiája van, ez határozza meg a viselkedését. Az aktív ágensnek viszont el kell döntenie, hogy melyik cselekvést válassza. Vegyük először az adaptív dinamikus programozás alapján tanuló ágenst, és nézzük meg azt, hogyan kell módosítani, hogy ezt az új lehetőséget kezelni tudja.

Először is az ágensnek nem csupán egy rögzített stratégia modelljét kell megtanulnia, hanem egy teljes modellt, amibe az összes cselekvésének lehetséges eredményeihez tartozó valószínűségek is beletartoznak. Erre a PASSZÍV-ADP-ÁGENS esetén alkalmazott egyszerű tanulási eljárás is kiválóan alkalmas. Ezek után figyelembe kell vennünk, hogy az ágens választhat a cselekvések között. A megtanulandó hasznosságok azok, amelyeket az *optimális* stratégia definiál. Ezek a hasznosságok megfelelnek a 714. oldalon közzött Bellman-egyenletnek, amit itt megismétlünk:

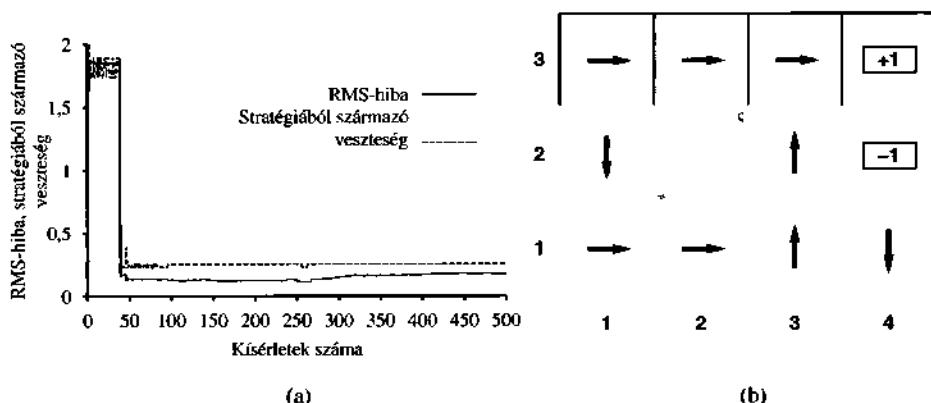
$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (21.4)$$

A 17. fejezetben közölt értékiterációs vagy stratégiaiterációs algoritmusok segítségével ezek az egyenletek megoldhatók az  $U$  hasznosságfüggvényre nézve. A végső kérdés az, hogy mit csinálunk az egyes lépésekben. Az egyik lehetőség, hogy a megtanult modell alapján nyert optimális  $U$  hasznosságfüggvény alapján az ágens meghatároz egy egy lépéstre előretekintő optimális cselekvést annak érdekében, hogy maximálja a várható hasznosságot. A másik lehetőség, hogy ha stratégiaiterációt használt, akkor az optimális stratégia már rendelkezésre áll, így „gondolkodás nélkül” csak végre kell hajtania az optimális stratégia által javasolt cselekvést. A kérdés az, hogy valóban végre kell-e hajtania?

## Felfedezés

A 21.6. ábra bemutatja annak a kísérletsorozatnak az eredményét, amely során az ADP-ágens az egyes lépések során tanult optimális stratégia által javasolt cselekvést követi. Látható, hogy az ágens *nem tanulja meg* sem a helyes hasznosságértékeket, sem az igazi optimális stratégiát! Ehelyett a 39-edik kísérletnél megtalál egy stratégiát, amely a +1 jutalomhoz vezet az alsó – (2,1), (3,1), (3,2), (3,3) – úton (lásd 21.6. ábra). Ezek után kisebb változtatásokkal gyűjti a tapasztalatokat, és a 276-ik kísérlettől kezdve mereven ragaszkodik ehhez a stratégiához, így soha nem tanulja meg a többi állapot hasznosságát, és soha nem talál rá a valóban optimális útra a (1,2), (1,3), (2,3) állapotokon keresztül. Ezt **mohó ágensnek (greedy agent)** nevezzük. Ismételt kísérletek azt mutatták, hogy a mohó ágens ebben a környezetben *nagyon ritkán* találja meg az optimális stratégiát, és időnként borzalmaz stratégikához konvergál.

Hogyan történhet meg, hogy optimális cselekvéseket választva szuboptimális eredményhez jutunk? A válasz abban rejlik, hogy a megtanult modell nem azonos az igazi környezettel, tehát az, ami optimális a megtanult modellben, szuboptimális lehet az igazi környezetben. Sajnálatos módon az ágens nem tudja, hogy milyen a valódi környezet, így nem képes az igazi környezetre nézve optimális cselekvést meghatározni. Mit kell akkor tennünk?



**21.6. ábra.** Annak a mohó ADP-ágensnek a felmutatott teljesítménye, amely végrehajtja a tanult modell alapján optimálisnak tűnő cselekvést. (a) A kilenc nem végállapot alapján számolt átlagos RMS-hiba. (b) Az a szuboptimális stratégia, amely felé ebben a kísérletsorozatban a mohó ágens konvergál.

Amit a mohó ágens nem vett figyelembe, az az, hogy a cselekvésnek több funkciója is van annál, mint hogy a jelenleg megtanult modell alapján jutalmat hozzon, ezenfelül hozzájárul az igazi modell megismeréséhez is a nyert észleléseken keresztül. A modell javítása viszont azt eredményezheti, hogy az ágens több jutalomhoz jut a jövőben.<sup>3</sup> Ennek megfelelően az ágensnek kompromisszumot kell kötnie a **kihasználás (exploitation)** – a jelenlegi hasznosságbecslésében tükrözött modell alapján történő jutalom maximalizálás – és a **felfedezés (exploration)** között, ez utóbbi a hosszú távú eredményességet szolgálja. Ha csupán a kihasználásra tör, akkor azt kockáztatja, hogy beragad valamilyen megszokott kerékvágásba. Ha pedig csak a felfedezéssel törődik, hogy tudását bővítsse, de soha nem ülteti át a tudást a gyakorlatba – nem hoz semmilyen hasznat. A való életben állandóan döntenünk kell, hogy folytassuk-e kényelmes életünket, vagy vágunk neki az ismeretlennek, hátha egy új és jobb életet fedezünk fel. Minél több minden értünk, annál kevesebb felfedezésre van szükségünk.

Tudunk-e ennél kicsit precízebbek lenni? Van egyáltalán *optimális* felfedezési stratégia? Kiderült, hogy ezt a kérdést a statisztikus döntéselmélet egy részterülete az ún. **rabló probléma (bandit problem)** kapcsán nagy mélységen tárgyalta (lásd a bekerezett részt).

Bár a rablók problémáját rendkívül nehéz pontosan megoldani úgy, hogy egy *optimális* felfedezési stratégiát nyerjünk, azonban készíthető egy *ésszerű* terv, ami végül az ágens optimális viselkedéséhez vezet. minden ilyen eljárás a végtelen felfedezés határán lehet mohó (**VFHM**) (*greedy in the limit of infinite exploration, GLIE*). Egy VFHM-séma alapján működő ágensnek korlátlan számban ki kell próbálnia minden cselekvést az összes állapotban. Ezáltal kerüli el annak veszélyét, hogy egy szokatlanul rossz kimenetű sorozat miatt véges (nem nulla) valószínűséggel nem talál meg egy optimális cselekvést. Egy ilyen terv alapján működő ADP-ágens végül megtanulja az igazi környezeti modellt. Egy VFHM-tervnek végül mohóvá kell válnia, így az ágens cselekvései optimálissá válnak a megtanult (azaz itt az igazi) modellre nézve.

## Rablók és felfedezés

Las Vegasban a *félkarú rabló* egy pénzbedobós jáékautomata. A játékos bedoh egy émnét, meghúzza a kart, és elveszi a nyereményt (ha nyer). Egy  $n$  karú rablónak (*n-armed bandit*)  $n$  karja van. A játékosnak minden érme bedobásakor választanak kell, hogy melyik kart húzza meg – azt, amelyik eddig a legjobban fizetett, vagy talán egy olyat, amelyet még nem próbált?

Az  $n$  karú rabló probléma a világ sok, létfontosságú területén számos valós feladat formális modelljeként szolgálhat, mint például az MI kutatás-fejlesztés éves költségvetésének kialakítása. Mindegyik kar egy cselekvésnek felel meg (mint például 400 millió forint elkülönítése MI-tankönyvek fejlesztésére), és a kar meghúzássával járó nyeremény a cselekvés választása esetén várható hasznának felel meg (esetünkben óriási haszon várható). Az ismeretlen terület feltárása – akár egy új kutatásról, akár egy új üzletközpontról van szó – kockázatos, drága, bizonytalan haszonnal jár; másrészt ha egyáltalán nem kutatunk, akkor soha nem fogunk *semmilyen* értékes cselekvést felfedezni.

Ahhoz, hogy a rabló problémát megfelelően formalizáljuk, pontosan meg kell határoznunk, hogy mit értünk optimális viselkedésen. Az irodalomban található legtöbb definíció azt feltételezi, hogy a cél az ágens élettartama alatt elérhető összes jutalom várható értékének maximalizálása. Ezek a definíciók megkívánják, hogy várható értéket képezzük egyrészt az összes lehetséges világ felett,

<sup>3</sup> Vegyük észre a 16. fejezetben tárgyalta információérték-elmélettel fennálló közvetlen analógiát.

amelybe csak kerülhet az ágens, másrészről bármelyik adott világ minden egyes lehetséges cselekvései szekvenciája fölött. Itt a „világot” a  $T(s, a, s')$  állapotátmenet-modell definíálja. Tehát ahhoz, hogy az ágens optimálisan cselekedjék, szüksége van a lehetséges modellek a priori eloszlására. Ezen az úton rendszerint teljesen kezelhetetlen optimalizációs problémákhoz jutunk.

Néhány esetben azonban – például amikor az egyes gépek nyereményei függetlenek, és leértékeltekt a jutalmakat használunk – minden egyes játékgépre kiszámítható az úgynevezett **Gittins-index** (Gittins, 1989). Az index csak annak függvénye, hogy hányszor választottuk a gépet, és eddig mennyit fizetett ki. Az index azt jelzi minden egyes gépre, hogy mennyire éri meg további pénzt bedobnunk, és a várható kifizetésen és az információ várható értékén alapul. A legnagyobb indexű gép választása az optimális felfedezési stratégia. Sajnálatos módon eddig még nem sikerült megtalálni annak a módot, hogy hogyan kell a Gittins-indexet szekvenciális döntési problémákra kiterjeszteni.

Az  $n$  karú rabló elmélet felhasználható arra, hogy bizonyítsuk a genetikus algoritmusok szelekciós stratégiájának ésszerűségét (lásd 4. fejezet). Tekintsünk az  $n$  karú rabló problémában minden egyes kart egy lehetséges génfüzérnek, továbbá a pénz bedobása ennél a karnál feleljen meg ezen gének reprodukciójának. Ebben az esetben a genetikus algoritmus optimálisan osztja szét a pénzt akkor, ha adott a függetlenségi feltételezések egy megfelelő halmaza.

Számos VFHM-séma ismert; az egyik legegyszerűbb, ha az ágens az idő  $1/t$  részében véletlen cselekvést választ, egyébként pedig mohó stratégiát követ. Tény, hogy ez végül optimális stratégiához vezet, de időnként rendkívül lassan konvergál. Egy értelmesebb megközelítés, ha az ágens némi súlyt ad azoknak a cselekvéseknek, amelyeket nem használt még gyakran, miközben igyekszik elkerülni azokat, amelyeknek a hasznosságát kicsinek hiszi. Ezt meg tudjuk valósítani úgy, ha úgy módosítjuk a (21.4) kényszeregenletet, hogy nagyobb hasznosságot tulajdonítson a relatíve kipróbatlan állapot-cselekvés pároknak. Lényegében egy optimista priorit hoz létre, az ágens kezdetben úgy viselkedik, mintha hatalmas jutalmak lennének szétszórva a helyszínen. Jelölje  $U^+(s)$  az  $s$  állapot hasznosságának (tehát a hátralevő jutalom várható értékének) optimista becslését,  $N(a, s)$  pedig azt a számot, ahányszor az  $s$  állapotban az  $a$  cselekvést választottuk. Tegyük fel, hogy az ADP-tanulási végző ágens értékiterációt végez, ez esetben a frissítési egyenletet [(17.6) egyenlet] át kell írnunk úgy, hogy az az optimista becslést tartalmazza. A következő összefüggés ezt megvalósítja:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f \left( \sum_{s'} T(s, a, s') U^+(s'), N(a, s) \right) \quad (21.5)$$

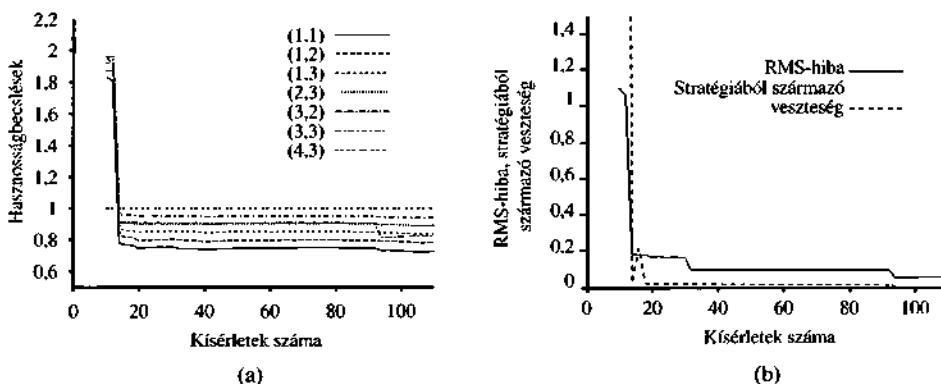
Itt  $f(u, n)$  az úgynevezett **felfedezési függvény** (exploration function). Ez határozza meg a mohóság (a magas  $u$  értékek preferenciája) és a kíváncsiság (az alacsony  $n$  értékek preferenciája – cselekvések, amelyeket még nem alkalmaztunk nagyon gyakran) közötti kompromisszumot. Az  $f(u, n)$  függvénynek  $u$ -ban növekvőnek,  $n$ -ben csökkenőnek kell lennie. Nyilvánvalóan sok lehetséges függvény megfelel ennek a feltételnek. Egy különösen egyszerű függvénydefiníció:

$$f(u, n) = \begin{cases} R^+ & \text{ha } n < N_e \\ u & \text{egyébként} \end{cases}$$

ahol  $R^+$  a tetszőleges állapotban kapható legnagyobb jutalom optimista becslése, míg  $N_e$  egy rögzített paraméter. Ennek alkalmazásával az ágens minden állapot-cselekvés párt legalább  $N_e$  alkalommal kipróbal.

Az, hogy a (21.5) egyenlet jobb oldalán nem  $U$ , hanem  $U^+$  szerepel, nagyon fontos. Ahogy a felfedezés folyik, a startállapot környéki állapotok és cselekvések jó néhány-

szor kipróbálásra kerülhetnek. Ha  $U$ -t – a passzimistább becslést – használtuk volna, akkor az ágens hamar elveszthetné a szabad területek felfedezése iránti vonzalmát. Az  $U^+$  szerepettelése azt jelenti, hogy a felfedezéshez kapcsolt jutalom visszaterjed a felfedezetlen terület határáról. Így nem csupán a szokatlan cselekvéseket preferáljuk, hanem a felértekkelük azokat a cselekvéset, amelyek a felfedezetlen területek felé fogják vinni az ágenst. Ennek a felfedezési stratégiának a hatása jól látható a 21.7. ábrán, gyors konvergenciát látunk az optimális teljesítményhez, ellentétben a mohó ágensnél látottakkal. Az optimális hoz nagyon közeli stratégiát talál mindenkor 18 kísérlet után. Végük észre, hogy maguk a hasznosságérték-becslések nem konvergálnak ilyen gyorsan. Ennek az az oka, hogy az ágens meglehetősen hamar felhagy az állapottér jutalmat nem eredményező részeinek felfedezésével, oda már csak „véletlenszerűen” téved. Mindamellett teljesen ésszerű az ágens részéről, hogy nem sokat törödik azon állapotok hasznosságának pontos értékével, amelyekről tudja, hogy nemkívánatosak és elkerülhetők.



21.7. ábra. A felfedező ADP-ágens teljesítménye,  $R = 2$  és  $N_e = 5$  esetén. (a) Néhány kiválasztott állapot hasznosságának becslése az idő függvényében. (b) A hasznosságértékek RMS-hibája, és az ehhez kapcsolható, a stratégiából származó veszteség.

## Egy cselekvésérték-függvény tanulása

Most, hogy már van egy aktív ADP-ágensünk, nézzük meg, hogy mi módon tudunk egy aktív időbeli különbség tanuló ágenst létrehozni. A legszembetűnőbb eltérés a passzív esethez képest, hogy az ágensnek többé nem áll rendelkezésére egy rögzített stratégia, tehát ha megtanul valamelyen  $U$  hasznosságfüggvényt, akkor meg kell tanulnia egy modellt is, hogy egy  $U$ -n alapuló egylépéses előretekintő keretben cselekvést tudjon választani. A modellkialakítás problémája ugyanaz az IK-ágensnél, mint az ADP-ágensnél. Mi a helyzet magával az IK frissítési szabállyal? Talán meglepő, de a (21.3) IK frissítési szabály változatlan marad. Ez furcsának tűnhet a következő okból: tegyük fel, hogy az ágens olyan lépést választ, ami normál esetben jó célhoz vezet, de a környezet nemdeterminisztikus volta miatt az ágens valamelyen katasztrófális állapotban végzi. Az IK frissítési szabály ezt éppoly komolyan veszi, mintha ez a kimenetel a cselekvés normál következménye lenne, pedig csak egy pech volt, és nem kellene sokat aggódnia.

miatt. Valójában az ilyen valószínűtlen kimenetel csak ritkán fordul elő egy nagy tanítósorozat-halmazban, így reményeink szerint hosszú távon a valószínűségével arányosan kap súlyt a hatása. Ismét azt mondhatjuk, hogy az IK-ágens ugyanazokhoz az értékekhez fog tartani, mint az ADP-ágens, ahogy a tanítósorozatok száma tart a vég-telenhez.

Van egy alternatív IK-módszer, amit ***Q*-tanulásnak (*Q*-learning)** hívunk, ez cselekvésérték-reprezentációt tanul, nem hasznosságot.  $Q(a, s)$ -sel fogjuk jelölni annak az értékét, ha az  $s$  állapotban az  $a$  cselekvést választjuk. A  $Q$ -értékek közvetlenül összekapcsolhatók a hasznosságértékekkel a következő összefüggés alapján:

$$U(s) = \max_a Q(a, s) \quad (21.6)$$

A  $Q$ -függvény úgy tűnhet, mintha csupán a hasznosságinformáció tárolásának egy elérő módja lenne, de van egy nagyon fontos tulajdonsága: *egy Q-függvényt tanuló IK-ágensnek nincs szüksége sem a tanulás, sem a cselekvés kiválasztás modelljére*. Ezért a *Q*-tanulást **modellmentes (model-free)** módszernek is nevezik. Éppúgy, mint a hasznosságoknál, felírhatunk egy egyensúlyi egyenletet, amelyet a korrekt  $Q$ -értékeknek kell elégíteniük:

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s') \quad (21.7)$$

Az ADP-tanulást végező ágenssel azonos módon itt is felhasználhatjuk ezt az egyenletet egy iterációs eljárásban arra, hogy egy becsült modell alapján pontos  $Q$ -értékeket számoljunk. Ez viszont azt kívánna meg, hogy a modellt is megtanuljuk, hiszen az egyenlet használja a  $T(s, a, s')$ -t. Másrészt viszont az IK-megközelítés nem igényel modellt. Így a  $Q$ -értékekre vonatkozó IK frissítési összefüggés:

$$Q(a, s) \leftarrow Q(a, s) + \alpha(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s)) \quad (21.8)$$

amely minden kiszámításra kerül, amikor az  $s$  állapotban az  $a$  cselekvést választottuk, amely az  $s'$  állapotba vitt.

A 21.8. ábra mutatja be az IK-módszert használó, felfedező *Q*-tanuló ágens teljes programját. Vegyük észre, hogy ugyanazt az  $f$  felfedezési függvényt használja, mint amit a felfedező ADP-ágens is használt – ezért van szükség a végrehajtott cselekvések gyakorisági statisztikáira (az  $N$  táblára). Ha egy egyszerűbb felfedezési stratégiát használnánk – mondjuk a lépések egy részében véletlenszerű működést iktatva be, ahol ez részarány az idővel csökken –, akkor mellőzhetnénk ezt a statisztikát.

A *Q*-tanuló ágens megtanulja a  $4 \times 3$ -as világra az optimális stratégiát, de ezt sokkal lassabban teszi, mint az ADP-ágens. Ennek oka, hogy az IK nem kényszerít modellje alapján konziszenciát az értékekre. Ez az összehasonlítás felvet egy általános kérdést: mi a jobb: egy modellt és egy hasznosságfüggvényt tanulni, vagy egy cselekvésértékfüggvényt? Más szavakkal, mi a legjobb módja egy ágensfüggvény reprezentálásának? Ez a mesterséges intelligencia alapjait érintő kérdés. Mint az 1. fejezetben kijelentettük, az MI legnagyobb részében kulcsfontosságú történeti örökség a (gyakran ki nem mondott) ragaszkodás a **tudásalapú (knowledge-based)** megközelítéshez. Ez vezet ahhoz a feltevéshez, hogy az ágensfüggvény reprezentációjának legjobb módja, ha valamilyen szempontból az ágenst körülvevő környezet modelljét építjük meg.

Néhány kutató ugyanakkor – mind az MI-közösségen belül, mind azon kívül – azt állítja, hogy a modellmentes módszerek, mint például a *Q*-tanulás léte azt mutatja, hogy

```

function Q-TANULÓ-ÁGENS(észlelés) returns egy cselekvés
  inputs: észlelés, egy észlelés, amely a pillanatnyi s' állapotot és az r' jutalomjelet adja
  static: Q, egy cselekvésérték-tábla; amelyet az állapottal és a cselekvéssel indexelünk
    Nsa, az állapot-cselekvés párok gyakoriságának táblája
    s, a, r, az előző állapot, cselekvés, illetve jutalom, kezdetben nulla

  if s nem nulla then do
    inkrementálja Nsa[s, a]-t
    Q[a, s] ← Q[a, s] + α(Nsa[s, a])(r + γ maxa' Q[a', s'] - Q[a, s])
  if VÉOÁLLAPOT?[s'] then s, a, r ← nulla
  else s, a, r ← s', argmaxa' f(Q[a', s']), Nsa[a', s'], r'
  return a

```

**21.8. ábra.** Egy felfedező *Q*-tanuló ágens. Egy aktív tanuló, amely megtanulja minden cselekvésnek minden egyes állapotban a  $Q(a, s)$  értékét. Ugyanazt az *f* felfedezési függvényt használja, amelyet a felfedező ADP-ágens is használt, de elkerüli az állapotámenet-modell tanulását, mivel az állapot *Q*-értékét közvetlenül a szomszédos állapotok *Q*-értékeihez tudja kapcsolni.

a tudásalapú megközelítés nem szükséges. Mindamellett nem sok egyéb van e kérdés eldöntésére, mint az intuícióink. A mi intuícióink szerint, ahogy a környezet komplexebbé válik, a tudásalapú megközelítés előnyei egyre nyilvánvalóbbá válnak. Ez még a játékokban – mint a sakk, a dámajáték és az ostáblajáték (lásd következő alfejezet) – is így jelentkezik, a modellalapú kiértékelő függvények tanulására fordított erőfeszítések több sikert hoztak, mint a *Q*-tanuló módszerek.

## 21.4. A MEGERŐSÍTÉSES TANULÁS ÁLTALÁNOSÍTÓ-KÉPESSÉGE

Az eddigiekben azt feltételeztük, hogy az ágens által megtanult hasznosságfüggvények és a *Q*-függvények táblázatos formában reprezentáltak, amelyben minden egyes bemeneti vektornak egy-egy kimeneti érték felel meg. Ez a megközelítés kis állapot-terekre jól működik, de a konvergencia ideje, valamint egy-egy iterációs lépés ideje (ADP esetén) gyorsan nő, ahogy a tér egyre nagyobb lesz. Alaposan kézben tartott közelítő ADP-módszerekkel kezelhető lehet 10 000 állapot vagy annál valamivel több. Ez kielégítő a kétdimenziós útvesztő jellegű problémák esetén, de reálisabb problémák szóba sem jöhettek. A sakk és az ostáblajáték a valós világ apró részei csupán, mégis állapotterük  $10^{50} - 10^{120}$  állapotot tartalmaz. Az teljesen abszurd feltételezés, hogy mindezeket az állapotokat be kell járnunk ahhoz, hogy megtanuljuk a játékot!

Ezeknek a problémáknak egy lehetséges kezelési módja, ha **függvényapproximáció** (**function approximation**) használunk, ami egyszerűen azt jelenti, hogy a leképezéseknek bármilyen reprezentációját használhatjuk, kivéve a táblázatos reprezentációt. A reprezentációt közelítőnek tekintjük, mivel nem áll fenn biztosan, hogy az *igazi* hasznosságfüggvény vagy *Q*-függvény reprezentálható a választott formában. A 6. fejezetben például leírtunk egy sakkjátékban használható **kiértékelő függvényt** (**evaluation**

**function), amelyet tulajdonságok (feature) (vagy bázisfüggvények (basis function))  $f_1, \dots, f_n$  halmazának lineáris kombinációjával reprezentáltunk:**

$$\hat{U}_\theta(s) = \theta_0 f_1(s) + \theta_1 f_2(s) + \dots + \theta_n f_n(s)$$

Megerősítéses tanulás során tanulhatjuk a  $\theta = \theta_0, \dots, \theta_n$  paramétereket úgy, hogy az az igazi hasznosságfüggvényt közelítse. Ahelyett, hogy mondjuk  $10^{120}$  értéket tartanánk egy táblázatban, ez a függvényközelítés mondjuk  $n = 20$  értékkel jellemzhető, ez elköpesztő tömörítés! Bár senki sem ismeri a sakkjáték igazi hasznosságfüggvényét, azt senki se hiszi, hogy pontosan jellemzhető lenne 20 számmal. Mindamellett, ha az approximáció kielégítően jó, az ágens még mindig képes lehet kiválasztani.<sup>4</sup>

A függvényapproximáció tulajdonságai lehetővé teszik, hogy praktikusan felhasználható legyen nagy állapotterek hasznosságfüggvényeinek reprezentálására, de nem ez az alapvető előnyük. A függvényapproximátorokkal elérte tömörítés lehetővé teszi a tanuló ágens számára az általánosítást azokról az állapotokról amelyeket bejárt, azokra az állapotokra, amelyeket nem járt be. Azaz a függvényapproximáció legfontosabb aspektusa nem az, hogy kevesebb helyet igényel, hanem az, hogy lehetővé teszi az induktív általánosítást a bemeneti állapotterén. Hogy némi fogalmat alkothassunk ennek a hatásnak az erejéről: a lehetséges ostáblajáték állapotok közül csupán minden  $10^{44}$ -edik állapotnak a vizsgálata alapján megtanulható olyan hasznosságfüggvény, amely képesse lehet egy programot, hogy olyan jól játszon, mint egy ember (Tesauro, 1992).

Az éremnek természetesen van másik oldala is: lehet, hogy a kiválasztott hipotézis-terben egyáltalán nincs olyan függvény, amely kielégítően approximálná az igazi hasznosságfüggvényt. Mint az induktív tanulásnál minden esetben, itt is kompromisszum van a hipotézistér mérete és a függvény megtanulásához szükséges idő között. Egy nagyobb hipotézistér valószínűbbé teszi, hogy jó approximációt lehet találni, de azt is jelenti, hogy a konvergencia valószínűleg lassabb lesz.

Kezdjük a legegyszerűbb esettel, a közvetlen hasznosságbecsléssel (lásd 21.2. alfejezet). Függvényapproximáció esetén ez nem más, mint a **felügyelt tanulás (supervised learning)** egy esete. Tegyük fel például, hogy a  $4 \times 3$  világ hasznosságértekeit egy egyszerű lineáris függvényvel reprezentáljuk. A négyzetek jellemzői egyszerűen az  $x$  és  $y$  koordinátáik, tehát:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y \quad (21.9)$$

Tehát ha  $(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1)$ , akkor  $\hat{U}_\theta(1, 1) = 0.8$ . Adott kísérlethalmaz esetén egy sor értéket nyerünk az  $\hat{U}_\theta(x, y)$  értékekre, és megtalálhatjuk a – mondjuk négyzetes értelemben – legjobb illesztést, standard lineáris regressziót használva (lásd 20. fejezet).

Megerősítéses tanulás esetén jobbnak tűnik *online* tanulást alkalmazni, amely minden egyes kísérlet után frissíti a paramétereket. Tegyük fel, hogy lefuttatunk egy kísérletet, és a teljes jutalom  $(1, 1)$ -ből indulva 0.4. Ez arra utal, hogy az  $\hat{U}_\theta(1, 1) = 0.8$  jelenlegi

<sup>4</sup> Tudjuk, hogy az igazi hasznosságfüggvény reprezentálható egy- vagy kétoleányi Lisp, Java vagy C++ kód-dal. Azaz reprezentálható egy programmal, ami pontosan oldja meg a játékot, valahányszor meghívjuk. Mi csak olyan függvényapproximátorokban vagyunk érdekeltek, amelyek **ésszerű** számítási mennyiséget végeznek. Valójában jobb lehet, ha egy nagyon egyszerű függvényapproximátor tanulunk meg, és kombináljuk valamilyen lépésszámú előretekintő kereséssel. A szükséges kompromisszumok mindenkorra egyelőre nem jól ismertek.

érték túl nagy, és csökkenteni kell. Hogyan kell a paramétereket módosítani, hogy ezt elérjük? Éppúgy, mint a neurális háló tanulásnál, egy hibafüggvényt írunk fel, és felírjuk a paraméterekre vonatkozó gradiensét. Ha  $u_j(s)$  a  $j$ -edik kísérlet során az  $s$  állapotból kiindulva gyűjtött teljes megfigyelt jatalom, akkor a hibát a jóslott és a tényleges teljes jatalom négyzetének különbségeként (tulajdonképpen annak feleként) definiálják:  $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$ . Az egyes  $\theta_i$  paraméterek változásának hatására a hiba változásának sebességét a  $\partial E_j / \partial \theta_i$  adja meg. Így, ha a paramétert abban az irányba akarjuk változtatni, hogy a hibát csökkentse, akkor azt akarjuk, hogy:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \quad (21.10)$$

Ezt Widrow–Hoff-szabálynak (**Widrow-Hoff rule**) vagy delta-szabálynak (**delta rule**) nevezik, és az online legkisebb négyzetes hibára vonatkozik. A (21.9) egyenletben felírt lineáris függvényapproximátorra a következő három egyszerű frissítési szabályt kapjuk:

$$\begin{aligned}\theta_0 &\leftarrow \theta_0 + \alpha (u_j(s) - \hat{U}_\theta(s)) \\ \theta_1 &\leftarrow \theta_1 + \alpha (u_j(s) - \hat{U}_\theta(s))x \\ \theta_2 &\leftarrow \theta_2 + \alpha (u_j(s) - \hat{U}_\theta(s))y\end{aligned}$$

Alkalmazhatjuk ezeket a szabályokat arra az esetre, amikor  $\hat{U}_\theta(1,1) = 0,8$  és  $u_j(1,1) = 0,4$ . Ekkor  $\theta_0$ ,  $\theta_1$  és  $\theta_2$  mindegyike  $0,4\alpha$ -val csökken, ami minden csökkenti az  $(1,1)$ -beli hibát. Vegyük észre, hogy a  $\theta_i$ -k változtatása az összes többi állapotra is megváltoztatja  $\hat{U}_\theta$  értékét! Ezt értettük azon, amit mondunk: a függvényapproximáció lehetővé teszi a megerősítéses tanulást végzőnek, hogy általánosítson a tapasztalataiból.

Azt várjuk, hogy az ágens gyorsabban fog tanulni, ha függvényapproximátort használ, feltéve, hogy a hipotézistér nem túl nagy, de van benne néhány olyan függvény, amely kielégítően illeszkedik az igazi hasznosságfüggvényre. A 21.7. feladatban azt várjuk az olvasótól, hogy értékelje a közvetlen hasznosságbecslést mind a függvényapproximáció felhasználásával, mind anélkül. A  $4 \times 3$ -as világban a javulás érzékelhető, de nem drámai, mivel nagyon kicsi az állappátról, amivel dolgozni kell. A javulás sokkal nagyobb a  $10 \times 10$ -es világban, ha +1 jatalom van a  $(10,10)$  mezőn. Ez a világ jól illeszthető a lineáris hasznosságfüggvényhez, mivel az igazi hasznosságfüggvény sima és közel lineáris (lásd 21.10. feladat). Ha a jatalmat az  $(5,5)$  mezőre tesszük, akkor az igazi hasznosságfüggvény piramis jellegű, és a (21.9) egyenletben felvett függvényapproximátor szánálmas kudarcot vall. Mindamellett nem veszett el minden! Emlékezzünk vissza, hogy a lineáris függvényapproximációjánál az számít, hogy a függvény paramétereiben legyen lineáris – a használt tulajdonságok önmagukban az állapotváltozók tetszőleges nemlineáris függvényei lehetnek. Tehát használhatunk egy  $\theta_3 = \sqrt{(x-x_g)^2 + (y-y_g)^2}$  típusú tagot, amely a céltól való távolságot méri.

Ugyanilyen jól alkalmazhatjuk ezeket az elveket az időbeli különbség alapján tanulókra is. Mindössze annyit kell tennünk, hogy a paramétereket úgy módosítjuk, hogy az egymásra következő állapotok időbeli különbségét csökkenteni próbáljuk. A (21.3) és (21.8) IK- és  $Q$ -tanulási egyenleteink új változata:

$$\theta_i \leftarrow \theta_i + \alpha [R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \quad (21.11)$$

a hasznosságfüggvényre, és

$$\theta_i \leftarrow \theta_i + \alpha \left[ R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s) \right] \frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i} \quad (21.12)$$

a  $Q$ -értékekre. Megmutatható, hogy ezek a frissítési szabályok a paramétereiben lineáris függvényapproximátor esetén az igazi függvény lehető legközelebbi<sup>5</sup> approximációjához konvergálnak. Szerencsétlen módon mindez nem áll meg, ha nemlineáris függvényeket használunk – mint amilyenek például a neurális hálók. Nagyon egyszerű példák találhatók, amelyeknél a paraméterek a végzetlenhez tartanak, pedig jó megoldások találhatók a hipotézistérben. Vannak kifinomultabb algoritmusok, amelyek elkerülik ezeket a problémákat, de napjainkban a megerősítéses tanulás általános függvényeket használó függvényapproximátorral – megmaradt a magas művészettel szintjén.

A függvényapproximáció a környezet modelljének tanulásában is nagyon hasznos lehet. Emlékezzünk rá, hogy egy *megfigyelhető* környezet modelljének tanulása *ellenőrzött* tanulási feladat, mivel a következő észlelés megadja a kívánt kimeneti állapotot. A 18. fejezetben ismertetett bármely ellenőrzött tanulási módszer használható, megfelelő módosításokkal. Ugyanis itt nem egy logikai változó vagy egyetlen valós érték predikciójáról van szó, hanem komplett állapotleírásokról. Ha például az állapot  $n$  logikai változóval adható meg, akkor  $n$  logikai függvényt kell megtanulnunk, hogy az összes változót megjósolhassuk. Sokkal nehezebb a tanulási probléma *részlegesen megfigyelhető* környezet esetén. Ha tudjuk, hogy melyek a rejtegett változók, és tudjuk, hogy milyen oksági viszonyban állnak egymással és a megfigyelhető változókkal, akkor a 20. fejezetben leírtak szerint rögzíthetünk egy megfelelő Bayes-hálóstruktúrát, és használhatjuk az EM-algoritmust a paraméterek tanulására. A rejtegett változók bevezetése, illetve a modellstruktúra tanulása jelenleg megoldatlan problémák.

Nézzük most a megerősítéses tanulás komolyabb alkalmazásait. Látni fogjuk, hogy azokban az esetekben, amelyekben a hasznosságfüggvényt (és ezzel együtt a modellt) használjuk, ott feltételezzük, hogy a modell adott. Például az osttáblajáték kiértékelő függvényének tanulása során általában feltesszük, hogy a legális lépések és azok hatása előzetesen ismert.

## Alkalmazások a játékok területén

A megerősítéses tanulás első fontos alkalmazása Arthur Samuel dámjáték programja volt (Samuel, 1959; 1967) – egyben ez volt a tanuló programok összes válfaja között is az első jelentősebb. Samuel először egy legfeljebb 16 tagú súlyozott lineáris állandókiértékelő függvényt használt. A (21.11) egyenlet egy variánsát alkalmazta frissítési szabályként. Mindamellett volt néhány lényeges eltérés az Ő programja és a jelenleg használt módszerek között. Az első az, hogy Ő a jelen állapot és a keresési fában végre hajtott teljes előretekintéses keresés által adott érték közti különbség alapján frissítette a súlyokat. Ez nagyon jól működik, mert ahhoz vezet, hogy az állapotteret különböző felbontással lássuk. A második különbség, hogy a programja *nem* használt semmilyen megfigyelt

<sup>5</sup> A hasznosságfüggvények közti távolság definíciója csak technikai kérdés. Lásd (Tsitsiklis és Van Roy, 1997).

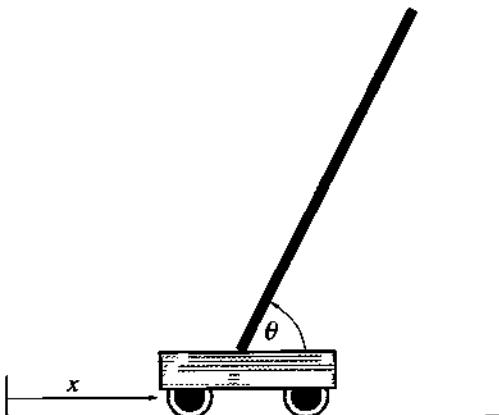
jutalmat! Tehát a végállapot értékét figyelmen kívül hagyta. Ez azt jelenti, hogy könyen elköpzelhető, hogy Samuel programja nem konvergál, vagy ha konvergál, akkor a nyerő helyett a vesztő stratégiához tart. Samuel ezt oly módon kerülte el, hogy ragaszkodott ahhoz, hogy az anyagi előnyhöz tartozó súly pozitív legyen. Érdekes módon ez elegendő volt ahhoz, hogy programját a súlyok terének olyan részébe irányítsa, amely jó dámajátékot eredményezett.

Gerry Tesauro TD-Gammon rendszere (Tesauro, 1992) jól mutatja a megerősítéses tanulási technikák erejét. Korábbi munkájában (Tesauro és Sejnowski, 1989) Tesauro megkísérítette a  $Q(a, s)$  függvény neurális reprezentációjának tanulását közvetlenül humán szakemberek által relatív értékekkel címkezett lépések alapján. Ez a megközelítés nagyon fárasztó volt a szakembereknek. Az eredményül kapott NEUROGAMMON nevű program a számítógépes programok között erős játékos volt ugyan, de nem volt versenyképes a humán szakemberekkel. A TD-Gammon projekt az önmagában való tanulás egy kísérlete volt. A TD-Gammon csupán az egyes játszmák végén adott egyetlen jutalomjelet. A kiértékelő függvény reprezentációját egy teljesen összekötött, egyetlen – 40 elemből álló – rejttet réteggel felépített neuronhálóval oldotta meg. A (21.11) egyenlet szimpla ismételt alkalmazásával a TD-Gammon lényegesen jobban megtanult játszani, mint a NEUROGAMMON annak ellenére, hogy a bemeneti reprezentáció a számított tulajdonságok nélküli nyers táblaállás volt. Ez körülbelül 200 000 játékot és két hétközött számítógépes futást igényelt. Bár ez hatalmas mennyiséggű játéknak tűnhet, de valójában az állapottérnek csak egy elhanyagolhatóan kis részét fedи le. Amikor a bemeneti reprezentációt számított tulajdonságokkal bővítették, akkor egy 80 rejttet rétegbeli elemmel felépített neurális háló 300 000 tanító játszma után képes volt olyan szinten játszani, mint a világ három legerősebb játécosa. Kit Woolsey, az egyik élvonalbeli játékos és szakíró azt nyilatkozta: „Kétségtelen, hogy a pozíciómegítélezésben sokkal jobb nálam.”

## Robotirányítási alkalmazások

A híres **rúd-egyensúlyozási (cart-pole)** feladat, amelyet **inverz ingának (inverted pendulum)** is neveznek, a 21.9. ábrán látható. A feladat a kocsi  $x$  pozíciójának vezérlése úgy, hogy a rúd nagyjából fölfelé álljon ( $\theta \approx \pi/2$ ), miközben a kocsi a pálya kijelölt határai között marad. A megerősítéses tanulás és a szabályozáselmélet területén több mint 2000 cikket publikáltak erről a látszólag egyszerű feladatról. A rúd-egyensúlyozási probléma annyiban különbözik az eddigiekben tárgyaltaktól, hogy az állapot-változói  $x$ ,  $\theta$ ,  $\dot{x}$ ,  $\dot{\theta}$  folytonosak. A cselekvés általában diszkrét: lökés jobbra vagy lökés balra, az úgynevezett **bang-bang szabályozás (bang-bang control)**.

Az első – ezen probléma tanulását célzó – munkát Michie és Chambers végezte (Michie és Chambers, 1968). Az Ő BOXES nevű algoritmusuk mindenkorban 30 próbálkozás után képes volt a rudat több mint egy óráig egyensúlyban tartani. Ráadásul, sok későbbi rendszertől eltérően, a BOXES egy valódi kocsit és rudat használt, nem szimulációt. Az algoritmus először is kvantálta, azaz tartományokra („dobozokra”) bontotta a négydimenziós állapottérét, innen jött az algoritmus neve is. Ezek után addig végeztek egy-egy kísérletet, amíg a rúd leesett vagy a kocsi elérte a pálya végét. Negatív megerősítést rendeltek az utolsó állapottér-tartományban az utolsó cselekvéshez, és ezt terjesztették vissza a megelőző szekvenciára. Azt találták, hogy a kvantálás akkor okoz bizonyos



**21.9. ábra.** A mozgó kocsi tetején álló hosszú rúd egyensúlyozásának vizsgálatára szolgáló berendezés. A kocsit az  $x$ ,  $\theta$ ,  $\dot{x}$ ,  $\dot{\theta}$  állapotváltozókat megfigyelő szabályzó jobbra vagy balra lökheti.

problémákat, ha a berendezést más pozícióból indították, mint amiket a tanítás során használtak, ami arra utal, hogy az általánosítás nem volt tökéletes. Jobb általánosítóképesség és gyorsabb tanulás érhető el, ha egy olyan algoritmust használunk, amely *adaptívan osztja* részekre az állapotteret, a jutalom megfigyelt változásainak megfelelően. Manapság egy *háromszoros* invertált inga egyensúlyozása közönséges feladatnak számít – ez már a legtöbb ember ügyességét messze meghaladja.

## 21.5. STRATÉGIAKERESÉS

A megerősítéses tanulással kapcsolatos, utolsó általunk tárgyalt megközelítés **stratégakeresés** (*policy search*) néven ismert. Bizonyos értelemben a stratégakeresés a legegyszerűbb a fejezet összes módszere között. Az alapötlet annyi, hogy engedjük változni a stratégiát, amíg javul, és utána állítsuk le a változást.

A vizsgálatot kezdjük magával a stratégiával. Emlékezzünk vissza, hogy a  $\pi$  stratézia egy olyan függvény, amely állapotokat képez le cselekvésekre. Mi elsősorban olyan paraméterezett  $\pi$  stratégiák vizsgálatában vagyunk érdekeltek, amelyeknek sokkal kevesebb paramétere van, mint ahány állapot található az állapottérben (éppúgy, mint az előző alfejezetben). Például paraméterezett  $Q$ -függvényekkel reprezentálhatjuk a  $\pi$  stratégiát, egy-egy  $Q$ -függvényt rendelve minden cselekvéshez, és azt a cselekvést választjuk, amely a legnagyobb jóolt értéket adja:

$$\pi(s) = \max_a \hat{Q}_\theta(a, s) \quad (21.13)$$

Minden egyes  $Q$ -függvény lehet a  $\theta$  paraméterek lineáris függvénye, mint a (21.9) egyenletben, vagy lehet nemlineáris függvény, mint például egy neurális háló. A stratégakeresés ezek után a  $\theta$  paramétereit változtatja oly módon, hogy javítsa a stratégiát. Vegyük észre, hogy ha a stratégiát  $Q$ -függvényekkel reprezentáljuk, akkor a stratégakeresés egy  $Q$ -függvény tanulási eljárást eredményez. Ez a folyamat azonban nem azonos a  $Q$ -tanulással! A függvényapproximációt használó  $Q$ -tanulásban az algoritmus



egy olyan  $\theta$ -t keres, amelyre a  $\hat{Q}_\theta$  „közel” van  $Q^*$ -hoz, az optimális  $Q$ -függvényhez. A stratégiakeresés ezzel szemben olyan  $\theta$ -t keres, amely jó működést eredményez, az eredménytől kapott értékek alapvetően eltérhetnek.<sup>6</sup> Egy másik nyilvánvaló példa a két dolog különbségére az az eset, amikor a  $\pi(s)$  stratégiát úgy számítjuk, hogy 10 lépés mélyében előretékintő keresést végezünk egy közelítő  $\hat{U}_\theta$  hasznosságfüggvény alapján. A jó eredményt adó  $\theta$  messze lehet attól, amely az  $\hat{U}_\theta$ -t az igazi hasznosságfüggvényhez hasonlóvá teszi.

A (21.13) egyenletben bemutatott stratégiareprezentáció egyik problémája, hogy diszkrét cselekvések esetén a stratégia a paraméterek *nemfolytonos* függvénye.<sup>7</sup> Azaz lesznek olyan  $\theta$  értékek, amelyeknél végletes kis változás a  $\theta$ -ban azt eredményezi, hogy a stratégia egyik cselekvésről a másikra vált. Ez azt is jelenti, hogy a stratégia értéke is változhat nemfolytonos módon, ami nehézzé teszi a gradiensalapú eljárások alkalmazását. Emiatt a stratégiakeresési eljárások gyakran használják a  $\pi_\theta(s, a)$  **sztochasztikus stratégia (stochastic policy)** reprezentációt, amely az  $s$  állapotban az  $a$  cselekvés választásának valószínűségét specifikálja. Egy népszerű reprezentáció a **ssoftmax függvény (softmax function)**:

$$\pi_\theta(s, a) = \exp(\hat{Q}_\theta(a, s)) / \sum_{a'} \exp(\hat{Q}_\theta(a', s))$$

A softmax függvény közel determinisztikussá válik, ha az egyik cselekvés sokkal jobb, mint a többi, de minden differenciálható  $\theta$  szerint; így a stratégia értéke (amely folytonos módon függ a cselekvésválasztási valószínűségtől) differenciálható függvény lesz  $\theta$ -nak.

Nézzünk most módszereket a stratégia javítására. Kezdjük a legegyszerűbb esettel: determinisztikus stratégia, determinisztikus környezetben. Ebben az esetben a stratégia értékelése triviális: egyszerűen végrehajtjuk a stratégiát, és megfigyeljük az összegyűjtött jutalmat, ez adja számunkra a **stratégia értékét (policy value)**,  $\rho(\theta)$ -t. A stratégia javítása ezek után egy standard optimalizációs probléma, ilyeneket a 4. fejezetben tárgyalunk. Követhetjük a **stratégiagradiens (policy gradient)** vektort,  $\nabla_\theta \rho(\theta)$ -t, feltéve, hogy  $\rho(\theta)$  differenciálható. Másik lehetőség, ha az **empirikus gradienst (empirical gradient)** követjük hegemászó módszerrel – azaz kiértékeljük a stratégiát minden egyes paraméter kis megváltozása esetére. A szokásos feltételek esetén ez a stratégia-tér egy lokális optimumához fog konvergálni.

Ha a környezet (vagy a stratégia) sztochasztikus, akkor a helyzet nehezebbé válik. Tegyük fel, hogy hegemászó módszert próbálunk alkalmazni, ami azt kívánja, hogy összehasonlítsuk  $\rho(\theta)$ -t  $\rho(\theta + \Delta\theta)$ -val valamelyen kis  $\Delta\theta$  esetén. Az a probléma, hogy a teljes jutalom nagyon nagyokat változhat kísérletről kísérletre, így a kisszámú kísérletből számított stratégiaérték nagyon megbízhatatlan lesz – két ilyen becslés összehasonlítása pedig még megbízhatatlanabb. Egy lehetséges megoldás, ha egyszerűen sok kísérletet futtatunk le, és a minta variancájával mérjük, hogy elegendő kísérletet futtattunk-e már ahhoz, hogy megbízhatónan jelezni tudjuk a  $\rho(\theta)$  javításának irányát. Sajnálatos módon ez sok valós probléma esetén nem praktikus, mert az egyes kísérletek drágák, időigényesek és esetleg veszélyesek lehetnek.

<sup>6</sup> Triviális, hogy a  $\hat{Q}_\theta(a, s) = Q^*(a, s)/10$  közelítő  $Q$ -függvény optimális teljesítményt ad, bár egyáltalán nincs közel  $Q^*$ -hoz.

<sup>7</sup> Folytonos cselekvési terekre a stratégia a paraméterek sima függvénye lehet.

A  $\pi_\theta(s, a)$  sztochasztikus stratégia esetén lehetőségünk van arra, hogy a  $\theta$ -ban végrehajtott kísérletek eredményei alapján a  $\nabla_\theta \rho(\theta)$  gradiens torzítatlan becslését állítsuk elő  $\theta$ -ban. Az egyszerűség kedvéért a becslést arra az egyszerű esetre vezetjük le, amikor nem szekvenciális környezetben közvetlenül az  $s_0$  startállapotbeli cselekvés után megkapjuk a jutalmat. Ebben az esetben a stratégiaérték egyszerűen a jutalom várható értéke, tehát:

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0, a) R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0, a)) R(a)$$

Most egy egyszerű trükköt alkalmazunk, hogy ezt az összegzést approximálni tudjuk a  $\pi_\theta(s, a)$  által meghatározott valószínűség-eloszlásból generált mintákkal. Tegyük fel, hogy összesen  $N$  kísérletünk van, és a  $j$ -edik kísérletben az  $a_j$  cselekvést választottuk. Ekkor

$$\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{(\nabla_\theta \pi_\theta(s_0, a)) R(a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^N \frac{(\nabla_\theta \pi_\theta(s_0, a_j)) R(a_j)}{\pi_\theta(s_0, a_j)}$$

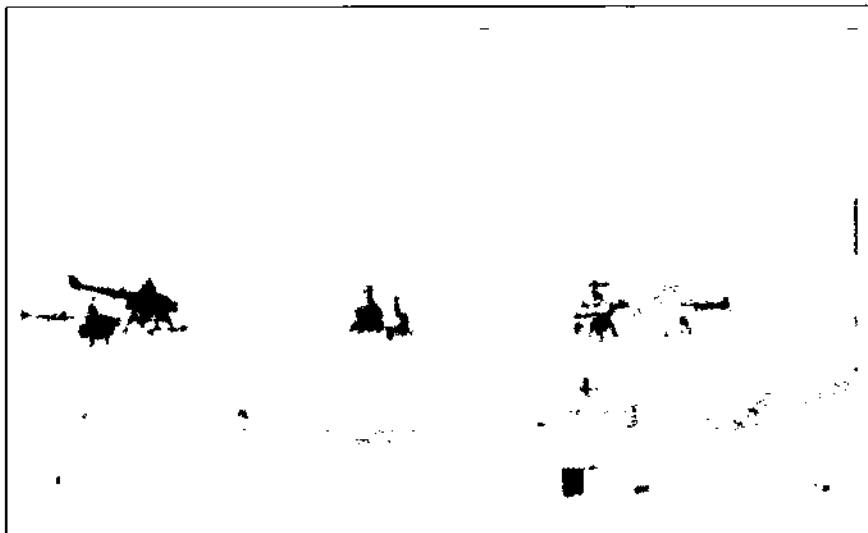
Tehát a stratégia igazi gradiensét egy szummával közelítettük, ahol a tagok az egyes kísérletek cselekvés-választási valószínűségeinek gradiensét tartalmazzák. A szekvenciális esetre ez a következőképp általánosítható:

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{(\nabla_\theta \pi_\theta(s, a_j)) R_j(s)}{\pi_\theta(s, a_j)}$$

minden egyes bejárt  $s$  állapotra. Itt  $a_j$  a  $j$ -edik kísérlet során az  $s$  állapotban választott cselekvés, és  $R_j(s)$  a  $j$ -edik kísérlet során az  $s$  állapotból kiindulva gyűjtött összes jutalom. Az eredményül kapott algoritmust REINFORCE-nak neveztek el (Williams, 1992). Rendszerint sokkal hatékonyabb, mint a minden egyes  $\theta$ -ban sok kísérletet használó hegymászó eljárás. Viszont sajnos még mindig sokkal lassabb annál, mint amire szükségünk lenne.

Vizsgáljuk a következő feladatot: adott két blackjack<sup>8</sup> program, döntsük el, hogy melyik a jobb! Egyik lehetőség, ha egy közös „bank” ellen játszatjuk őket egy adott számú kártyaleosztásban, és megnézzük, hogy melyik nyert többet. Ezzel az a probléma – mint láttuk –, hogy mindegyik program nyereségei nagy fluktuációt fognak mutatni annak függvényében, hogy milyen lapokat kapott. Egy kézenfekvő megoldás erre, ha előre generálunk egy sor leosztást, egy *leosztáshalmazt*. Ezzel elkerüljük a különböző kártyaleosztások okozta mérési hibát. Ez a PEGASUS algoritmus ötletének alapja (Ng és Jordan, 2000). Az algoritmus olyan területeken alkalmazható, ahol rendelkezésünkre áll egy szimulátor, ezáltal a „véletlen” kísérlet kimenetelek megismételhetővé válnak. Az algoritmus  $N$  véletlen számsorozatot generál előre, mindenkor felhasználható arra, hogy egy tetszőleges stratégia alapján kísérletet futtassunk vele. A stratégiakeresést úgy hajtjuk végre, hogy mindenkor stratégiájelöltet az alapján értékelünk, hogy ugyanazt a véletlen sorozat halmazt használja a kísérletek kimeneti értékeinek meghatározásához. Megmutatható, hogy az összes stratégia értékének jó becsléséhez szükséges véletlen sorozatok száma csak a stratégiatér komplexitásától függ, és egyáltalán nem függ a mögöttes terület komplexitásától. A PEGASUS algoritmussal számos területen (például az autonóm helikoptervezetés területén) több hatékony stratégiát fejlesztettek ki (lásd 21.10. ábra).

<sup>8</sup> 21-es játékként is ismert.



**21.10. ábra.** Időben elolt képek egymásra másolása útján kapott eredő kép, amelyen egy nagyon bonyolult „körberrepülés orral a köt középpontja fele” manővert hajtanak végre. A helikoptert egy PEGASUS stratégiakereső algoritmussal fejlesztett stratégiával vezérlék. Egy szimulátorral fejlesztek a valós helikopter egyes vezérlési beavatkozásokra adott válaszainak vizsgálatára. Ezek után egész éjszaka futatták az algoritmust a szimulátoron. Egy sor vezérlőt fejlesztettek ki különböző manőverekre. minden esetben, amikor távirányítást használtak, messze jobb volt az eredmény, mint a képzett humán pilótáké. (A képet Andrew Ng engedélyével közöljük.)

## 21.6. ÖSSZEFoglalás

Ebben a fejezetben a megerősítéssel tanulással foglalkoztunk: azzal, hogy egy ismeretlen környezetben működő ágens hogyan válhat gyakorlottá pusztán megfigyelései, valamint az esetenként kapott jutalmak felhasználásával. A megerősítéssel tanulás a teljes MI-témakör mikrovilágának is tekinthető, amelyet azonban az előrehaladás érdekében egy sor egyszerűsítő feltétel mellett tárgyalunk. A következő fontosabb megállapításokat tettük:

- Az ágens teljes felépítése határozza meg azt, hogy milyen információt kell megtanulni. A tárgyalt három fő felépítési lehetőség: egyrészt a modellalapú, amely a  $T$  modellt és az  $U$  hasznosságfüggvényt használja; másrészt a modell nélküli felépítés, amely a  $Q$  cselekvésérték-függvényt használja, és a reflexszerű, amely a  $\pi$  stratégiát használja.
- A hasznosságértékek háromféle megközelítés alapján tanulhatók:
  1. A közvetlen hasznosságbecslés (direct utility estimation), amely a megfigyelt hátralévő jutalmat használja egy-egy állapot esetén, mint a hasznosság tanulásának közvetlen jellemzőjét.
  2. Az adaptív dinamikus programozás (ADP) (adaptive dynamic programming) megfigyelései alapján egy modellt és egy jutalomfüggvényt tanul, ezek után érték-

vagy stratégiaiterációt használ ahhoz, hogy az állapotok hasznosságbecsléséhez vagy az optimális stratégiához eljusson. Az ADP optimálisan használja fel a környezetnek a szomszédos állapotokra vonatkozó struktúrájából adódó lokális kényezeteket.

3. Az **időbeli különbség (IK) (temporal-difference, TD)** megközelítés úgy módosítja a hasznosságbecsléseket, hogy megfeleljenek az adott állapotot követő állapotokra vonatkozó becsléseknek. Tekinthetjük egyszerűen az ADP megközelítés olyan approximációjának is, amely a tanuláshoz nem igényel modellt. Mindamellett, ha a megtanult modellt pszeudotapasztalatok generálására használjuk, gyorsabb tanuláshoz juthatunk.
- A cselekvésérték-függvények vagy  $Q$ -függvények ADP- vagy IK-megközelítéssel tanulhatók. IK esetén a  $Q$ -tanulás sem a tanulás, sem a cselekvés kiválasztásának fázisában nem igényli modell használatát. Ez egyszerűsíti a tanulási feladatot, de potenciálisan korlátozhatja a bonyolult környezetben való tanulási képességet, mivel az ágens nem képes szimulálni a lehetséges cselekvéssorozatok eredményét.
- Ha a tanuló ágens a tanulás során a cselekvések kiválasztásáért is felelős, akkor kompromisszumot kell kötnie a választott cselekvés jelenlegi hasznossága és hasznos új információ tanulásának lehetősége között. A felfedezési probléma egzakt megoldása kezelhetetlen, de néhány egyszerű heurisztika elég jó megoldásra vezet.
- Nagy állapotterek esetén a megerősítéses tanuló algoritmusnak közelítőfüggvény-reprezentációt kell használnia az állapotok feletti általánosítóképesség érdekében. Az időbeli különbség jel közvetlenül felhasználható a neurális háló jellegű reprezentációk paramétereinek frissítésére.
- A **stratégakeresési** módszerek közvetlenül a stratégia reprezentációját használják, a megfigyelt teljesítmény alapján próbálva meg javulást elérni. Súlyos problémát okoz sztochasztikus területeken a teljesítményben mutatkozó nagy variancia. Szimulációval vizsgálható területeken ez megoldható úgy, hogy előre rögzítjük a véletlen hatásokat.

A vezérlési stratégiát biztosító program kézi kódolásának elkerülhetősége miatt a megerősítéses tanulás a gépi tanulás egyik legaktívabban művelt területe. A robotikai alkalmazások különösen értékesnek ígérkeznek, de ezek *folytonos, sokdimenziós, csak részlegesen megfigyelhető* környezetek kezelésére képes módszereket igényelnek, ráadásul a sikeres működéshez több ezer vagy akár millió elemi cselekvésre is szükség lehet.

## Irodalmi és történeti megjegyzések

Turing javasolta a megerősítéses tanulás megközelítést, bár nem volt teljesen meggyőződve annak hatékonyságáról (Turing, 1948; 1950). Azt írta: „A büntetés és jutalmazás legjobb esetben is csak egy része lehet a tanítási folyamatnak.” Arthur Samuel munkája volt valószínűleg az első sikeres kutatás a gépi tanulás területén (Samuel, 1959). Bár munkája nem volt formálisan megfogalmazva, és számos hiányossága volt, mégis a megerősítéses tanulás legtöbb modern gondolatát tartalmazta, beleértve az időbeli különbségek képzést és a függvényekkel történő approximációt. Körülbelül ugyanebben az időben az adaptív szabályozások területével foglalkozó kutatók (Widrow és Hoff, 1960), Hebb munkájára építve (Hebb, 1949), egyszerű hálókat tanítottak a delta-szabály segítségével.

(A neurális hálók és a megerősítéses tanulás ezen korai kapcsolata vezethetett ahhoz a félreérteshez, hogy ez utóbbi az előbbinek részterülete.) Michie és Chambers rúd-egyenlőtlenség-vezérlési kutatása szintén egy függvényapproximációval történő megerősítéses tanulási módszernek tekinthető (Michie és Chambers, 1968). A megerősítéses tanulással foglalkozó pszichológiai irodalom sokkal régebbi, Hilgard és Bower jó áttekintést adnak erről (Hilgard és Bower, 1975). A méhek nektárgyűjtési szokásainak kutatása közvetlen bizonyítékokat szolgáltatott a megerősítéses tanulás működésére állatoknál. Nyilvánvaló neurális kapcsolat van a jutalomjellel. Ennek formája egy nagy neurális leképezés a nektár bevitelét érzékelő szenzoruktól a motoros kéregig (Montague és társai, 1995). A sejtszintű emlékezéssel kapcsolatban folytatott kutatások azt sugallják, hogy a főemlősök agyában a dopamin rendszer valami olyasmit valósít meg, ami a függvénytanulásra emlékeztet (Schultz és társai, 1997).

A megerősítéses tanulás és a Markov döntési folyamatok közti kapcsolatra először Werbos mutatott rá (Werbos, 1977), de a megerősítéses tanulás MI-n belüli fejlesztése a University of Massachusetts falain belül kezdődött az 1980-as évek elején (Barto és társai, 1981). Sutton írása jó történeti áttekintést ad (Sutton, 1988). Ennek a fejezetnek a (21.3) egyenlete egy speciális, a  $\lambda = 0$  melletti esete Sutton általános IK( $\lambda$ ) algoritmusának. Az IK( $\lambda$ ) egy sorozat összes állapotának értékét frissíti oly módon, hogy a múltban  $t$  időlénysorozatban lévő átmenetekhez egy  $\lambda'$  szerint csökkenő faktor vezet. Az IK(1) azonos a Widrow–Hoff- vagy delta-szabályal. Bradtke és Barto érvelésére (Bradtk és Barto, 1996) építve Boyan azt állítja, hogy az IK( $\lambda$ ) és a hozzá kapcsolható algoritmusok nem használják ki hatékonyan a tapasztalatokat (Boyan, 2002). Szerinte ezek lényegében online regressziós algoritmusok, amelyek sokkal lassabban konvergálnak, mint az offline regresszió. Az  $\delta$  LSTD( $\lambda$ ) algoritmus egy olyan online regresszió, ami az offline regresszióval azonos eredményt ad.

Az időbeli különbség tanulásnak és a szimulációs tapasztalatok modellbázisú generálásának kombinálását Sutton a DYNAmic architektúrában javasolta (Sutton, 1990). A prioritásos végigsoprás ötletét egymástól függetlenül Moore és Atkeson (Moore és Atkeson, 1993), illetve Peng és Williams (Peng és Williams, 1993) vetették fel. A  $Q$ -tanulást Watkins dolgozta ki PhD-disszertációjában (Watkins, 1989).

A rabló problematikát, amely a nemszekvenciális döntések területén történő felfedezést modellezte, részletesen Berry és Fristedt tanulmányozták (Berry és Fristedt, 1985). Számos helyzetben optimális felfedezési stratégia nyerhető a Gittins-index segítségével (Gittins, 1989). Barto és munkatársai a szekvenciális döntési problémákban alkalmazott felfedezési módszerek számos változatát tárgyalják (Barto és társai, 1995). Kearns és Singh, valamint Brafman és Tennenholz olyan algoritmusokat írnak le (Kearns és Singh, 1998; Brafman és Tennenholz, 2000), amelyek ismeretlen környezet felfedezését végezik, és bizonyított, hogy polinomiális idő alatt konvergálnak közel-optimális stratégiákhoz.

A megerősítéses tanulás területén a függvényapproximáció alkalmazása Samuel munkájáig vezethető vissza, aki mind lineáris, mindenlineáris kiértékelő függvényeket alkalmazott, továbbá a tulajdonságok terének csökkentésére tulajdonságselekciós algoritmusokat is használt. A későbbiekben bevezetett módszerek közé tartozik a CMAC (Cerebellar Model Articulation Controller) (Albus, 1975), amely lényegében átlapolódó lokális kernelfüggvények összegéből, és a Barto és társai által bevezetett asszociatív neurális hálókból áll (Barto és társai, 1983). Napjainkban a neurális hálók a legnép-

szerűbb függvényapproximátorok. A legismertebb alkalmazás az ebben a fejezetben bemutatott TD-Gammon (Tesauro, 1992, 1995). A neurális hálókkal megvalósított IK-tanulóknak súlyos problémája az, hogy hajlamosak elfelejteni korábbi tapasztalataikat. Ez különösen igaz azokra térrészekre, amelyeket a szakértelmük megszerzése után elkerülnek. Ez katasztrofális következményekkel járhat, ha helyzetek újra jelentkeznek. A példányalapú tanulás (**instance-based learning**) alkalmazásával ez a probléma elkerülhető (Ormoneit és Sen, 2002; Forbes, 2002).

A függvényapproximációt használó megerősítéses tanulás konvergenciája kimondottan technikai kérdés. Lineáris approximáló függvények alkalmazása esetén az IK tanulási eredmények folyamatosan javulnak (Sutton, 1988; Dayan, 1992; Tsitsiklis és Van Roy, 1997), de nemlineáris függvények esetén számos divergenciát mutató példát találtak (a jelenség tárgyalását lásd a (Tsitsiklis és Van Roy, 1997)-ben). Papavassiliou és Russell a megerősítéses tanulás új formáját adták (Papavassiliou és Russell, 1999), amely tetszőleges struktúrájú függvényapproximátor esetén konvergál, feltéve, hogy található a megfigyelt adatokra egy legjobban illeszkedő approximáció.

A stratégiakeresésre Williams hívta fel a figyelmet (Williams, 1992), ō volt az, aki kifejlesztette a REINFORCE algoritmuscsaládot. A későbbi munkák megerősítették és általánosították a stratégiakeresés konvergenciájára vonatkozó eredményeket: (Marbach és Tsitsiklis, 1998; Sutton és társai, 2000; Baxter és Bartlett, 2000). Ng és Jordan (2000) alkották meg a PEGASUS algoritmust (Ng és Jordan, 2000), bár hasonló technikák már Van Roy PhD-disszertációjában is megtalálhatók (Van Roy, 1998). Mint a fejezetben említettük, a *sztochasztikus* stratégiák teljesítménye a paramétereik folytonos függvénye, ami elősegíti a gradiensalapú technikák alkalmazhatóságát. Nem ez az egyetlen előnyük: Jaakkola és társai amellett érvelnek, hogy részlegesen megfigyelhető környezetben a sztochasztikus stratégiák jobban működnek, mint a determinisztikusok, ha minden stratézia esetén a cselekvés pusztán a jelenlegi észleléseren alapul (Jaakkola és társai, 1995). (Ennek egyik oka, hogy a sztochasztikus algoritmusok kevésbé hajlamosak „beragadni”, ha valamilyen eddig még nem látott akadály jelentkezik.) A 17. fejezetben rámutattunk, hogy részlegesen megfigyelhető MDF-ekben az optimális stratégiák inkább a *hiedelemállapot* determinisztikus függvényei, mint az aktuális megfigyeléseké. Ennek megfelelően még jobb eredményeket várnánk, ha a 15. fejezetben tárgyalt **szűrési (filtering)** módszerekkel nyomon követnénk a hiedelemállapot alakulását. Sajnálatos módon a hiedelemállapot-terek sokdimenziósak és folytonosak, és az eddigiekben nem sikerült hatékony módszereket kifejleszteni a hiedelemállapotok megerősítéses tanulására.

A valós környezetekre az is jellemző, hogy gyakran rengeteg elemi cselekvési lépésre van szükség ahhoz, hogy jelentős jutalomhoz jussunk. Például ha egy robot focizik, akkor százezer elemi lábmozdulatot is lehet, mielőtt gólt lő. Egy szokásos módszer erre a **jutalommalakítás (reward shaping)**, melyet eredetileg az állatok tanításánál alkalmaztak. Ennek része, hogy további jutalmakat adunk az ágensnek, ha az „haladást ér el”. A foci esetén például jutalmat adhatunk, ha hozzáér a labdához vagy a kapu felé lő. Ezeket a jutalmakat rendszerint nagyon egyszerű kialakítani, és rendkívül felgyorsíthatják a tanulást, de megvan az a veszélye, hogy az ágens a tanulás során a „pszeudojutalmak” maximálását végzi el, nem az igazi jutalmakét. A foci tanulása esetén például a labdához való közelség és a labdával való sok-sok érintkezés, a „vibrálás”, lehet ennek az eredménye. Ng és társai megmutatták, hogy az ágens így is megtanulja az optimális stratégiát, ha az  $F(s, a, s')$  pszeudojutalom kielégíti az  $F(s, a, s') = \gamma \Phi(s') - \Phi(s)$

egyenletet, ahol  $\Phi$  az állapot tetszőleges függvénye (Ng és társai, 1999). A  $\Phi$ -t úgy alkothatjuk meg, hogy az állapot valamely kívánatos vonásait tükrözze, például valamilyen részről elérését vagy a célállapottól való távolságát.

Az összetett viselkedés kialakítását **hierarchikus megerősítéses tanulási** (**hierarchical reinforcement learning**) módszerekkel segíthetjük elő, amelyek több absztrakciós szinten igyekeznek megoldani a problémát, a 12. fejezetben ismertetett **HFH-tervkészítéshez** (**HTN planning**) hasonlóan. Például a „góllövés” szétfontható „labdaszerzés”, „a kapu felé cselezés” és „lövés” részekre, és ezek minden tovább bonthatók alacsonyabb szintű mozgásokra. Ezen a területen Forestier és Varaiya értek el alapvető eredményt (Forestier és Varaiya, 1978). Bebizonyították, hogy tetszőleges bonyolultságú alsóbb szintű viselkedés egyszerűen elemi cselekvésnek tekinthető az ott kiváltó magasabb szintű viselkedés szempontjából (bár az alsóbb szintű viselkedés által megvalósított cselekvések eltérő időt vehetnek igénybe). Parr és Russell, Dietterich, Sutton és társai, valamint Andre és Russell jelen kutatásaiban erre az eredményre építettek: olyan módszereket fejlesztettek, amelyek **részprogrammal** (**partial program**) lábják el az ágenst, ezzel viselkedését valamilyen speciális hierarchikus struktúrába sorolva (Parr és Russell, 1998; Dietterich, 2000; Sutton és társai, 2000; Andre és Russell, 2002). A megerősítéses tanulást ezek után arra használják, hogy a részprogrammal konzisztens legjobb viselkedést tanítsák. A függvényapproximáció, a jutalommalakítás és a hierarchikus megerősítéses tanulás összekombinálása lehetővé teheti, hogy nagy bonyolultságú problémák megoldását kíséreljük meg a sikeres reményében.

Az irodalom áttekintéséhez jó kiindulópontot ad Kaelbling tanulmánya (Kaelbling és társai, 1996). A terület két úttörőjénk, Suttonnak és Bartónak a publikációja az architektúrákra és algoritmusokra koncentrál, megmutatva, hogy a megerősítéses tanulás összeköti a tanulás, a tervezés és a cselekvés elveit (Sutton és Barto, 1998). Bertsekas és Tsitsiklis inkább technikai részletekkel foglalkozó munkája a dinamikus programozás és a sztochasztikus konvergencia elméletének precíz megalapozását nyújtja (Bertsekas és Tsitsiklis, 1996). A *Machine Learning*, illetve a *Journal of Machine Learning Research* folyóiratok, továbbá az International Conferences on Machine Learning és a Neural Information Processing Systems rendezvények kiadványai közölnek gyakran megerősítéses tanulással foglalkozó cikkeket.

## Feladatok

- 21.1.** Egy – a  $4 \times 3$ -as világhoz hasonló – egyszerű környezetben valósítson meg egy ~~passzív~~ passzív tanuló ágenst! Hasonlítsa össze a közvetlen hasznosságbecslést, az IK-t és az ADP-t előzetesen nem ismert környezeti modellek esetén! Végezze el az összehasonlítást az optimális stratégiára és több, véletlen módon generált stratégiára! Melyikre konvergálnak gyorsabban a hasznosságbecslések? Mi történik, ha növeli a környezet méretét? (Vizsgáljon pályákat akadályokkal és azok nélkül!)
- 21.2.** A 17. fejezetben úgy definiáltuk az MDF számára **megfelelő stratégiát** (**proper policy**), mint ami biztosan eléri a végállapotot. Mutassa meg, hogy egy passzív ADP-ágens képes egy olyan állapotátmenet-modell megtanulására, amelyre nézve  $\pi$  stratégiája nem megfelelő, még ha  $\pi$  megfelelő is az igazi MDF-re; az ilyen

modellekkel  $\gamma = 1$  esetben az értékmeghatározási lépés meghiúsulhat! Mutassa meg, hogy ez a probléma nem jelentkezik, ha az értékmeghatározást csak a kísérlet végeztével alkalmazzuk a megtanult modellre.

- 21.3.** Induljon ki egy passzív ADP-ágensből, és módosítsa úgy, hogy a fejezetben leírt közelítő ADP-algoritmust kapja! Két lépésben oldja meg a feladatot:
- Építse fel egy prioritássort a hasznosságbecslések módosítására! Valahány-szor állapotmódosításra kerül, az öt megelőző állapotok is potenciálisan módosítandók, tehát be kell állítani őket a sorba. A sor inicializálását annak az állapotnak a felhasználásával végezzük, amelyből a legutolsó átmenet történt. Csak rögzített számú módosítást engedjen meg!
  - Kísérletezzen a prioritássor rendezésének különböző heurisztikáival, vizsgálja meg a tanulás sebességére és a számítási időre gyakorolt hatásukat!
- 21.4.** A 21.2. alfejezetben bemutatott közvetlen hasznosságbecslési módszerek elkülönített végállapotokat használtak a kísérlet végének jelzésére. Hogyan módosíthatók végállapottal nem rendelkező, leérintélt jutalmakat használó környezetekre?
- 21.5.** Hogyan használható az értékmeghatározási algoritmus úgy, hogy egy  $U$  hasznosságbecslés-halmazt és egy  $M$  becsült modellt használó ágensnek a helyes értékeket használó ágenshez képesti várható veszteségét számítsa?
- 21.6.** Adaptálja a 2. fejezetben bemutatott porszívóvilágot megerősítéses tanulásra! Az ágens kapjon jutalmat minden felszívott szemétért, a helyére való visszaérés-kor és kikapcsoláskor! Megfelelő észlelésekkel tegye megfigyelhetővé a világot! Kísérletezzen különböző megerősítéses tanulást végző ágensekkel! Szükség van-e függvényapproximációra a sikeres érdekelőben? Milyen approximátor működik erre az alkalmazásra?
- 21.7.** Valósítson meg egy felfedező megerősítéses tanuló ágenst, amely közvetlen hasznosságbecslést végez. Készítsen két változatot, az egyik táblázatos reprezentációt használjon, a másik a (21.9) egyenlet függvényapproximátorát. Hasonlítsa össze teljesítményüket az alábbi három különböző környezetben:
- A fejezetben leírt  $4 \times 3$ -as világ.
  - Egy  $10 \times 10$ -es világ, amelyben nincsenek akadályok, és a  $(10, 10)$  mezőn van +1 jutalom.
  - Egy  $10 \times 10$ -es világ, amelyben nincsenek akadályok, és az  $(5, 5)$  mezőn van +1 jutalom.
- 21.8.** Írja fel az IK-tanulás paraméterfrissítési egyenletét a következő esetre:
- $$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$$
- 21.9.** Találjon ki megfelelő jellemzőket a sztochasztikus négyzetrácvilágra (a  $4 \times 3$ -as világ általánosítása), amely világ több akadályt és több (+1, illetve -1 jutalommal) jellemzhető) végállapotot tartalmaz!

**21.10.** Számítsa ki a valós hasznosságfüggvényt és az  $x$ ,  $y$  függvényében felírható legjobb lineáris approximációt (éppúgy, mint a (21.9) egyenletben tettük) a következő környezetekre:

- (a) Egy  $10 \times 10$ -es világ, amelyben a  $(10, 10)$  az egyetlen +1 végállapot mező.
- (b) Mint (a), de a  $(10, 1)$ -en van egy további -1 végállapot.
- (c) Mint (b), de tegyen 10 véletlen módon választott mezőre akadályt.
- (d) Mint (b), de vegyen fel egy falat  $(5, 2)$ -től  $(5, 9)$ -ig.
- (e) Mint (a), de a végállapot legyen az  $(5, 5)$ .

A cselekvések a négy lehetséges irányba való determinisztikus lépések. minden esetben háromdimenziós ábrákat használva végezze el az összehasonlítást! Mindegyik környezetben adjon javaslatot további olyan tulajdonságokra ( $x$ -en és  $y$ -on túl), amelyek javíthatják az approximációt, majd ábrázolja az eredményeket!

**21.11.** Terjessze ki a standard játékkörnyezetet (6. fejezet) úgy, hogy tartalmazzon játalomjelet is! Helyezzen két ágenst ebbe a környezetbe (természetesen osztozhatnak az ágensprogramon), és játszassa egymás ellen őket! Alkalmazza az általánosított IK frissítési szabályt (21.11 egyenlet) a kiértékelő függvény frissítésére. Lehet, hogy érdemes egy egyszerű súlyozott lineáris kiértékelő függvényel kezdeni, és egy egyszerű játékkal, mint a  $3 \times 3$ -as amőba (tic-tac-toe).

**21.12.** Valósítsa meg a REINFORCE és PEGASUS algoritmusokat a  $4 \times 3$ -as világra, és alkalmazza egy saját maga által választott stratégiacsaládra! Értékelje az eredményeket!

**21.13.** Vizsgálja meg a megerősítéses tanulás elveinek humán, illetve állati viselkedésre való alkalmazását!

**21.14.** Alkalmas absztrakt modellje-e az evolúciónak a megerősítéses tanulás? Milyen kapcsolat van – ha van egyáltalán – a beépített jatalomjelek és az evolúciós rátermeuség között?

# VII. RÉSZ

# KOMMUNIKÁCIÓ,

# ÉSZLELÉS ÉS

# CSELEKVÉS

---

# 22. KOMMUNIKÁCIÓ

*Ebben a fejezetben meglátjuk, hogy az ágensek miért akarhatnak információt tartalmazó üzeneteket küldeni egymásnak, illetve azt, hogy ezt hogyan tehetik meg.*

Alkonyodik az Amboseli Nemzeti Park szavannás erdőségében, közel a Kilimandzsáró tövéhez. A fehérbarkójú cerkófmajom vagy grivet (*Cercopithecus aethiops – a ford.*) egy csoportja épp élelem után kutat, amikor az egyik grivet hangos kiáltó hívást hallat. A csoport többi tagja felismeri, hogy ez egy leopárdra figyelmeztető felhívás (amely különbözik a sasra figyelmeztető rövid köhrintéstől vagy a kígyókra vonatkozó sziszegéstől), és a fákra sietnek. A grivet sikeresen kommunikált a csoporttal.

A **kommunikáció** (*communication*) az információ szándékolt átadása közös jelrendszerből választott jel létrehozása és észlelése által. Az állatok többsége használ jelzéseket fontos üzenetek kifejezésére: itt van élelem, ragadozó a közelben, gyere közelebb, vonulj vissza, párosodjunk. Egy részlegesen megfigyelhető világban a kommunikáció segít az ágenseknek sikeresnek lenni, mivel megfigyelhetnek mások által észlelt vagy kikövetkeztetett információt.

Az embereket a **nyelvként** (*language*) ismert, strukturált üzenetekből álló összetett rendszer különbözteti meg más élőlényektől, amely lehetővé teszi számunkra, hogy a világról szóló ismereteink nagy részét átadhassuk egymásnak. Bár a csimpánzok, delfinek és más emlősök több száz jeles szókincsel rendelkeznek, és van valamiféle hajlamuk ezek összekapcsolására, az ember az egyedüli faj, amely minőségileg különböző üzenetek korlátlan számát képes megbízhatóan átadni.

Természetesen vannak más tulajdonságok, amelyek egyedülállóan emberiek: egyetlen más faj sem visel ruhát, készít művészeti alkotásokat vagy néz naponta három órán át televíziót. Azonban amikor Turing a tesztjét javasolta (lásd 1.1. alfejezet), a nyelvre alapozta azt, mivel a nyelv bensőséges kapcsolatban áll a gondolkodással. Ebben a fejezetben elmagyarázzuk, hogyan működik egy kommunikáló ágens, és bemutatjuk az angol nyelv egy egyszerűsített részletét.

## 22.1. A KOMMUNIKÁCIÓ MINT CSELEKVÉS

Egy ágens számára a beszéd létrehozása lehetséges cselekvéseinek egyike. Ezt **szólaš-aktusnak** (*speech act*) nevezzük. A „beszédet” olyan értelemben használjuk, mint „szabad közlést”, nem mint „beszélgetést”, így az elektronikus levelezés, a mutogatás és a jelbeszéd használata mind szólašaktusnak számít. Az angol nyelvben nincs általános szó a beszédet létrehozó ágensre, amely beszél, ír vagy másképp fejezi ki magát. A **beszélő** (*speaker*), a **hallgató** (*listener*) és a **megnyilatkozás** (*utterance*) kifejezése-

ket használjuk a kommunikáció bármilyen módjának általános leírására. A szó (**word**) kifejezést használjuk mindenféle szokásos kommunikációs jelre.

Miért foglalkozna egy ágens szólásaktus létrehozásával, amikor „hagyományos” cselekvést is végrehajthat? A 12. fejezetben láttuk, hogy többágenses környezetekben az ágensek használhatják a kommunikációt annak érdekében, hogy közös tervet alakítanak ki. Például a wumpus világot felfedező ágensek egy csoportja együtt (egyéni vagy csoportos) előnyhöz jut, ha képes a következőt megtenni:

- **Megkérdezik (query)** egymást a világ különfélé területeiről. Ezt tipikusan kérdések feltevésével végzik: *Érezted a wumpus bűzét valahol?*
- **Értesítik (inform)** egymást a vilagról. Ezt kijelentésekkel teszik meg: *Szellő van itt, a [3, 4]-en*. Egy kérdés megválaszolása egy másik fajta értesítés.
- **Kérnek (request)** más ágenseket cselekvések végrehajtására: *Kérlek segíts az aranyat vinni!* Néha egy **indirekt szólásaktus (indirect speech act)** (egy kijelentés vagy kérdés formájú kérés) udvariasabbnak számít: *Használni tudnék egy kis segítséget ennek a cipelésében*. Egy fennhatósággal rendelkező ágens adhat parancsokat (*Alfa jobbra menj; Bravo és Charlie balra*), és egy hatalommal rendelkező ágens fenyegethet (*Ide nekem az aranyat, vagy...*). Ezen szólásaktusokat együtt **direktívának (directives)** nevezik.
- **Tudomásul veszik (acknowledge)** a kéréseket: *Rendben*.
- **Megírnek (promise)** vagy hozzájárulnak egy tervhez: *Én lelövöm a wumpust; te megragadod az aranyat*.

Minden szólásaktus befolyásolja a világot a levegőmolekulák rezgetésével (vagy ekvivalens hatással más médiumban), és ily módon megváltoztatja más ágensek mentális állapotát, valamint végső fokon a jövőbeli cselekedeteiket is. Bizonyos szólásaktusok információt juttatnak a hallgatóhoz, feltételezve, hogy a hallgató döntéshozatalát megfelelően befolyásolja az információ. Mások sokkal célzottabban késztetik a hallgatót valamelyen cselekvés véghezvitelére. A szólásaktusok további osztálya a **deklaratív (declarative)**, amely sokkal közvetlenebb hatást gyakorol a világra: *férjnek és feleségnek nyilvánítom öröket vagy Harmadik találat, ön kiesett*. Természetesen a hatást az érintett ágensek mentális állapotai komplex hálózatának létrehozása vagy tudomásulvétele éri el: házasnak lenni vagy kiesni, olyan állapotok, amelyeket megállapodások rögzítenek és nem a világ „fizikai” jellemzői.

A kommunikáló ágens feladata annak előtíse, hogy *mikor* van szükség egy szólásaktusra, és hogy *melyik* a helyénvaló az összes lehetséges közül. A szólásaktusok megértésének problémája hasonlatos más **megértési (understanding)** problémákhöz, mint például a képek megértése vagy a betegségek diagnosztizálása. Kapunk egy halmazt többérmű bemenetekkel, amikből visszafelé haladva azt kell eldöntenünk, hogy a világ mely állapota hozhatta létre őket. Azonban mivel a beszéd tervezett cselekvés, a megértés magában foglalja a tervfelismerést is.

## A nyelv alapjai

A **formális nyelvet (formal language)** **karakterfüzérek (strings)** (lehetséges, hogy végtelen) halmazaként definiáljuk. minden egyes füzér az úgynevezett záró (vagy terminális) **szimbólumok (terminal symbols)** – amelyeket néha szavaknak hívunk – össz-

szekapcsolt sorozata. Például az elsőrendű logikában a záró szimbólumok között van a  $\wedge$  és a  $P$ , és egy tipikus füzér a „ $P \wedge Q$ ”. A „ $P \wedge Q$ ” füzér nem része a nyelvnek. A formális nyelvek, mint például az elsőrendű logika vagy a Java, szigorú matematikai definíciókkal rendelkeznek. Ezzel ellentétben a **természetes nyelvek (natural languages)**, mint például a kínai, a dán és az angol, nem rendelkeznek szigorú definícióval, hanem beszélők egy közössége használja. Ebben a fejezetben a természetes nyelveket megpróbáljuk formális nyelveknek tekinteni, bár tisztában vagyunk azzal, hogy az illeszkedés nem lesz tökéletes.

A **nyelvtan (grammar)** a nyelvet meghatározó szabályok véges halmaza. A formális nyelveknek minden van egy hivatalos nyelvtana, amelyet kezelési útmutatókban vagy könyvekben határoznak meg. A természetes nyelveknek nincs hivatalos nyelvtna; bár a nyelvészek törekszenek a nyelv tulajdonságainak felfedezésére tudományos vizsgálatokkal, majd ezen felfedezések rögzítésére egy nyelvtanban. Eddig egyetlen nyelvész sem járt teljes sikkerrel. Figyeljük meg, hogy a nyelvész tudósok, akik a nyelvet úgy próbálják definiálni, ahogy az létezik. Vannak olyan előíró nyelvészek is, akik megpróbálják diktálni, hogy a nyelvnek milyennek *kellene lennie*. Olyan szabályokat alkotnak, mint például a „használ az ikes ragozást”, amelyeket néha megjelentetnek stílusútmutatókban, de kevés valódi hatásuk van a beszélt nyelvre.

Mind a formális, mind a természetes nyelvek jelentést, vagy **szemantikát (semantics)** rendelnek minden egyes érvényes füzérhez. Például az aritmetika nyelvén lenne egy szabályunk, hogy ha „ $X$ ” és „ $Y$ ” kifejezések, akkor „ $X + Y$ ” szintén kifejezés, amelynek a szemantikája  $X$  és  $Y$  összege. Természetes nyelvben hasonlóan fontos a füzérek **pragmatikáját (pragmatics)** is megérteni: a füzér aktuális jelentését, ahogy az elhangzik egy adott szituációban. A jelentés nemcsak a szavakban rejlik, hanem a szavak *in situ* értelmezésében.

A legtöbb nyelvtani szabály formalizmus a **kifejezsstruktúra (phrase structure)** ötletére épül – azaz arra, hogy a füzérek **kifejezéseknek (phrases)** nevezett, különböző kategóriából származó részfüzérekből állnak össze. Például „a wumpus”, „a király” és „a sarokban levő ágens”<sup>1</sup> kifejezések minden az úgynevezett **fónévi kifejezés (noun phrase)** (vagy röviden *NP*) kategória példái. Két ok létezik arra, hogy a kifejezéseket ilyen módon megkülönböztessük. Az egyik, hogy a kifejezések általában természetes szemantikus elemeknek felelnek meg, amelyekből a kijelentés jelentése megalkotható; például a fónévi kifejezések a világ objektumaira vonatkoznak. A másik, hogy a kifejezések kategorizálása segít a nyelv megengedett füzéreinek leírásában. Azt mondhatjuk, hogy a fónévi kifejezések bármelyike kombinálható egy **igei kifejezéssel (verb phrase) (VP)**, mint például „is dead” (halott), hogy közösen egy olyan kifejezést formáljanak, amelyik a **mondat (sentence) (S)** kategóriába tartozik. A fónévi és igei kifejezés köz-

<sup>1</sup> A fejezetben hozott példák fordításakor a következőképpen jártunk el: az olyan angol kifejezéseket, amelyek magyarul is szemléletesek az adott mondanivaló szempontjából, magyar fordításban adjuk meg. A kifejezések és részletesebb példák többségénél ez nem követhető, mivel a szerző az angol nyelv sajátosságaihoz illeszti a fejezet ismeretanyagának és példáinak megfogalmazását, így a fordítás lényegi változást hozott volna a fejezet tartalmában is. Ezeken a részeken a magyar változatban is meghagyjuk az eredeti angol kifejezéseket, szükség esetén rövid magyar fordításukkal kiegészítve. A szerzők létrehozzák az angol nyelv egy formális részhalmazát, amin bemutatják a fejezetben ismertetett módszerek alkalmazását. Ezt is változatlan formában helyezzük át a magyar nyelvű kiadásba. Ezen részek magyar fordítását sem készítjük el, mivel a fordítás a mondanivaló szempontjából lényegtelen. (A *ford.*)

## Generálóképesség

A nyelvtani formalizmusok osztályozhatók generálóképességgük (generative capacity) szerint: azoknak a nyelveknek a halmaza szerint, amelyeket képesek reprezentálni. Chomsky a nyelvtani formalizmusok négy osztályát írta le, amelyek csak az átiró szabályok formájában különböznek (Chomsky, 1957). Az osztályok hierarchiába rendezhetők, ahol minden osztály felhasználható az összes olyan nyelv leírására, ami leírható egy nála gyengébb osztálytal, és még néhány további nyelvre is. Itt most felsoroljuk a hierarchiát, a legerősebbel kezdve.

A rekurzíván felsorolható nyelvtanok (recursively enumerable grammars) korlátosztás nélküli szabályokat használnak: az átiró szabályok minden oldala tetszőleges számú záró és nem záró szimbólumot tartalmazhat, mint például az  $A \rightarrow C$  szabály. Ezen nyelvtanok kifejezőereje meg-egyezik a Turing-gépével.

A környezetérzékeny nyelvtanok (context-sensitive grammars) csak olyan értelemben korlátoztak, hogy a jobb oldalnak legalább annyi szimbólumot kell tartalmaznia, mint a bal oldalnak. A környezetérzékeny (környezetfüggő) elnevezés abból a tényből ered, hogy egy  $A S B \rightarrow A X B$  szabály azt mondja, hogy egy  $S$  átírható  $X$ -re egy megelőző  $A$  és egy követő  $B$  környezetben. A környezetérzékeny nyelvtanok leírhatnak például olyan nyelvet, mint az  $a^n b^n c^n$  ( $a$ -k  $n$  hosszúságú sorozata, amelyet ugyanannyi  $b$  és  $c$  követ).

A környezetfüggetlen nyelvtanok (context-free grammars) (vagy CFG-k) esetében a bal oldal egyetlen nem záró szimbólumot tartalmaz. Így minden egyes szabály megengedi a nem záró szimbólum átírását a jobb oldalra tetszőleges környezetben. A CFG-k népszerűek a természetes és programozási nyelvek nyelvtani körében, bár ma már széles körben elfogadott, hogy létezik legalább néhány olyan természetes nyelv, mely tartalmaz olyan szerkezeteket, melyek nem írhatók le környezetfüggetlen nyelvtannal (Pullum, 1991). A környezetfüggetlen nyelvtanok képesek reprezentálni  $a^n b^n$ -t, de nem  $a^n b^n c^n$ -t.

A reguláris nyelvtanok (regular grammars) a legkorlátozottabb osztály. minden szabálynak van egyetlen nem zárója a bal oldalán, és egy záró szimbólum a jobb oldalon, amit opcionálisan követhet egy nem záró. A reguláris nyelvtanok a véges automatával ekvivalens erejüek. Gyengén alkalmazás programozási nyelvek számára, például nem képesek nyitó és bezáró zárójelek egyensúlyát leíró szerkezetek ábrázolására (az  $a^n b^n$  nyelv egy változata). Ehhez legközelebb  $a^* b^*$  reprezentálásával juthatnak, amely tetszőleges számú  $a$  sorozatát követő tetszőleges számú  $b$ .

A hierarchiában fentebb helyezkedő nyelvtanoknak nagyobb a kifejezőereje, de a kapcsolódó algoritmusok kevésbé hatékonyak. Az 1980-as évek közepéig a nyelvészek a környezetfüggetlen és környezetérzékeny nyelvtanokkal foglalkoztak. Azóta nagyobb hangsúlyt fektetnek reguláris nyelvtanokra, amelyeket az elektronikusan elérhető szövegek mega- és gigabájtjainak gyors, még a kevésbé teljes analízis árán is történő feldolgozási igénye hívott életre. Ahogy Fernando Pereira mondta: „Minél öregebb leszek, annál lentebb megyek a Chomsky-hierarchián.” Hogy lássuk, mit értett ezen, vesse össze (Pereira és Warren, 1980; Mohri, Pereira és Riley, 2002)

bülső fogalma nélküli sokkal bonyolultabb lenne elmagyarázni, hogy miért jó mondat a „the wumpus is dead” és miért nem a „wumpus the dead is”.

Az  $NP$ , a  $VP$  és az  $S$  kategóriák úgynevezett nem záró szimbólumok (nonterminal symbols). A nyelvtanok átiró szabályok (rewrite rules) segítségével definiálják a nem záró szimbólumokat. Az átiró szabályok leírására a Backus–Naur-formát (BNF) fogjuk átvenni, amelyet a B) függelékben írnak le az 1112. oldalon. Ezen jelölés szerint egy

$$S \rightarrow NP \ VP$$

szabály jelentése az, hogy egy  $S$  egy tetszőleges  $NP$  kategóriájú kifejezésből és az azt követő tetszőleges  $VP$  kategóriájú kifejezésből áll.

## A kommunikációt alkotó lépések

Egy tipikus kommunikációs epizód, amikor  $S$  beszélő  $P$  állítást szeretné átadni  $H$  hallgatónak  $W$  szavakkal, hétfolyamatból áll:

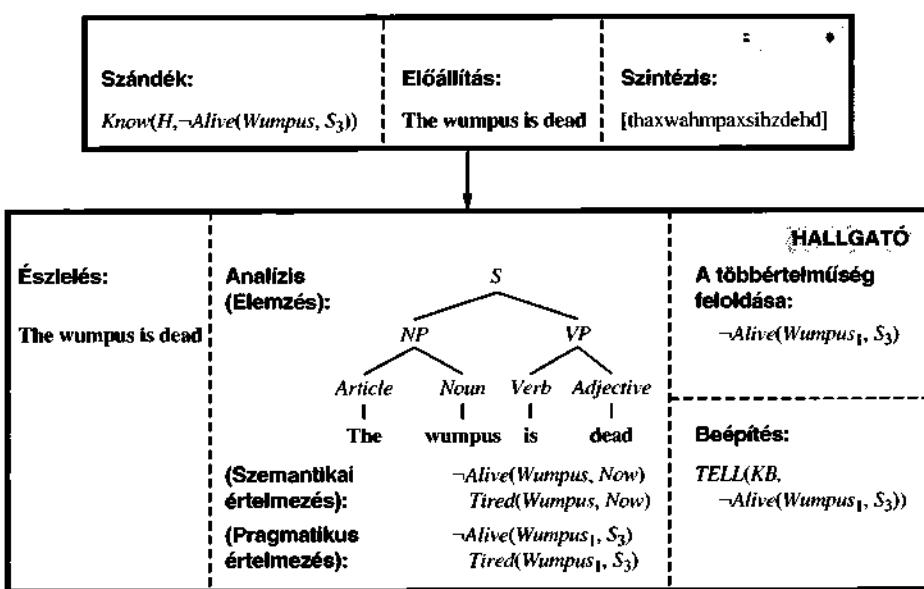
**Szándék (intention).**  $S$  beszélő valahogy eldönti, hogy van egy  $P$  propozíció, amit érdemes elmondani  $H$  hallgatónak. A példánkban a beszélőnek az a szándéka, hogy a hallgató tudomására hozza, hogy a wumpus már nem él.

**Létrehozás (generation).** A beszélő megtervezи, hogy  $P$  propozíciót hogyan alakítsa át olyan kijelentéssé, amely valószínűvé teszi, hogy a hallgató, fogadva a kijelentést a jelenlegi szituációban, kikövetkeztetheti  $P$  jelentést (vagy egy hozzá közelít). Tételezzük fel, hogy a beszélő képes a következő szavakat létrehozni: „The wumpus is dead” (A wumpus halott), és jelöljük ezt  $W$ -vel.

**Szintézis (synthesis).** A beszélő létrehozza  $W$  szavak fizikai realizációját,  $W'$ -t. Ezt teheti tintával papíron, rezgésekkel levegőben vagy valamilyen más médiumon. A 22.1. ábrán egy olyan ágenst mutatunk, amely a 661. oldalon definiált fonetikus ábécével leírt  $W'$  hangfüzér szintézisét végzi: „[thaxwahmpaxsihdehd]”. A szavak egybefolytak, ami tipikus a gyorsan beszélt nyelv esetében.

**Észlelés (perception).**  $H$  észleli a  $W'$  fizikai megvalósulást, mint  $W'_1$ -t, és  $W_2$  szavakként dekódolja. Amikor a médium a beszéd, az észlelési lépést **beszédfelismerésnek** (speech recognition) nevezzük; amikor az írás, akkor **optikai karakter felismerésének** (optical character recognition). Mindkettő elmozdult az elvont léttől a minden nap előfordulás irányába a kilencvenes években, főként az asztali számítógépek jelentős mértékben növekvő teljesítményének köszönhetően.

**Analízis (analysis).**  $H$  kikövetkezteti, hogy  $W_2$ -nek  $P_1, \dots, P_n$  lehetséges jelentése van. Az analízist három fő részre osztjuk: szintaktikai elemzésre (parsing), szemantikai



22.1. ábra. A kommunikációban részt vevő hétfolyamat, a „The wumpus is dead” mondat felhasználásával

értelmezésre és pragmatikus értelmezésre. Az **elemzés (parsing)** egy bemeneti füzérhez tartozó **elemzési**, más szóval **levezetési fa (parse tree)** építésének a folyamata, ahogy a 22.1. ábrán látható. Az elemzési fa belső csomópontjai kifejezéseket reprezentálnak, míg a levelek szavakat jelképeznek. A **szemantikus értelmezés (semantic interpretation)** az a folyamat, amelynek során kinyerjük egy kijelentés jelentéstartalmát valamilyen reprezentációs nyelven. A 22.1. ábrán két lehetséges szemantikai értelmezést mutatunk be: azt, hogy a wumpus nem él, és azt, hogy fáradt (a *dead* egy köznyelvi jelentése). A több lehetséges értelmezéssel rendelkező kijelentéseket **többértelműnek (ambiguous)** mondjuk. A **pragmatikus értelmezés (pragmatic interpretation)** figyelembe veszi azt a tényt, hogy ugyanazon szavaknak más jelentése lehet eltérő helyzetekben. Míg a szintaktikai értelmezés egy egy argumentummal, a füzérrel rendelkező függvény, addig a pragmatikus értelmezés a kijelentést és annak környezetét vagy szituációját figyelembe vevő függvény. A példában a pragmatikus elemzés két dolgot tesz: a *Now* konstans értéket kicseréli az  $S_3$  konstanssal, ami a jelenlegi szituációt jelenti, illetve a *Wumpus*-t kicseréli *Wumpus<sub>1</sub>*-re, ami arra az egyetlen Wumpusra utal, amelyről tudott, hogy a barlangban van. Általánosságban a pragmatikus elemzés a kijelentés végső értelmezéséhez sokkal többel járulhat hozzá; gondolunk csak például arra, amikor a „A gyémántra nézek” elhangzik egy ékszerész, illetve egy baseballjátékos szájából.<sup>2</sup> A 22.7. alfejezetben látni fogjuk, hogy a pragmatikus elemzés segít értelmezi az „It is dead” kijelentést úgy, hogy a wumpus halott, ha egy olyan szituációban vagyunk, amikor a wumpus áll a figyelem középpontjában.

A **többértelműség feloldása (disambiguation)**.  $H$  kikövetkezteti, hogy  $S P_i$ -t szándékozott közölni (ahol ideális esetben  $P_i = P$ ). A beszélők többsége nem szándékosan többértelmű, de a legtöbb kijelentésnek több megengedett értelmezése van. A kommunikáció azért működik, mivel a hallgató elvégzi azt a munkát, hogy rájöjjön, hogy melyik az az értelmezés, amit a beszélő valószínűleg közölni akart. Vegyük észre, hogy ez az első alkalom, amikor a *valószínűleg* szót használtuk, és a többértelműség feloldása az első eljárás, ami erőteljesen valószínűsígi következtetésen alapul. Az analízis lehetséges értelmezéseket állít elő; ha egynél több értelmezést talál, akkor a többértelműség feloldása választja ki a legjobbat.

**Beépítés (incorporation).**  $H$  eldönti, hogy elhiszi  $P_i$ -t (vagy nem). Egy teljesen naiv ágens minden elhíhet, amit hall, de egy kifinomultabb ágens úgy kezeli a szólásaktust, mint egy  $P_i$ -t alátámasztó tényt, és nem mint annak a megerősítését.

Mindezeket összerakva a 22.2. ábrán látható ágensprogramot kapjuk. Itt az ágens robotszolgaként működik, amit egy gazda irányíthat. A szolga minden lépésben megválaszolja a gazda kérdését, illetve végrehajtja a parancsát, és a szolga minden, a gazda által kijelentett állást elhisz. Ezenkívül megállapítást tesz az aktuális szituációra (csak egyszer), ha nincs más tennivalója, és megtervezи saját akcióját, ha magára hagyják. Íme, egy tipikus párbeszéd:

**ROBOTSZOLGA**

Szellőt érzek.

Semmi sincs itt.

Szellőt és bűzt érzek, valamint csillagást látok.

**GAZDA**

Menj 1, 2-re!

Menj északnak!

Ragadd meg az aranyat!

<sup>2</sup> A baseballpálya angol elnevezése *diamond*, mivel alakjával a gyémántra hasonlít. (A ford.)

```

function NATV-KOMMUNIKÁLÓ-ÁGENS(észlelés) returns cselekvés
  static: TB, egy tudásbázis
    állapot, a környezet jelenlegi állapota
    cselekvés, a legutóbbi cselekvés, kezdetben semmi

  állapot ← ÁLLAPOT-FRÍSSÍTÉS(állapot, cselekvés, észlelés)
  szavak ← BESZÉD-RÉSZ(észlelés)
  szemantika ← EGYÉRTELMŰSÍT(PRAGMATIKUS-ÉRTELMEZÉS(SZEMANTIKA(ELEMZÉS(szavak))))
  if szavak = Üres and cselekvés nem MOND then /* Írd le az állapotot! */
    return MOND(LEÍRÁS-KÉSZÍTÉS(állapot))
  else if TÍPUS[szemantika] = Utasítás then /* Hajtsd végre az utasítást! */
    return TARTALOM[szemantika]
  else if TÍPUS[szemantika] = Kérdés then /* Válaszd meg a kérdést! */
    válasz ← KÉRDEZ(TB, szemantika)
    return MOND(LEÍRÁS-KÉSZÍTÉS(válasz))
  else if TÍPUS[szemantika] = Állítás then /* Hidd el az állítást! */
    MOND(TB, TARTALOM[szemantika])
  /* Ha idáig eljutunk, akkor a „hagyományos” cselekvést hajtsd végre! */
  return ELSŐ(TERVEZÉS(TB, állapot))

```

**22.2. ábra.** Egy kommunikáló ágens, amely elfogad parancsokat, kérdéseket és állításokat. Az ágens leírhatja az aktuális állapotot, valamint végrehajthat „hagyományos”, nem szólásaktus cselekvést is, ha nincs mit mondania.

## 22.2. FORMÁLIS NYELVTAN AZ ANGOL NYELV EGY TÖREDÉKÉRE

Ebben a fejezetben definiálunk egy nyelvtant az angol nyelv egy kis töredékére, ami megfelelő a wumpus világról szóló kijelentésekhez. Ezt a nyelvet  $\mathcal{E}_0$ -nak nevezzük. A későbbi részek javítani fognak az  $\mathcal{E}_0$ -n, hogy valamelyest jobban hasonlítsan a beszélt angolra. Igen valószínűtlen, hogy valaha is megszerkesztjük az angol nyelv teljes nyelvtanát, már csak azért sem, mivel nincs két ember, aki teljesen megegyezne abban, mit is tekint érvényes angolnak.

### Az $\mathcal{E}_0$ szókincse

Először definiáljuk a szókincset (**lexicon**), azaz a megengedett szavak listáját. A szavakat a szótárasználók számára ismerős módon csoporthozjuk kategóriákba, vagy beszédrészekbe: főnevek, névmások és nevek dolgok elnevezésére, igék események jelölésére, jelzők a főnevek módosítására és határozószók az igék módosítására. Egyes olvasók számára kevessé ismertek lehetnek a következő kategóriák: **névelők** (**articles**) (például a *the*), **elöljárószavak** (**prepositions**) (például *in*) és **kötőszók** (**conjunctions**) (például *and*). A 22.3. ábra mutat egy kis szókincset.

Minden egyes kategória ...-tal végződik annak jelzésére, hogy vannak más szavak is a kategóriában. Azt is meg kell jegyeznünk azonban, hogy két különálló ok van a sza-

<i>Noun</i>	→	stench   breeze   glitter   nothing   wumpus   pit   pits   gold   east   ...
<i>Verb</i>	→	is   see   smell   shoot   feel   stinks   go   grab   carry   kill   turn   ...
<i>Adjective</i>	→	right   left   east   dead   back   smelly   ...
<i>Adverb</i>	→	here   there   nearby   ahead   right   left   east   south   back   ...
<i>Pronoun</i>	→	me   you   I   it   ...
<i>Name</i>	→	John   Mary   Boston   Aristotle   ...
<i>Article</i>	→	the   a   an   ...
<i>Preposition</i>	→	to   in   on   near   ...
<i>Conjunction</i>	→	and   or   but   ...
<i>Digit</i>	→	0   1   2   3   4   5   6   7   8   9

22.3. ábra.  $\mathcal{E}_0$  szókincse

vak hiányára. A főnevek, az igék, a jelzők és a névmások esetében elméletileg sem lehetséges az összes felsorolása. Nemcsak azért, mert minden osztálynak ezernyi vagy tízezernyi tagja lehet, hanem azért is, mivel folyamatosan hozzáadódnak újak, mint például az *MP3* vagy az *anime*. Ezt a négy kategóriát nyitott osztályoknak (*open classes*) nevezzük. A többi kategóriát (névmások, névelők, elöljáró- és kötőszók) **lezárt osztályoknak** (*closed classes*) hívjuk. Mindegyikben kevés számú szó van (néhánytól pártucatig), amelyek elméletileg teljesen felsorolhatók. A lezárt osztályok évszázadok folyamán változnak, nem hónapok alatt. Például a „thee” és a „thou” a 17. században elterjedt névmások voltak, hanyatlóban voltak a 19. században, és manapság csak versekben és egyes régiók dialektusában lehet találkozni velük.

## Az $\mathcal{E}_0$ nyelvtana

A következő lépés a szavak kifejezésekkel történő kombinálása. Őt nem záró szimbólumot fogunk használni a különböző típusú kifejezések definiálására: mondat (*S*), főnévi kifejezés (*NP*), igei kifejezés (*VP*), elöljárói kifejezés (*PP*) és relativ klóz (*RelClause*).<sup>3</sup> A 22.4. ábra  $\mathcal{E}_0$  számára mutat egy nyelvtant, minden egyes átíró szabályra példát adva. Az  $\mathcal{E}_0$ jó angol mondatokat generált, mint amilyenek például a következők:

John is in the pit

The wumpus that stinks is in 2 2

Mary is in Boston and John stinks

Sajnálatos módon a nyelvtan **túlgenerál** (*overgenerates*) – előállít olyan mondatokat, amelyek nyelvtanilag helytelenek –, mint például a „Me go Boston” és az „I smell pit

<sup>3</sup> Egy relativ klóz főnévi kifejezést követ és módosít. (A klóz a nyelvzetben egy alárendelt mellékmondat – a *ford.*) Egy relativ névmásból (mint a „who” és a „that”) és egy azt követő igei kifejezésből áll. (Egy másik típusú relativ klóz a 22.12. feladatban szerepel.) Példa egy relativ klózra a *that stinks* a „The wumpus that stinks is in 2 2” mondatban.

<i>S</i>	$\rightarrow$	<i>NP VP</i> <i>S Conjunction S</i>	I + feel a breeze I feel a breeze + and + I smell a wumpus
<i>NP</i>	$\rightarrow$	<i>Pronoun</i>	I
	$\rightarrow$	<i>Name</i>	John
	$\rightarrow$	<i>Noun</i>	pits
	$\rightarrow$	<i>Article Noun</i>	the + wumpus
	$\rightarrow$	<i>Digit Digit</i>	3 4
	$\rightarrow$	<i>NP PP</i>	the wumpus + to the east
	$\rightarrow$	<i>NP RelClause</i>	the wumpus + that is smelly
<i>VP</i>	$\rightarrow$	<i>Verb</i>	stinks
	$\rightarrow$	<i>VP NP</i>	feel + a breeze
	$\rightarrow$	<i>VP Adjective</i>	is + smelly
	$\rightarrow$	<i>VP PP</i>	turn + to the east
	$\rightarrow$	<i>VP Adverb</i>	go + ahead
<i>PP</i> <i>RelClause</i>	$\rightarrow$	<i>Preposition NP</i>	to + the east
	$\rightarrow$	<i>that VP</i>	that + is smelly

22.4. ábra. Az  $\mathcal{E}_0$  nyelvtana, minden egyes szabályra példa kifejezést adva

gold wumpus nothing east". A hűl is generál (**undergenerates**): sok olyan angol mondat van, amit elutasít, mint például a „I think the wumpus is smelly”. (Egy másik hátránya, hogy a nyelvtan nem nagybetűvel kezdi a mondatokat, és nem rak pontot a végükre. Ez azért van, mivel elsősorban beszédre terveztek, és nem írásra.)

## 22.3. SZINTAKTIKAI ANALÍZIS (ELEMZÉS)

Már definiáltuk az elemzést (parsing) mint az adott bemeneti füzérhez tartozó levezelesi fa megtalálásának folyamatát. Azaz a PARSE függvény meghívása, mint például a

PARSE(„the wumpus is dead”,  $\mathcal{E}_0$ , *S*)

olyan fát kell visszaadnia, amelynek gyökerében *S* áll, levelei a „the wumpus is dead” és belső csomópontjai az  $\mathcal{E}_0$  nyelvtan nem záró szimbólumai. A 22.1. ábrán láthattunk egy ilyen fát. Folytonos szövegként a következőképpen írhatjuk:

[*S*: [*NP*: [*Article*: **the**] [*Noun*: **wumpus**]]  
[*VP*: [*Verb*: **is**][*Adjective*: **dead**]]]

Az elemzés tekinthető egy levezetési fa megkeresésének folyamataként. A keresési tér meghatározásának két szélsőséges (és sok közülös) módja van. Az egyik szerint kiindulhatunk az *S* szimbólumból, és kereshetünk egy olyan fát, amely leveleiben tartalmazza a szavakat. Ez az úgynevezett **fentről lefelé elemzés (top-down parsing)** (mivel *S*-et a fa tetejére helyezzük). Másrészt kiindulhatunk a szavakból, és kereshetünk egy fát, ahol *S* a gyökércsomópont. Ez a **lentről felfelé elemzés (bottom-up)**



parsing).<sup>4</sup> A fentről lefelé történő elemzés pontosan definiálható a következő keresési problémaként:

- A **kezdeti állapot (initial state)** egy elemzési fa, amelynek  $S$  a gyökérkocsomópontja, és egy ismeretlen gyerekcsomópontja van:  $[S: ?]$ . Általánosságban a keresési téren minden állapot egy levezetési fa.
- Az **állapotátmenet-függvény (successor function)** kiválasztja azt a legszélső csomópontot bal oldalon a fában, amelynek ismeretlen gyereke van. Ezek után a nyelvtanban olyan szabályokat keres, amelyek ezt a csomópontot tartalmazzák gyökérelemként. minden ilyen szabályra generál egy következő állapotot, ahol a  $??$  szimbólumot felcseréli a szabály jobb oldalának megfelelő listával. Például az  $\mathcal{E}_0$  nyelvtanban két szabály van  $S$ -re, így az  $[S: ?]$  fát a következő két származtatottal cseréli le:

$[S: [S: ?] [Conjunction: ?] [S: ?]]$

$[S: [NP: ?] [VP: ?]]$

A másodiknak hét származtatottja lesz, minden  $NP$  átírási szabályhoz egy.

- A **célteszt (goal test)** ellenőrzi, hogy a levelezési fa levelei pontosan megfelelnek-e a bemeneti füzérnek, nincsenek-e ismeretlenek és lefedetlen bemenetek.

A fentről lefelé történő elemzés egyik nagy problémája az úgynevezett **bal-rekurzív szabályok (left-recursive rules)** kezelése, melyek  $X \rightarrow X \dots$  alakúak. Mélységi keresséssel egy ilyen szabály végtelen ciklusban  $X$ -et  $[X: X \dots]$ -ra cserélne. Szélességi keressés esetén sikeresen megtaláljuk az érvényes mondatok elemzéseit, de érvénytelen mondatok esetében egy végtelen keresési téren ragadnánk le.

A lentről felfelé történő elemzés keresésként történő formalizálása a következő:

- A **kezdeti állapot (initial state)** a bemeneti füzérben található szavak listája, minden egyiket egy olyan levelezési faként ábrázolva, melynek csak egy levele van, például: **[the, wumpus, is, dead]**. Általánosságban a keresés minden állapota levelezési fák egy listája.
- Az **állapotátmenet-függvény (successor function)** megvizsgál minden  $i$  pozíciót a fák listájában a nyelvtan szabályainak minden lehetséges jobb oldalán. Ha a lista  $i$  pozíciójában kezdődő részsorozata illeszkedik a jobb oldalra, akkor a részsorozatot lecseréli egy új fára, amelynek kategóriája a szabály bal oldala, és amelynek gyerekei a részsorozat. „Illeszkedés” alatt azt értjük, hogy a csomópont kategóriája meggyezik a jobb oldal elemének kategóriájával. Például az **Article**  $\rightarrow$  **the** szabály illeszkedik a **[the, wumpus, is, dead]** első csomópontjából álló részsorozatra, így a következő állapotot az **[[Article: the], wumpus, is, dead]** lenne.
- A **célteszt (goal test)** egy olyan állapotot keres, ahol egyetlen fa van, melynek gyötere az  $S$ .

A 22.5. ábra bemutat egy példát a lentről felfelé történő elemzésre.

A lentről felfelé és a fentről lefelé történő elemzés is lehet kevésbé hatékony azon módonk számossága miatt, ahogy különböző kifejezésekhez többféle elemzés rendelhető.

<sup>4</sup> Eszrevehető, hogy a fentről lefelé és a lentről felfelé történő elemzés hasonló az előre-, illetve hátrafélé következtetéshez, amiket a 7. fejezetben írtunk le. Hamarosan látni fogjuk, hogy ez az analógia pontos.

lépés	csomópontok listája	részszorozat	szabály
KEZD	<b>the wumpus is dead</b>	<b>the</b>	Article → <b>the</b>
2	<i>Article</i> <b>wumpus is dead</b>	<b>wumpus</b>	Noun → <b>wumpus</b>
3	<i>Article Noun</i> <b>is dead</b>	<i>Article Noun</i>	<i>NP</i> → <i>Article Noun</i>
4	<i>NP</i> <b>is dead</b>	<b>is</b>	<i>Verb</i> → <b>is</b>
5	<i>NP Verb</i> <b>dead</b>	<b>dead</b>	<i>Adjective</i> → <b>dead</b>
6	<i>NP Verb Adjective</i>	<i>Verb</i>	<i>VP</i> → <i>Verb</i>
7	<i>NP VP</i> <b>Adjective</b>	<i>VP Adjective</i>	<i>VP</i> → <i>VP Adjective</i>
8	<i>NP VP</i>	<i>NP VP</i>	<i>S</i> → <i>NP VP</i>
CÉL	<b>S</b>		

**22.5. ábra.** A „the wumpus is dead” lentről felfelé történő elemzési lépései. A szavakat tartalmazó csomópontok listájával kezdünk. Ezek után egy szabály jobb oldalára illeszkedő részsorozatokat felcserélünk egy új csomóponttal, amelynek gyökere a szabály bal oldala. Például a harmadik sorban az *Article* és *Noun* csomópontokat lecseréljük egy *NP* csomópontra, amelynek e két csomópont lesz a gyereke. A fentről lefelé elemzés hasonló lépéseket eredményezne, de épp a fordított irányban.

Mindkettő elpazarolhatja az időt a keresési tér irreleváns részeinek vizsgálatával. A fentről lefelé keresés generálhat olyan közbülső csomópontokat, amelyek sosem zárhatók le szavakkal, és a lentről felfelé keresés generálhat olyan részleges elemzéseket a szavakhoz, amelyek nem fordulhatnak elő egy *S*-ben sem.

Még ha lenne is egy tökéletes heurisztikánk, amely lehetővé tenné az irreleváns kiterők nélküli keresést, ezek az algoritmusok akkor sem lennének hatékonyak, mivel bizonyos mondatoknak *exponenciálisan sok* levezetési fája van. A következő alfejezet meghatározza, hogy mit kezdhetünk ezzel a problémával.

## Hatókony elemzés

Vizsgáljuk meg a következő két mondatot:

Have the students in section 2 of Computer Science 101 take the exam.

Have the students in section 2 of Computer Science 101 taken the exam?

Bár az első 10 szavuk közös, teljesen különböző levezetéstük van, mivel az első egy felszólítás, a második egy kérdés. Egy balról jobbra elemző algoritmusnak tippelnie kellene, hogy az első szó egy felszólítás vagy egy kérdés része-e, és nem tudná megmondani, hogy a tipp helyes-e egészen a tizenegyedik szóig: *take* vagy *taken*. Ha az algoritmus rosszul tippelt, egészen az első szóig kellene visszalépnie. Az ilyen típusú visszalépés elkerülhetetlen, de ha azt szeretnénk, hogy az algoritmusunk hatékony legyen, akkor el kell kerülnie a „the students in section 2 of Computer Science 101” *NP*-ként történő újraelemzését minden alkalommal, amikor visszalép.

Ebben a részben egy olyan algoritmust alakítunk ki, amely ezt a hatékonyiségi problémát kezelni tudja. Az alapötlet a **dinamikus programozás (dynamic programming)** egy példája: *minden alkalommal, amikor egy részfüzérét elemzünk, tárolj az eredményt, így később majd nem kell újraelemzünk*. Például ha egyszer rájöttünk, hogy a „the students in section 2 of Computer Science 101” egy *NP*, ezt eltárolhatjuk egy **diagramnak (chart)** nevezett adatstruktúrában. Az így működő algoritmusokat **diagramelemzőknek**



(chart parsers) nevezik. Mivel környezetfüggetlen nyelvtanokkal foglalkozunk, a keresési tér egy ágának kontextusában talált tetszőleges kifejezés éppúgy szerepelhet a keresési tér bármilyen más ágában is.

Egy  $n$  szóból álló mondat diagramja  $n + 1$  csomópontból (vertex) és számos, ezeket összekötő élből (edges) áll. A 22.6. ábrán láthatunk egy diagramot hat csomóponttal (körök) és három éllel (vonalak). Például a

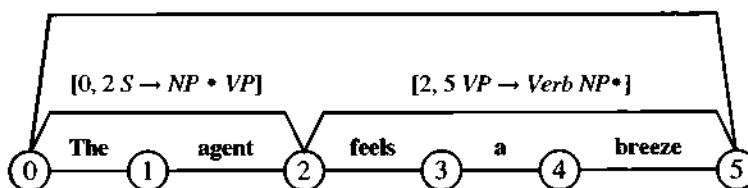
$[0, 5, S \rightarrow NP VP^{\bullet}]$

címkéjű él azt jelenti, hogy egy  $NP$ -t egy  $VP$  követ, és együtt alkotnak egy  $S$ -et, mely a 0-tól 5-ig terjedő karaktersorozatot fedи le. Egy élben található  $\bullet$  jel elválasztja a már megtaláltat a még keresettől.<sup>5</sup> A  $\bullet$  jellel záródó éleket teljes éleknek (complete edges) nevezik. A

$[0, 2, S \rightarrow NP \bullet VP]$

él szerint egy  $NP$  lefedi a 0-tól 2-ig terjedő karaktersorozatot (az első két szó), és ha tudunk találni egy ezt követő  $VP$ -t, akkor lenne egy  $S$ -ünk. Az ilyen élek, ahol a pont a végük előtt van, úgynévezett befejezetlen (nem teljes) élek, és azt mondjuk, hogy az él egy  $VP$ -t vár.

$[0, 5 S \rightarrow NP VP^{\bullet}]$



22.6. ábra. A „The agent feels a breeze” mondat diagramjának egy részlete. Mind a hat csomópont lát-ható, de csak három él szerepel azok közül, melyek teljes elemzést eredményeznének.

A 22.7. ábra a diagramelemző algoritmust szemlélteti. A lényege a fentről lefelé és a lentről felfelé legjobb tulajdonságainak ötvözése. A JÓSLÓ eljárás fentről lefelé működik: olyan bejegyzéseket helyez el a diagramban, amelyek megmondják, hogy milyen szimbólumok milyen helyszínen elvártak. A SZKENNER egy lentről felfelé eljárás, amely a szavakból indul ki, de egy szót csak egy meglevő diagrambejegyzés kiegészítésére használ fel. Hasonlóképpen a KITERJESZTŐ a komponenseket lentről felfelé építi, de csak egy meglevő diagrambejegyzést egészít ki.

Egy trükköt alkalmaztunk a teljes algoritmus elindításához: hozzáadtunk egy  $[0, 0, S' \rightarrow \bullet S]$  élet a diagramhoz, ahol  $S$  a nyelvtan kezdeti szimbóluma,  $S'$  pedig egy általunk most kitalált új szimbólum. Az ÉL-HOZZÁAD meghívása azt eredményezi, hogy a JÓSLÓ éleket ad az olyan szabályokhoz, melyek  $S$ -t eredményezhetnek, azaz  $[S \rightarrow NP VP]$ . Ezek után megvizsgáljuk ezen szabály első alkotóelemét,  $NP$ -t, és mindenféle módon olyan szabályokat adunk hozzá, melyek  $NP$ -t eredményeznek. Végső soron a JÓSLÓ fentről lefelé minden lehetéges élт hozzáad, ami felhasználható a végső  $S$  megalkotásában.

<sup>5</sup> A  $\bullet$  miatt nevezik az éleket néha pontozott szabályoknak (dotted rules).

```

function DIAGRAMELEMZŐ(szavak, nyelvtan) returns diagram

diagram  $\leftarrow$  a [0... Hossz(szavak)] üres listák elrendezése
ÉL-HOZZÁAD([0, 0,  $S' \rightarrow \bullet S$ ])
for i  $\leftarrow$  from 0 to Hossz(szavak) do
    SZKENNER(i, szavak[i])
return diagram

procedure ÉL-HOZZÁAD(él)
    /* Adj hozzá egy élét a diagramhoz, és vizsgáld meg, hogy kiterjeszt, vagy jósol-e más élt! */
    if él nincs a diagram[VÉGE(él)]-ben then
        hozzáfűz él a diagram[VÉGE(él)]-hez
        if él-nek nincs része a pont után then KITERJESZTŐ(él)
        else JÓSLÓ(él)

procedure SZKENNER(j, szó)
    /* minden egyes élterjessz ki, amely a megadott kategóriájú szót várja! */
    for each [i, j,  $A \rightarrow \alpha \bullet B \beta$ ] in diagram[j] do
        if szó B kategórájából való then
            ÉL-HOZZÁAD([i, j + 1,  $A \rightarrow \alpha B \bullet \beta$ ])

procedure JÓSLÓ([i, j,  $A \rightarrow \alpha \bullet B \beta$ ])
    /* minden olyan B-re vonatkozó szabályt adj a diagramhoz, ami segíthet ezen élt kiterjeszteni! */
    for each (B  $\rightarrow \gamma$ ) in ÁTIR(B, nyelvtan) do
        ÉL-HOZZÁAD([j, j,  $B \rightarrow \bullet \gamma$ ])

procedure KITERJESZTŐ([j, k,  $B \rightarrow \gamma \bullet$ ])
    /* Nézd meg, mely élek terjeszthetők ki ezzel az éellel! */
     $e_B \leftarrow$  ezen eljárás bemeneteként kapott él
    for each [i, j,  $A \rightarrow \alpha \bullet B' \beta$ ] in diagram[j] do
        if B = B' then
            ÉL-HOZZÁAD([i, k,  $A \rightarrow \alpha e_B \bullet \beta$ ])

```

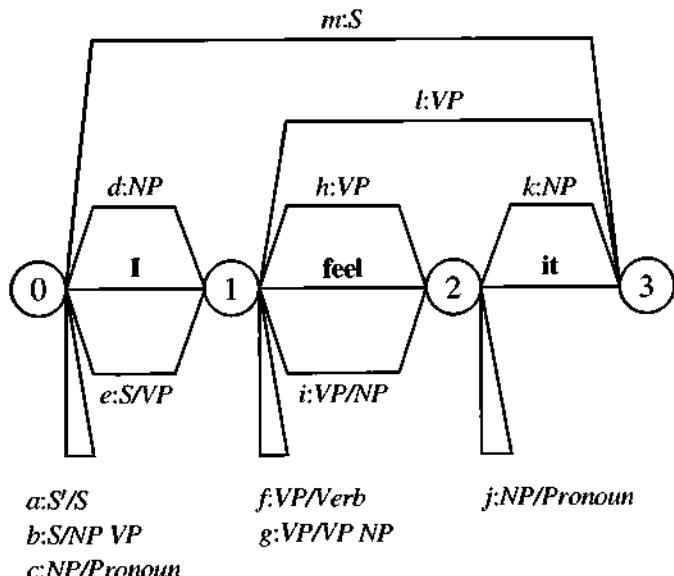
**22.7. ábra.** A diagramelemző algoritmus. *S* a kezdő szimbólum és *S'* egy új nem záró szimbólum, *diagram*[*j*] pedig azon élek listája, melyek a *j* csomópontban végződnek. A görög ábécé betűi nulla vagy több szimbólumból álló füzérekre illeszkednek.

Amikor *S'* jósólja készen van, beléünk egy ciklusba, amely meghívja a **SZKENNER**-t a mondat minden egyes szavára. Ha a *j* pozícióban álló szó egy *B* kategória tagja, amit valamelyik él keres a *j* pozícióban, akkor kiterjesztjük azt az élet megjelölve a szót, mint *B* egy példányát. Vegyük észre, hogy a **SZKENNER** minden egyes meghívása végződhet a **JÓSLÓ** és a **KITERJESZTŐ** rekurzív meghívásával, íly módon ötvözve a fentről lefelé és a lentről felfelé feldolgozást.

A másik lentről felfelé komponens, a **KITERJESZTŐ**<sup>6</sup> vesz egy teljes élt, amelynek bal oldalán *B* van, és felhasználja a diagramban levő bármely nem teljes szabály kiterjesztésére, amely ott végződik, ahol a teljes él kezdődik, ha a nem teljes szabály egy *B*-re vár.

<sup>6</sup> A **KITERJESZTŐ** eljárásunkat tradicionálisan TELJESSÉ-TEVŐ-nek hívták. Ez a név félrevezető, mivel az eljárás nem fejez be éleket: bemenetként vesz egy teljes élt, és kiterjeszt nem teljes éleket.

A 22.8. és 22.9. ábrák mutatják az „I feel it” mondat (amely a „Do you feel a breeze?” kérdésre adott válasz) diagramját és elemzési lépései. Tizenhárom él (a–m jelekkel ellátva) szerepel a diagramon, melyek közül öt teljes (a diagram vertexei csomópontjai felett) és nyolc befejezetlen (alattuk). Vegyük észre a JÓSLÓ, SZKENNER és KITERJESZTŐ akciót ciklusát. Például a JÓSLÓ felhasználja azt a tényt, hogy az „a” él egy S-t vár ahhoz, hogy megelőlegezze egy NP („b” él) és egy Pronoun („c” él) jósítatát. Ezek után a SZKENNER felismeri, hogy van egy Pronoun a megfelelő helyen („d” él), és a KITERJESZTŐ kombinálja a „b” nem teljes élt a „d” teljes éllel, így előállítva egy új élt, „e”-t.

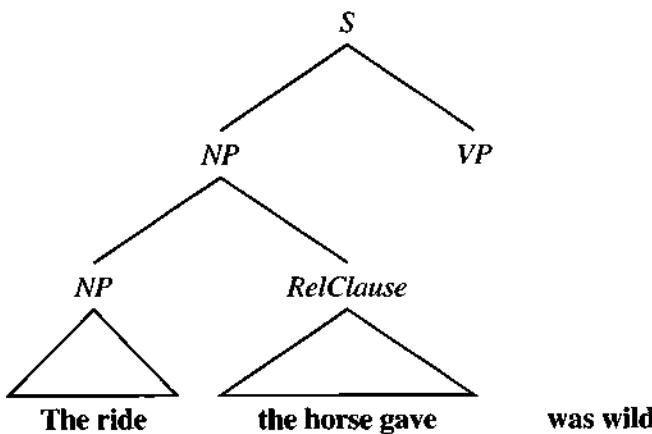


**22.8. ábra.** Az „<sub>0</sub> I <sub>1</sub> feel <sub>2</sub> it <sub>3</sub>” mondat diagramelemzése. Az *m:S* jelölés azt jelenti, hogy az *m* élnek egy *S* áll a jobb oldalán, míg az *f:VP/Verb* azt, hogy az *f* élnek *VP* van a bal oldalán, de egy *Verb*-et vár. Öt teljes él van a csomópontok felett, és nyolc hiányos alattuk.

A diagramelemző algoritmus elkerüli az élek nagy halmazának építését, amit az egyszerű lentről felfelé eljárás véggvizsgált volna. Vegyük a „The ride the horse gave was wild” mondatot. Egy lentről felfelé elemzés megjelölné a „ride the horse”-t mint egy *VP*-t, majd elvetné az elemzési fát, amint rájönne, hogy nem illeszkedik egy nagyobb *S*-be. Azonban az  $\mathcal{E}_0$  nem engedi meg, hogy egy *VP* a „the”-t kövesse, így a diagramelemző algoritmus sosem jósol *VP*-t ezen a ponton, így elkerüli ott a *VP* szerkezet időpocsékoló építését. Azokat az algoritmusokat, melyek balról jobbra működnek, és elkerülik ezen lehetetlen szerkezetek felépítését, **balsarok-elemzőknek (left-corner parsers)** nevezzük, mivel egy olyan elemzési fát építenek fel, amely a nyelvtan kezdő szimbólumával indul, és a mondat legszélén balra (a bal sarokban) álló szó felé terjeszkednek. Egy élt csak akkor adnak a diagramhoz, ha ezen elemzési fa kiterjesztését szolgálhatja (példaként lásd 22.10. ábra).

Él	Eljárás	Származtatás
a	INICIALIZÁLÁS	[0, 0, $S' \rightarrow \bullet S$ ]
b	JÓSLÓ(a)	[0, 0, $S \rightarrow \bullet NP VP$ ]
c	JÓSLÓ(b)	[0, 0, $NP \rightarrow \bullet Pronoun$ ]
d	SZKENNER(c)	[0, 1, $NP \rightarrow Pronoun \bullet$ ]
e	KITERJESZTŐ(b, d)	[0, 1, $S \rightarrow NP \bullet VP$ ]
f	JÓSLÓ(e)	[1, 1, $VP \rightarrow \bullet Verb$ ]
g	JÓSLÓ(e)	[1, 1, $VP \rightarrow \bullet VP NP$ ]
h	SZKENNER(f)	[1, 2, $VP \rightarrow Verb \bullet$ ]
i	KITERJESZTŐ(g, h)	[1, 2, $VP \rightarrow VP \bullet NP$ ]
j	JÓSLÓ(g)	[2, 2, $NP \rightarrow \bullet Pronoun$ ]
k	SZKENNER(h)	[2, 3, $NP \rightarrow Pronoun \bullet$ ]
l	KITERJESZTŐ(i, k)	[1, 3, $VP \rightarrow VP NP \bullet$ ]
m	KITERJESZTŐ(e, l)	[0, 3, $S \rightarrow NP VP \bullet$ ]

22.9. ábra. Az „ $I_0 I_1 feel I_2 it I_3$ ” elemzési lépései. Az a–m élek mindegyikére megmutatjuk azt az eljárást, amit az él más, a diagramon már szereplő élekből történő származtatására használtunk. Bizonyos éleket a rövidség érdekében elhagytunk.



22.10. ábra. Egy balsarok-elemző algoritmustal elérhető a „ride” szóval kezdődő VP jóslása, de jóslni fog egy „was” szóval kezdődő VP-t, mivel a nyelvtan egy NP-t követő VP-t vár. A „the horse gave” felettes háromszög azt jelenti, hogy a szavaknak van egy RelCause elemzésük, amihez azonban közvetlenül további részek kapcsolódnak, amik nem láthatók.

A diagramelemző csak polinomiális idő- és tárigényű.  $O(kn^2)$  helyet igényel az élek tárolására, ahol  $n$  a mondatban levő szavak száma és  $k$  egy, a nyelvtantól függő konstans. Amikor már nem tud több élt építeni, megáll, így tudjuk, hogy az algoritmus befejeződött (még akkor is, ha vannak bal-rekurzív szabályok). Valójában  $O(n^3)$  időt igényel legrosszabb esetben, ami a legjobb, amit elérhetünk környezetfüggetlen nyelvtanokra. A DIAGRAUMELEMZŐ szűk keresztmetszete a KITERJESZTŐ, amelynek meg kell próbálnia kiterjeszteni a  $j$ -ben végződő  $O(n)$  nem teljes él mindegyikét az ugyanott kezdődő  $O(n)$  teljes él mindegyikével,  $j$  lehetséges  $n + 1$  különböző értékére. Ezeket összeszorozva

kapjuk  $O(n^3)$ -at. Ez egyfajta paradoxont ad a számunkra: hogyan tud egy  $O(n^3)$  algoritmus visszaadni egy választ, amely exponenciális számosságú elemzési fát tartalmazhat? Vegyük egy példát: a

„Fall leaves fall and spring leaves spring”

mondat többértelmű, mivel minden egyes szó (az „and” kivételével) lehet főnév és ige is, valamint a „fall” és a „spring” lehet melléknév is. Ennek a mondatnak minden összesen négy elemzése van:<sup>7</sup>

[S: [S: [NP: Fall leaves] fall] and [S: [NP: spring leaves] spring]  
 [S: [S: [NP: Fall leaves] fall] and [S: spring [VP: leaves spring]]]  
 [S: [S: Fall [VP: Fall leaves] fall] and [S: [NP: spring leaves] spring]  
 [S: [S: Fall [VP: Fall leaves] fall] and [S: spring [VP: leaves spring]]]

Ha  $n$  többértelmű összekapcsolt részmondatunk lett volna, akkor  $2^n$  módon választhatnánk elemzést a részmondatokra.<sup>8</sup> Hogyan kerüli el a diagramelemző az exponenciális feldolgozási időt? Két választ is adhatunk a kérdésre. Először is, a DIAGRAMELEMZŐ algoritmus maga valójában egy *felismerő*, nem egy elemző. Ha van egy  $[0, n, S \rightarrow \alpha \bullet]$  alakú teljes él a diagramon, akkor felismertünk egy  $S$ -t. Az elemzési fa előállítását ebből az élből nem tekintjük a DIAGRAMELEMZŐ feladatának, de elvégezhető. Vegyük észre, hogy a KIBŐVÍTŐ utolsó sorában  $\alpha$ -t élek  $e_B$  listájaként építjük fel, nemcsak kategória nevek listájaként. Így egy él elemzési fává történő átalakításához egyszerűen az összetevő éleket kell rekurzív módon végignézni, minden egyes  $[i, j, X \rightarrow \alpha \bullet]$  él törökítendő. Ez egyenes/közvetlen módszer, de csak *egy* elemzési fát ad.

A második válasz szerint ha minden lehetséges elemzést szeretnénk, akkor mélyebbre kell ásnunk a diagramban. Miközben az  $[i, j, X \rightarrow \alpha \bullet]$  él törökítendő, azt is megvizsgáljuk, hogy van-e másik,  $[i, j, X \rightarrow \beta \bullet]$  alakú él. Amennyiben van, ezek az élek további elemzési fákat generálnak. Így kapjuk azt a választási lehetőséget, hogy mit is kezdjünk velük. Felsorolhatjuk az összes lehetőséget, ami azt jelenti, hogy a paradoxont feloldanánk, és exponenciálisan sok időre lenne szükségünk az összes elemzés felsorolásához. Vagy továbbvihetjük a rejteltyt egy kicsit és az elemzéseket egy **tömörített erdő** (*packed forest*) nevű struktúrával reprezentálhatjuk, amely a következőképpen néz ki:

[S: {S: {[NP: Fall leaves] [VP: fall]} } és [S: {[NP: spring leaves [VP: spring]} ] } ]

A lényeg az, hogy minden csomópont lehet egy hagyományos elemzési fa csomópont, illetve csomópontok egy halmaza is. Ez lehetővé teszi számunkra, hogy exponenciális számú elemzést reprezentáljunk polinomiális időben és tárhelyen. Természetesen  $n = 2$  esetén nincs sok különbség  $2^n$  és  $2n$  között, de nagy  $n$ -ekre egy ilyen reprezentáció jelentős megtakarítást eredményez. Sajnálatos módon ez az egyszerű tömörített erdő megközelítés nem kezeli az összes  $O(n!)$  lehetőséget a kapcsolódások összerendelésére. Maxwell és Kaplan (1995) megmutatja, hogy egy, az igazság-karbantartó rendszerek alapelveire épülő, összetettebb reprezentáció ezeket a fákat még jobban összetömörítheti.

<sup>7</sup> Az [S: Fall [VP: leaves fall]] elemzés megegyezik az „Autumn abandons autumn” mondattal.

<sup>8</sup> Emellett a komponensek összekapcsolódásának  $O(n!)$  számú értelmezése lenne – például ( $X$  és  $Y$  és  $Z$ ), illetve ( $(X$  és  $Y$ ) és  $Z$ ). De ez már egy másik történet, amelyet Church és Patil (1982) igen jól mesél el.

## 22.4. KITERJESZTETT NYELVTANOK

A 22.2. alfejezetben láttuk, hogy az  $\mathcal{E}_0$  egyszerű nyelvtana előállítja az angol nyelv „I smell a stench” és sok más mondatát. Sajnos azonban sok nem megengedett mondatot is előállít, mint például a „Me smell a stench”. Ezen probléma elkerülése érdekében a nyelvtanunknak tudnia kellene, hogy a „me” nem érvényes *NP* akkor, amikor a mondat alanya. A nyelvészek azt mondják, hogy az „I” névmás alanyesetű, míg a „me” tárgyesetű.<sup>9</sup> Ha figyelembe vesszük az eseteket, akkor rájövünk, hogy az  $\mathcal{E}_0$  nem környezetfüggetlen: nem igaz az, hogy minden *NP* minden mással egyenértékű, függetlenül a környezetétől. A probléma megoldására bevezethetünk olyan új kategóriákat, mint az  $NP_S$  és  $NP_O$  a tárgyas és alanyi főnévi kifejezésekre. Ezenkívül kellé kellene bontanunk a *Pronoun* kategóriát a  $Pronoun_S$ -re (amelyik tartalmazza az „I”-t) és  $Pronoun_O$ -ra (amelyik tartalmazza a „me”-t). A 22.11. ábra felső része meghatározza az esetegyzetet teljes BNF nyelvtanát; az így kapott nyelvet  $\mathcal{E}_1$ -nek nevezzük. Vegyük észre, hogy az összes *NP* szabályt meg kell kettőznünk, egyszer az  $NP_S$ , mászzor az  $NP_O$  esetre.

Sajnos az  $\mathcal{E}_1$  még mindig túlgenerál. Az angol és sok más nyelv megköveteli az alany és a mondat fő igéjének **egyeztetését (agreement)**. Például ha „I” az alany, akkor az „I smell” nyelvtanilag helyes, míg az „I smells” nem. Ha „it” az alany, akkor fordítva igaz. Az angolban az egyeztetésből származó különbségek minimálisak: a legtöbb igének egy alakja van egyes szám harmadik személyű alanyokra (he, she vagy it), és egy másik alakja az egyes és többes szám és a személyek minden más kombinációjára. Egy kivétel van: az „I am / you are / he is” három formával rendelkezik. Ha kombináljuk ezt a három

$S$	$\rightarrow$	$NP_S \ VP \mid \dots$
$NP_S$	$\rightarrow$	$Pronoun_S \mid Name \mid Noun \dots$
$NP_O$	$\rightarrow$	$Pronoun_O \mid Name \mid Noun \dots$
$VP$	$\rightarrow$	$VP \ NP_O \mid \dots$
$PP$	$\rightarrow$	$Preposition \ NP_O$
$Pronoun_S$	$\rightarrow$	$I \mid you \mid he \mid she \mid it \mid \dots$
$Pronoun_O$	$\rightarrow$	$me \mid you \mid him \mid her \mid it \mid \dots$
$S$	$\rightarrow$	$NP(\text{Subjective}) \ VP \mid \dots$
$NP(\text{eset})$	$\rightarrow$	$Pronoun(\text{eset}) \mid Name \mid Noun \dots$
$VP$	$\rightarrow$	$VP \ NP(\text{Objective}) \mid \dots$
$PP$	$\rightarrow$	$Preposition \ NP(\text{Objective})$
$Pronoun(\text{Subjective})$	$\rightarrow$	$I \mid you \mid he \mid she \mid it \mid \dots$
$Pronoun(\text{Objective})$	$\rightarrow$	$me \mid you \mid him \mid her \mid it \mid \dots$

**22.11. ábra.** Felül: Az  $\mathcal{E}_1$  egy BNF nyelvtana, amely kezeli az alanyi és a tárgyas esetet a főnévi kifejezésekben, és ezért annyira nem is generál túl. Az  $\mathcal{E}_1$  nyelvtanával megegyező részeket elhagytuk. Lent: Az  $\mathcal{E}_1$  egy definit klóz nyelvtana (DCG).

<sup>9</sup> Az angolban az alanyesetre a „subjective case” mellett alkalmanként a „nominative case” kifejezést is használják. Hasonlóképpen a tárgyesetre az „objective case” mellett az „accusative case” is használatos esetenként. Sok nyelvnek van részes esete is az indirekt tárgy pozícióban álló szavakra.

különbségtételt az  $NP_S$  és  $NP_O$  különbségtétellel, akkor  $NP$  hat alakját kapjuk. Ahogy további különbözőségeket fedezünk fel, végül egy exponenciális sokaságot kapunk.

Az alternatív megközelítés a nyelvtan létező szabályainak **kiterjesztése (augmenting)** új szabályok bevezetése helyett. Először megmutatjuk egy példán keresztül, hogy hogyan szeretnénk, hogy a kiterjesztett szabályok kinézzenek (a 22.11. ábra alsó fele), azután formálisan megadjuk, hogyan kell ilyen szabályokat értelmezni. A kiterjesztett szabályok nem záró kategóriák esetében megengedik paraméterek alkalmazását. A 22.11. ábra megmutatja, hogyan írhatjuk le az  $\mathcal{E}_1$  nyelvtant kiterjesztett szabályokkal. Az  $NP$  és *Pronoun* kategóriák rendelkeznek egy olyan paraméterrel, amely leírja az esetüket. (A főnevek nem rendelkeznek esettel az angolban, bár sok más nyelvben igen.) Az  $S$ -re vonatkozó szabályban az  $NP$ -nek alanyi esetűnek kell lennie, míg a  $VP$  és  $PP$  szabályokban tárgyasnak. Az  $NP$  szabály egy változót fogad paraméterként, az *eset*-et. A szándék az, hogy az  $NP$  tetszőleges esetű lehet, de ha átírjuk *Pronoun*-ra, akkor ugyanolyan esetűnek kell lennie. Egy változó ilyen használata – elkerülve a döntést, amikor a különbségtétel nem fontos – visszatartja a szabályhalmazt méretének a tulajdonságok számával arányosan exponenciális növekedését.

A kiterjesztés ezen formalizmusát **definit klóz nyelvtannak (definite clause grammar)** vagy **DCG**-nek hívjuk, mivel minden nyelvtani szabály értelmezhető a Horn-logikában szereplő definit klózként.<sup>10</sup> Elsőként azt mutatjuk meg, hogy egy normális, kiterjesztés nélküli szabály hogyan értelmezhető definit klózként. minden kategória jelet karakter-sorozatra vonatkozó predikátumnak tekintünk, így az  $NP(s)$  igaz, ha az  $s$  karaktersorozat egy  $NP$ -t formál. Az

$$S \rightarrow NP \ VP$$

CFG-szabály a következő definit klóz rövidítése:

$$NP(s_1) \wedge VP(s_2) \Rightarrow S(s_1 + s_2)$$

Itt  $s_1 + s_2$  két karaktersorozat összefűzését jelenti, így ez a szabály azt mondja, hogy ha van egy  $s_1$  füzér, ami egy  $NP$  és egy  $s_2$  füzér, ami egy  $VP$ , akkor az összekapcsolásukkal keletkező füzér egy  $S$ , ami pontosan megegyezik azzal, ahogya a CFG-szabályt korábban értelmeztük. Fontos észrevennünk, hogy a DCG-k lehetővé teszik számunkra, hogy az elemzésről mint logikai következetésről beszéljünk. Ez lehetővé teszi, hogy nyelvek és karaktersorozatok felett sok különböző módon következtessünk. Például ez azt jelenti, hogy egy lentről felfelé elemzést előrefelé következetéssel, egy fentől lefelé elemzést pedig hátrafelé következetéssel végezhetünk el. Azt is látni fogjuk, hogy ez azt is jelenti, hogy ugyanazt a nyelvtant elemzésre és generálásra is használhatjuk.

A DCG-megközelítés igazi haszna az, hogy kiterjeszthetjük a kategóriaszimbólumokat a karakterfüzéreken kívül további argumentumokkal. Például a

$$NP(eset) \rightarrow Pronoun(eset)$$

szabály a következő definit klóz rövidítése:

$$Pronoun(eset, s_1) \Rightarrow NP(eset, s_1)$$

<sup>10</sup> Idézze fel, hogy a definit klóz, amit ha implikációs formában írnunk, pontosan egy atomot tartalmaz a következmény részében, és nulla vagy több atom konjunkcióját a feltétel részében. Két példa az  $A \wedge B \wedge \dots \Rightarrow C$ , illetve önmagában a  $C$ .

Eszerint ha az  $s_1$  füzér az *eset* változó által meghatározott esetű *Pronoun*, akkor  $s_1$  egy ugyanolyan esetű *NP* is. Általánosságban egy kategóriaszimbólumot tetszőleges számú argumentummal kiterjeszhetünk, és az argumentumok egyben paraméterek, amelyek a szokásos Horn-klóz következtetés egyesítésének tárgyai.

Van ára is ennek a kényelemnek: a nyelvtan íróját ellátjuk a tételebizonyítók teljes erejével, így feladjuk a szintaktikai elemzők által garantált  $O(n^3)$ -at; a kiterjesztésekkel kiégészített elemzés lehet NP-teljes, vagy akár eldönthetetlen, a kiterjesztéstől függően.

Néhány további trükk szükséges ahhoz, hogy a DCG működjön; például szükségünk van egy módszerre, hogy záró szimbólumokat adhassunk meg, és kényelmes egy olyan módszert is alkalmaznunk, amely *nem* engedi az automatikus karakterSORozat-argumentum hozzáadását. Mindent összerakva, a definit klóz nyelvtant a következőképpen definiálhatjuk:

- Az  $X \rightarrow Y Z \dots$  jelölés fordítása  $Y(s_1) \wedge Z(s_2) \wedge \dots \Rightarrow X(s_1 + s_2 + \dots)$
- Az  $X \rightarrow Y | Z | \dots$  jelölés fordítása  $Y(s) \vee Z(s) \vee \dots \Rightarrow X(s)$
- Mindkét megelőző szabályban bármelyik  $Y$  nem záró szimbólum egy vagy több argumentummal kiterjeszhető. Bármely argumentum lehet változó, konstans vagy argumentumok függvénye. A fordításban ezen argumentumok megelőzik a karakterfüzér argumentumot (például az *NP(eset)* fordítása *NP(eset, s<sub>1</sub>)*).
- A  $\{P(\dots)\}$  jelölés szerepelhet egy szabály jobb oldalán, és változás nélkül átalakul  $P(\dots)$ -vé. Ez lehetővé teszi a nyelvtan írója számára, hogy egy  $P(\dots)$ -re vonatkozó tesztet anélkül elhelyezzen, hogy ahhoz automatikusan hozzáadódjon a karakterfüzér argumentum.
- Az  $X \rightarrow \text{word}$  jelölés fordítása  $X([\text{word}])$ .

Az alany-ige egyeztetése szintén kezelhető kiterjesztéssel, de ezt a 22.2. feladatra halasztjuk. Helyette egy nehezebb problémával foglalkozunk: az igék alkategóriákba osztásával.

## Igék alkategóriákba osztása

Az  $\mathcal{E}_1$  nyelv fejlettebb az  $\mathcal{E}_0$ -hoz képest, de még ez is túlgenerál. Az egyik probléma abban a módban rejlik, ahogy az igei kifejezéseket összefűzi. El akarjuk fogadni a „give me the gold” és a „go to 1,2” jellegű igés kifejezéseket. Ezek mindegyike létezik az  $\mathcal{E}_1$ -ben, de sajnos éppígy létezik a „go me the gold” és a „give to 1,2” is. Az  $\mathcal{E}_2$  nyelv megszünteti ezeket a *VP*-ket úgy, hogy egyértelműen megadja, melyik igét melyik kifejezés követheti. Ezt az igék **alkategória** (*subcategorization*) listájának hívjuk. Az el-képzelés az, hogy a *Verb* kategóriát alkategóriákra bontjuk – egy a tárggyal nem rendelkezők igék számára, egy az egyetlen tárggyal rendelkezőkre és így tovább.

Az ötlet megvalósításához minden egyes igéhez rendelünk egy alkategória-listát (*subcategorization list*), amely a kiegészítőit (*complements*) tartalmazza. A kiegészítő olyan kötelező kifejezés, amely az igei kifejezésen belül követi az igét. Azaz a „give the gold to me”-ben az *NP* a „the gold” és a „to me” *PP* a „give” kiegészítője.<sup>11</sup> Ezt leírhatjuk így:

$$\text{Verb}([\text{NP}, \text{PP}]) \rightarrow \text{give} \mid \text{hand} \mid \dots$$

<sup>11</sup> Ez a kiegészítő (*complement*) egyik definíciója, de vannak más terminológiát használó szerzők is. Egyesek szerint az ige alanya is egy kiegészítő. Mások szerint csak az előljárói kifejezés kiegészítő, és a főnévi kifejezést egy érvélésnek (*argument*) kell hívni.

Lehetséges, hogy egy igéhez több, különböző alkategóriára bontás létezik, csakúgy, mint ahogyan egy szó számára is lehetséges, hogy több, különböző kategóriába tartozzon. Valójában a „give” rendelkezik az [NP, NP] alkategória-listával is, mint a „Give me the gold” mondatban. Ezt ugyanúgy kezelhetjük, mint minden más többérteleműséget. A 22.12. ábra ad pár példát igékre és alkategória-listáikra (angolul röviden *subcat*).

Ige	Alkategóriák	Példa igei kifejezés
give	[NP, PP] [NP, NP]	give the gold in 3 3 to me give me the gold
smell	[NP] [Adjective] [PP]	smell a wumpus smell awful smell like a wumpus
is	[Adjective] [PP] [NP]	is smelly is in 2 2 is a pit
died	[]	died
believe	[S]	believe the wumpus is dead

22.12. ábra. Példák igékre az alkategória-listáikkal együtt

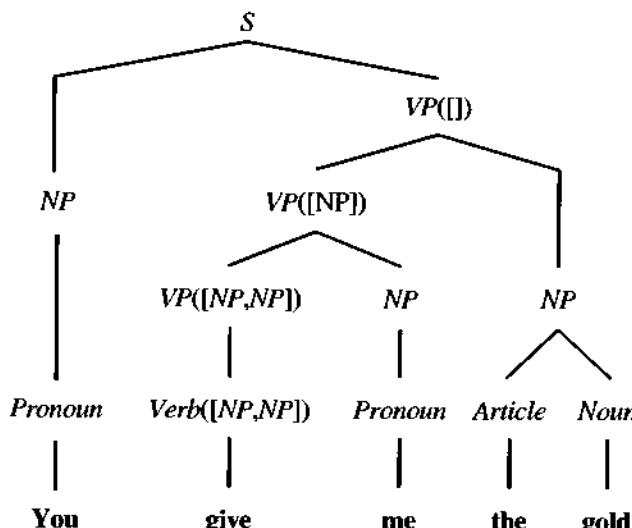
Három dolgot teszünk az igei alkategóriák nyelvtanba illesztéséhez. Az első lépés a VP kategória kiterjesztése egy alkategória-argumentummal, *VP(subcat)*, amely felülrolja azokat a komponenseket, amikkel teljes VP formálható. Például a „give” teljes VP-vé tehető [NP, PP] hozzáadásával, „give the gold” teljessé tehető [PP] hozzáadásával, és a „give the gold to me” már teljes; alkategória-listája üres, []. Ez a következő szabályokat adja VP-re:

$$\begin{array}{ccc}
 VP(subcat) & \rightarrow & Verb(subcat) \\
 & | & \\
 & VP(subcat + [NP]) NP(Objective) & \\
 & | & \\
 & VP(subcat + [Adjective]) Adjective & \\
 & | & \\
 & VP(subcat + [PP]) PP &
 \end{array}$$

Az utolsó sor így olvasható: „Adott *subcat* alkategória-listával rendelkező VP formálható egy beágyazott VP-ből, amit egy PP követ, akkor, ha a beágyazott VP rendelkezik egy olyan alkategória-listával, ami a *subcat* lista elemeivel kezdődik és PP-vel folytatódik.” Például egy *VP([])*-t egy *VP([PP])* és az azt követő PP alkot. Az első sor szerint egy VP, amely *subcat* alkategória-listával rendelkezik, formálható egy ugyanolyan alkategória-listával rendelkező Verb segítségével. Például a *VP([NP])* létrehozható egy *Verb([NP])* által. Egy példa ilyen igére a „grab”, tehát a „grab the gold” egy *VP([])*.

A második lépés az S-re vonatkozó szabály megváltoztatása úgy, hogy szükségesse tegyen egy igei kifejezést, melynek összes kiegészítése megvan, és így [] alkategória-listája van. Ez azt jelenti, hogy az „I grab the gold” érvényes mondat, de a „You give” nem az. Az új szabály,

$$S \rightarrow NP(Subjective) VP([])$$



**22.13. ábra.** A „You give me the gold (Ideadod nekem az aranyat)” elemzési fája, amely bemutatja az ige és az igei kifejezés alkategóriáit

a következőképpen olvasható: „Egy mondatot összeállíthatunk egy alanyi esetben álló *NP*-ből és az azt követő, üres alkategória-listájú *VP*-ból.” A 22.13. ábra bemutat egy elemzési fát, amelyik ezt a nyelvtant használja.

A harmadik lépésben gondolunk arra, hogy a kiegészítések mellett az igei (és egyéb) kifejezések segédszavakat (*adjuncts*) is kaphatnak, melyek nem egy igéhez kötődő kifejezések, hanem mindenféle igei kifejezésben feltűnhetnek. Az időt és helyet jelképező kifejezések segédszavak, mivel szinte minden cselekvés vagy esernyő rendelkezhet időponttal és helyszínnel. Például a „now” határozószerző az „I smell a wumpus now” mondatban és az „on Tuesday” *PP* a „give me the gold on Tuesday”-ban segédszavak. Íme, két szabály a segédszavak megengedéséhez:

$$\begin{array}{ccc} VP(subcat) & \rightarrow & VP(subcat) PP \\ & | & \\ & & VP(subcat) Adverb \end{array}$$

## Kiterjesztett nyelvtanok generálóképessége

Minden kiterjesztett szabály egy **szabályséma** (rule schema), mely a szabályok egy olyan halmazát jelenti, amit a kiterjesztett alkotórészek értékeinek minden lehetséges kombinációjával állítunk elő. A kiterjesztett nyelvtanok generálóképessége a kombinációk számától függ. Ha véges számú van, akkor a kiterjesztett nyelvtan ekvivalens egy környezetfüggetlen nyelvtannal: a szabályséma felcserélhető egyedi környezetfüggetlen szabályokkal. Azonban ha végtelen számú van, akkor a kiterjesztett nyelvtanok nem környezetfüggetlen nyelveket is leírhatnak. Például az *a"b"c"* környezetfüggő nyelv leírható a következő kiterjesztett nyelvtannal:

$$\begin{array}{ll}
 S(n) \rightarrow A(n) \ B(n) \ C(n) \\
 A(1) \rightarrow \mathbf{a} & A(n+1) \rightarrow \mathbf{a} \ A(n) \\
 B(1) \rightarrow \mathbf{b} & B(n+1) \rightarrow \mathbf{b} \ B(n) \\
 C(1) \rightarrow \mathbf{c} & C(n+1) \rightarrow \mathbf{c} \ C(n)
 \end{array}$$

## 22.5. SZEMANTIKAI ÉRTELMEZÉS

Eddig csak a nyelv szintaktikai analízisét vizsgáltuk. Ebben a fejezetben rátérünk a szemantikára (**semantics**) – a megnyilatkozás (utterance) jelentésének kinyerésére. Ebben a fejezetben az elsőrendű logikát használjuk reprezentációs nyelvként, így a szemantikai értelmezés egy FOL kifejezés és egy nyelvi kifejezés összerendelésének a folyamata. Intuitív módon közelítve, a „the wumpus” kifejezés jelentése az a nagy, szörös szörnyeteg, amit a logikában a  $Wumpus_1$  logikai kifejezéssel jelzünk, és a „the wumpus is dead” jelentése a  $Dead(Wumpus_1)$  logikai állítás. Ez a fejezet ezt az intuíciót pontosítja. Egy egyszerű példával kezdünk: a négyzetháló helyszíneit leíró szabállyal:

$$NP \rightarrow Digit \ Digit$$

Kiterjesztjük a szabályt úgy, hogy minden összetevőhöz egy argumentumot illesztünk, amely az összetevő szemantikáját reprezentálja. A következőt kapjuk:

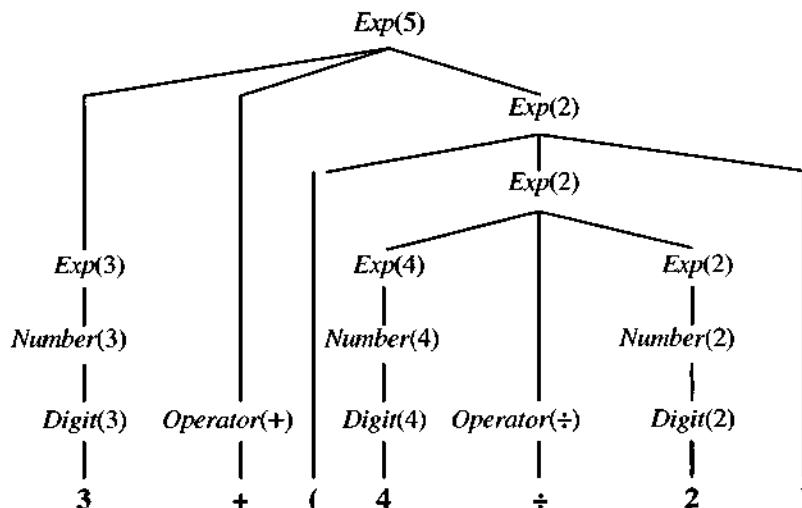
$$NP([x, y]) \rightarrow Digit(x) \ Digit(y)$$

E szerint egy karaktersorozat, amely egy  $x$  szemantikájú számjegyből és egy azt követő  $y$  szemantikájú számjegyből áll, egy  $[x, y]$  szemantikával rendelkező  $NP$ -t formál, amely a háló egy négyzetére alkalmazott jelölésünk.

Vegyük észre, hogy a teljes  $NP$  szemantikája nagymértékben az összetevőinek szemantikáján alapszik. Már láttuk a **kompoziciós szemantika** (**compositional semantics**) ezen ötletét korábban: a logikában a  $P \wedge Q$  jelentését  $P, Q$  és  $\wedge$  határozzák meg; az aritmetikában az  $x + y$  jelentését  $x, y$  és  $+$  határozzák meg. A 22.14. ábra megmutatja, hogy a DCG jelölésrendszer hogyan használható egy aritmetikai kifejezéseket leíró nyelvtan szemantikával történő kiterjesztésére, és a 22.15. ábra mutatja a  $3 + (4 \div 2)$  ezen nyelvtan alapján kapott elemzési fáját. Az elemzési fa gyökere  $Exp(5)$ , egy kifejezés, melynek szemantikai értelmezése 5.

$$\begin{aligned}
 Exp(x) &\rightarrow Exp(x_1) \ Operator(op) \ Exp(x_2) \quad \{x = Apply(op, x_1, x_2)\} \\
 Exp(x) &\rightarrow (Exp(x)) \\
 Exp(x) &\rightarrow Number(x) \\
 Number(x) &\rightarrow Digit(x) \\
 Number(x) &\rightarrow Number(x_1) \ Digit(x_2) \quad \{x = 10 \times x_1 + x_2\} \\
 Digit(x) &\rightarrow x \quad \{0 \leq x \leq 9\} \\
 Operator(x) &\rightarrow x \quad \{x \in \{+, -, \div, \times\}\}
 \end{aligned}$$

**22.14. ábra.** Szemantikával kibővített nyelvtan az aritmetikai kifejezésekre. minden  $x_i$  változó egy összetevő szemantikáját reprezentálja. Vegye észre, hogy a {teszt} jelölést használjuk olyan logikai predikátumok definiálására, amelyeket ki kell elégíteni, de nem összetevők!

22.15. ábra. A „ $3 + (4 \div 2)$ ” füzér elemzési fája szemantikai értelmezésekkel

## Az angol nyelv egy részletének szemantikája

Most már készen állunk arra, hogy az angol nyelv egy kis részhalmazára megírjuk a szemantikai kibővítést. Első lépésként azzal kezdünk, hogy melyik kifejezéshez milyen szemantikai értelmezéseket akarunk rendelni. A „John loves Mary” egyszerű mondatot fogjuk vizsgálni. A „John” *NP* szemantikus értelmezése a *John* logikai term kell legyen, és a mondatnak, mint egésznek, a *Loves(John, Mary)* logikai állítás kell a szemantikai értelmezése legyen. Ennyi világosnak látszik. A bonyolultabb rész a „loves Mary” *VP*. Ezen kifejezés szemantikai értelmezése se nem logikai term, se nem teljes logikai mondat. Intuitív módon kezelhetjük úgy, hogy a „loves Mary” egy leírás, ami lehet, hogy egy adott személyre vonatkozik, de lehet, hogy nem. (Jelen esetben Johnra vonatkozik.) Ez azt jelenti, hogy a „loves Mary” egy **predikátum (predicate)**, amit ha egy személyt reprezentáló termmel kombinálunk (a személy, aki szeret), akkor egy teljes logikai mondatot állít elő. A λ jelölésrendszert használva (lásd 7.2. alfejezet) a „loves Mary”-t a következő predikátumként reprezentálhatjuk:

$$\lambda x \text{ Loves}(x, \text{Mary})$$

Ezután szükségünk van egy szabályra, amely szerint „egy *obj* szemantikájú *NP*, melyet egy *rel* szemantikájú *VP* követ, együtt egy olyan mondatot eredményez, amelynek szemantikája a *rel* alkalmazása az *obj*-ra:

$$S(\text{rel}(\text{obj})) \rightarrow NP(\text{obj}) VP(\text{rel})$$

A szabály szerint a „John loves Mary” szemantikai értelmezése

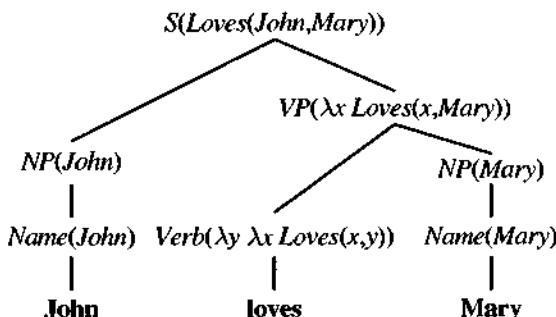
$$(\lambda x \text{ Loves}(x, \text{Mary}))(John)$$

ami megfelel a *Loves(John, Mary)*-nek.

$S(rel(obj)) \rightarrow NP(obj) VP(rel)$   
 $VP(rel(obj)) \rightarrow Verb(rel) NP(obj)$   
 $NP(obj) \rightarrow Name(obj)$

$Name(John) \rightarrow John$   
 $Name(Mary) \rightarrow Mary$   
 $Verb(\lambda y \lambda x Loves(x, y)) \rightarrow loves$

22.16. ábra. Egy nyelvtan, amely képes egy elemzési fa és szemantikai értelmezés levezetésére a „John loves Mary” (és három másik) mondat számára



22.17. ábra. A „John loves Mary” füzér elemzési fája szemantikai értelmezésekkel

A szemantika hátralevő része az eddigi választásainkból egyenesen következik. Mivel a  $VP$ -ket predikátumként reprezentáljuk, jó ötlet konzisztens módon az igéket is predikátumként ábrázolni. A „loves” ige reprezentációja  $\lambda y \lambda x Loves(x, y)$  predikátum, ami például egy  $Mary$  argumentum esetén  $\lambda x Loves(x, Mary)$  predikátummal tér vissza.

A  $VP \rightarrow Verb\ NP$  szabály alkalmazza az ige szemantikai értelmezéséből kapott predikátumot az  $NP$  (szemantikai) értelmezéséből kapott objektumra a teljes  $VP$  szemantikai értelmezésének kinyerésére. Végül a 22.16. ábrán látható nyelvtant és a 22.17. ábrán található elemzési fát kapjuk.

## Idő és igeidő

Most tegyük fel, hogy a „John loves Mary” és a „John loved Mary” közötti különbséget szeretnénk reprezentálni. Az angol igeidőket (past [múlt], present [jelen] és future [jövő]) használ egy esemény relatív idejének jelzésére. Egy jó választás az események idejének reprezentálására a 10.3. alfejezet eseménykalkulus jelölésszabánya. Az eseménykalkulussal a két mondatunk a következőképpen értelmezhető:

$$\begin{aligned} e \in Loves(John, Mary) \wedge During(Now, e) \\ e \in Loves(John, Mary) \wedge After(Now, e) \end{aligned}$$

Eszerint a „loves” és „loved” szavakra vonatkozó két lexikai szabályunk a következő lehet:

$$\text{Verb}(\lambda y \lambda x e \in \text{Loves}(John, Mary) \wedge \text{During}(Now, e)) \rightarrow \text{loves}$$

$$\text{Verb}(\lambda y \lambda x e \in \text{Loves}(x, y) \wedge \text{After}(Now, e)) \rightarrow \text{loved}$$

E változástól eltekintve a nyelvtannal kapcsolatban minden más változatlan marad, ami összönöz hír; azt sugallja, hogy helyes úton járunk, ha ilyen könnyedén bonyolíthatjuk például az igeidővel (bár csak érintettük az időre és igeidőre vonatkozó teljes nyelvtan felszínét). E bemelegítésként elért sikerrel készen állunk egy sokkal nehezebb reprezentációs probléma kezelésére.

## Kvantifikálás

Nézzük a következő példát: „Every agent smells a wumpus.” A mondat valójában többértelmű: a javasolt értelmezés szerint az ágensek különböző wumpusokat észlelhetnek, de egy alternatív jelentés szerint csak egy wumpus van, akit mindenki érez.<sup>12</sup> A két interpretációt a következő módon ábrázolhatjuk:

$$\begin{aligned} \forall a \ a \in \text{Agents} \Rightarrow \exists w \ w \in \text{Wumpuses} \wedge \exists e \ e \in \text{Smell}(a, w) \wedge \text{During}(Now, e) \\ \exists w \ w \in \text{Wumpuses} \ \forall a \ a \in \text{Agents} \Rightarrow \exists e \ e \in \text{Smell}(a, w) \wedge \text{During}(Now, e) \end{aligned}$$

A többértelműség problémáját elhalasztjuk későbbre, most csak az első értelmezést vizsgáljuk. Megpróbáljuk az összetevői szerint elemezni, *NP* és *VP* komponensekre bontva:

$$\begin{aligned} \text{Every agent } & NP(\forall a \ a \in \text{Agents} \Rightarrow P) \\ \text{smells a wumpus } & VP(\exists w \ w \in \text{Wumpuses} \wedge \exists e \ (e \in \text{Smell}(a, w) \wedge \text{During}(Now, e))) \end{aligned}$$

Máris két nehézség tárult elő: először is, a teljes mondat szemantikája, úgy tűnik, meggyezik az *NP* szemantikájával, melyben a *VP* tölti ki a *P* rész szemantikáját. Ez azt jelenti, hogy a mondat szemantikáját nem formálhatjuk *rel(obj)* segítségével. Megtehetjük *obj(rel)* által, ami (legalábbis első ránézésre) kicsit különösnek tűnik. A második probléma az, hogy az *a* változót a *Smell* reláció argumentumaként kell megkapnunk. Más szavakkal, a mondat szemantikája a *VP* szemantikájának megfelelő *NP* argumentumrekeszbe való illesztésével adódik, miközben az *NP*-ben található *a* változót is betessük a *VP* szemantika megfelelő argumentumrekeszébe. Úgy tűnik, mintha két funkcionális kompozíció lenne, és ez azt sugallja, hogy meglehetősen könnyű lesz összeavarodni. A bonyolultság abból a tényből ered, hogy a szemantikai struktúra nagyon különböző a szintaktikaitól.

Hogy elkerüljék ezt a zűrzavart, sok modern nyelvtan más módszert választ. Definiálnak egy **közbülső formát (intermediate form)** a szintaxis és a szemantika közötti közvetítésre. A közbülső formának két kulcskulajdonsága van. Egyrészt strukturálisan hasonló a mondat szintaxisához, így könnyen előállítható kompozíciós eszközökkel. Másrészt elég információt tartalmaz, és így lefordítható egy reguláris elsőrendű logikai

<sup>12</sup> Ha ez az értelmezés valószínűlennekk tűnik, akkor vizsgálja meg az „Every Protestant believes in a just God” mondatot.

mondattá. Mivel a szintaktikai és a logikai alak között helyezkedik el, néha **kvázilogikai formának (quasi-logical form)**<sup>13</sup> nevezik. A fejezetben egy kvázilogikai formát használunk, mely a teljes elsőrendű logikát tartalmazza, lambdafejezésekkel és egy új konstrukcióval kiegészítve, amit **kvantifikált termnek (quantified term)** nevezünk. Az „every agent” szemantikai értelmezését jelentő kvantifikált term a következőképpen írható:

$$[\forall a \ a \in Agents]$$

Ez olyan, mint egy logikai mondat, de ugyanolyan módon használjuk, mint a logikai termeket. Az „Every agent smells a wumpus” kvázilogikai értelmezése:

$$\exists e (e \in Smell([\forall a \ a \in Agents], [\exists w \ w \in Wumpuses]) \wedge During(Now, e))$$

A kvázilogikai alak generálásához szabályaink közül sok változatlan marad. Az *S*-re vonatkozó szabály *S* szemantikáját még mindig *rel(obj)* segítségével adja meg. Bizonyos szabályok változnak; az „a” szóra vonatkozó lexikai szabály a következő:

$$Article(\exists) \rightarrow a$$

míg a névelő és fónév kombinációjának szabálya:

$$NP([q \ x \ sem(x)]) \rightarrow Article(q) \ Noun(sem)$$

E szerint az *NP* szemantikája egy kvantifikált term – ahol a kvantifikálást a névelő határozza meg – egy új *x* változóval és egy propozícióval, amely a fónév szemantikáját az *x* változóra alkalmazza. Az *NP*-re vonatkozó további szabályok hasonlók. A 22.18. ábra mutatja be a szemantikai típusokat, és példát ad minden egyes szintaktikai kategóriára a kvázilogikai ábrázolás szerint. A 22.19. ábra az „Every agent smells a wumpus” e megközelítés szerinti elemzését mutatja, a 22.20. pedig a teljes nyelvtant.

Ezek után át kell alakítanunk a kvázilogikai formát valódi elsőrendű logikaivá a kvantifikált termek igazi term-mé alakításával. Ezt egy egyszerű szabállyal tesszük meg: minden egyes *QLF* kvázilogika formán belüli  $[q \ x \ P(x)]$  kvantifikált termet a cseréljük ki *x*-re, és cseréljük a *QLF*-et  $q \ x \ P(x)$  op *QLF* formára, ahol az *op* operátor  $\Rightarrow$  akkor, ha a *q*  $\forall$ , és  $\wedge$ , amennyiben a *q*  $\exists$  vagy  $\exists!$ . Például az „Every dog has a day (Mindnen kutyának van egy napja)” mondat a következő kvázilogikai formával rendelkezik:

$$\exists e (e \in Has([\forall d \ d \in Dogs], [\exists a \ a \in Days], Now))$$

Nem specifikáltuk, hogy a két kvantifikált term melyikét kell elsőként elővenni, így valójában két lehetséges értelmezés van:

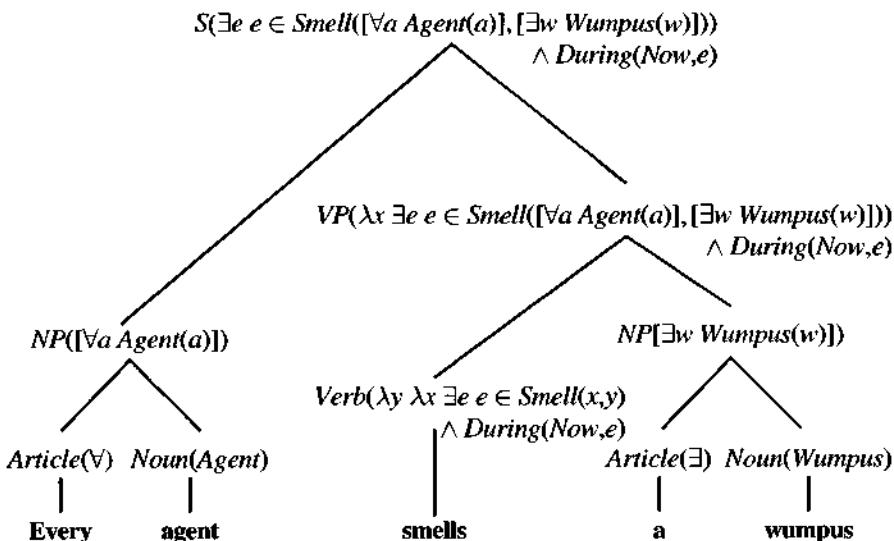
$$\forall d \ d \in Dogs \Rightarrow \exists a \ a \in Days \wedge \exists e \ e \in Has(d, a, Now)$$

$$\exists a \ a \in Days \wedge \forall d \ d \in Dogs \Rightarrow \exists e \ e \in Has(d, a, Now)$$

<sup>13</sup> Egyes kvázilogikai formákra jellemző az is, hogy olyan többértelműségek tömör kifejezésére is alkalmasak, amelyeket logikai formában csak hosszú diszjunkciók segítségével lehetne kifejezni.

Kategória	Típus	Példa	Kvázilogikai alak
<i>S</i>	<i>Sentence</i>	I sleep.	$\exists e \in Sleep(Speaker) \wedge During(Now, e)$
<i>NP</i>	<i>object</i>	a dog	$\{\exists d Dog(d)\}$
<i>PP</i>	<i>object</i> <sup>2</sup> → <i>sentence</i>	in [2, 2]	$\lambda x In(x, [2, 2])$
<i>RelClause</i>	<i>object</i> → <i>sentence</i>	that sees me	$\lambda x \exists e \in Sees(x, Speaker) \wedge During(Now, e)$
<i>VP</i>	<i>object</i> <sup>n</sup> → <i>sentence</i>	sees me	$\lambda x \exists e \in Sees(x, Speaker) \wedge During(Now, e)$
<i>Adjective</i>	<i>object</i> → <i>sentence</i>	smelly	$\lambda x Smelly(x)$
<i>Adverb</i>	<i>event</i> → <i>sentence</i>	today	$\lambda e During(e, Today)$
<i>Article</i>	<i>quantifier</i>	the	$\exists!$
<i>Conjunction</i>	<i>sentence</i> <sup>2</sup> → <i>sentence</i>	and	$\lambda p, q (p \wedge q)$
<i>Digit</i>	<i>object</i>	7	7
<i>Noun</i>	<i>object</i> → <i>sentence</i>	wumpus	$\lambda x x \in Wumpuses$
<i>Preposition</i>	<i>object</i> <sup>2</sup> → <i>sentence</i>	in	$\lambda x \lambda y In(x, y)$
<i>Pronoun</i>	<i>object</i>	I	<i>Speaker</i>
<i>Verb</i>	<i>object</i> <sup>n</sup> → <i>sentence</i>	eats	$\lambda y \lambda x \exists e Eats(x, y) \wedge During(Now, e)$

**22.18. ábra.** minden szintaktikai kategória kvázilogikai formájú kifejezésének típusát mutató tábla. A  $t \rightarrow r$  jelölés egy függvényt takar, amely egy  $t$  típusú argumentumot fogad, és  $r$  típusú eredménnyel tér vissza. Például a Preposition szemantikai típusa  $object^2 \rightarrow sentence$ , ami azt jelenti, hogy az előjárószó szemantikája egy függvény, amelyet ha két logikai objektumra alkalmazunk, akkor egy logikai mondatot eredményez.



**22.19. ábra.** Az „Every agent smells a wumpus” mondat elemzési fája, amely mind a szintaktikai struktúrát, mind a szemantikai értelmezéseket mutatja

$S(rel(obj)) \rightarrow NP(obj) VP(rel)$   
 $S(conj(sem_1, sem_2)) \rightarrow S(sem_1) Conjunction(conj) S(sem_2)$ 
  
  
 $NP(sem) \rightarrow Pronoun(sem)$   
 $NP(sem) \rightarrow Name(sem)$   
 $NP([q \times sem(x)]) \rightarrow Article(q) Noun(sem)$   
 $NP([q \times obj \wedge rel(x)]) \rightarrow NP([q \times obj]) PP(rel)$   
 $NP([q \times obj \wedge rel(x)]) \rightarrow NP([q \times obj]) RelClause(rel)$   
 $NP([sem_1, sem_2]) \rightarrow Digit(sem_1) Digit(sem_2)$ 
  
  
 $VP(sem) \rightarrow Verb(sem)$   
 $VP(rel(obj)) \rightarrow VP(rel) NP(obj)$   
 $VP(sem_1 (sem_2)) \rightarrow VP(sem_1) Adjective(sem_2)$   
 $VP(sem_2 (sem_2)) \rightarrow VP(sem_1) PP(sem_2)$ 
  
  
 $RelClause(sem) \rightarrow that VP(sem)$ 
  
  
 $PP(\lambda x rel(x, obj)) \rightarrow Preposition(rel) NP(obj)$

22.20. ábra. Szemantikával ellátott nyelvtan kvázilogikai formában

Az első szerint minden kutyának megvan a maga napja, míg a második szerint van egy különleges nap, ami minden kutyára ugyanaz. A kettő közötti választás a többértelműség feloldásának feladata. A kvantifikált termek balról jobbra sorrendje gyakran illeszkedik a kvantifikálók balról jobbra sorrendjére, de más tényezők is számítanak. A kvázilogikai alak előnye az, hogy tömörén reprezentálja az összes lehetőséget. A hátránya pedig az, hogy nem segíti a választást közöttük; amihez szükségünk van a többértelműség feloldásának teljes erejére az összes tényt felhasználva.

## Pragmatikus értelmezés

Megmutattuk, hogy egy ágens hogyan képes észlelni egy szófűzért, és hogyan képes egy nyelvtant használni a lehetséges szemantikai értelmezések halmazának előállítására. Most annak a problémáját vizsgáljuk, hogy az értelmezés hogyan tehető teljessé az egyes jelöltek értelmezéshez az adott szituációt leíró kontextusfüggő információk hozzáadásával.

A gyakorlati információra legnyilvánvalóbban a **referenciális indexek (indexicals)** feloldásához van szükség, amelyek közvetlenül az adott szituációra utaló kifejezések. Például az „I am in Boston today (Ma Bostonban vagyok)” mondatban az „I” és „today” referenciális indexek értelmezése attól függ, hogy ki és mikor ejtette ki a mondatot. A referenciális indexeket „konstansokkal” (mint például a *Speaker*) reprezentáljuk, amelyek valójában változó **dolgok (fluents)** – azaz a szituációtól függnak. A hallgatónak, aki észleli a szólásaktust, azt is észlelnie kell, hogy ki a beszélő, és felhasználja ezt az információt a referenciális indexek feloldására. Például a hallgató tudhatja, hogy  $T((\text{Speaker} = \text{Agent}_B), \text{Now})$ .

Egy felszólítás, mint például a „go to 2, 2” implicit módon a hallgatóra vonatkozik. Eddig az *S* nyelvtanunk csak deklaratív mondatokat írt le. Könnyen kiterjeszhetjük felszólítások kezelésére.<sup>14</sup>

A felszólítást olyan *VP* segítségével formálhatjuk, ahol az alany implicit módon a hallgató. Meg kell különböztetnünk a felszólításokat az állításuktól, ezért megváltoztatjuk az *S*-re vonatkozó szabályokat a szólásaktus típusának beillesztésével a kvázilogikai formába:

$$\begin{aligned} S(\text{Statement}(Speaker, rel(obj))) &\rightarrow NP(obj) VP(rel) \\ S(\text{Command}(Speaker, rel(Hearer))) &\rightarrow VP(rel) \end{aligned}$$

Így a „go to 2, 2” kvázilogikai alakja:<sup>15</sup>

$$\text{Command}(\exists e \in Go(Hearer, [2, 2]))$$

## Nyelv generálása DCG-kkel

Eddig egy nyelv *elemzésével* foglalkoztunk, nem a generálásával. A generálás hasonlóan gazdag téma. A megfelelő megnyilatkozás kiválasztása egy állítás kifejezésére sok hasonló választást von maga után, mint egy megnyilatkozás elemzése.

Emlékezzünk vissza, hogy a DCG egy logikai programozási rendszer, amely kényszerűket határoz meg egy karaktersorozat és annak elemzése között. Tudjuk, hogy az *Append* predikátum logikai programozási definíciója felhasználható annak közlésére is, hogy az *Append*([1, 2], [3], *x*) esetében *x* = [1, 2, 3], és arra is, hogy felsoroljuk azon *x* és *y* értékeit, melyek az *Append*(*x*, *y*, [1, 2, 3]) kifejezést igazzá teszik. Hasonlóképpen írhatunk egy definíciót *S*-re, amely kétféleképpen használható: az elemzéshez megkérdezzük, hogy *S*(*sem*, [*John*, Loves, *Mary*]), és azt kapjuk vissza, hogy *sem* = Loves(*John*, *Mary*); a generáláshoz azt kérdezzük, hogy *S*(Loves(*John*, *Mary*), *words*), és azt kapjuk vissza, hogy *words* = [*John*, Loves, *Mary*]. Tesztelhetünk egy nyelvtant az *S*(*sem*, *words*) kérdéssel, visszakapva azokat a [*sem*, *words*] párok sorozatát válaszként, melyeket a nyelvtan generált.

Ez a módszer a fejezetben bemutatott egyszerű nyelvtanokra működik, de nagyobb nyelvtanokra történő felskálázás során lehetnek problémák. A logikai következetető gép által használt keresési stratégia nagyon fontos; a mélységi keresési stratégiák végtelen ciklusokhoz vezethetnek. Figyelmet kell fordítani a szemantikai alak pontos részleteire is.

<sup>14</sup> Egy teljes kommunikáló ágens megvalósításához szükségünk lenne a kérdések nyelvtanára is. A kérdések kezelése kívül esik e könyv területén, mivel összetevők közötti hosszú távú függvégeket (long-distance dependencies) vonzanak magukkal. Például a „Whom did the agent tell you to give the gold to?” mondatban a záró „to” elemzésének hiányzó *NP*-jű *PP*-nek kell lennie; a hiányzó *NP*-t a mondat első szava, a „who” helyettesíti. Kiterjesztések egy komplex rendszere biztosítja, hogy a hiányzó *NP*-k összeillesztődjenek a helyettesítő szavakkal.

<sup>15</sup> Végyük észre, hogy egy felszólítás kvázilogikai formája nem foglalja magában az esemény idejét (azaz a *During*(*Now*, *e*)-t)! Ez azért van, mert a „go” valójában a szó igeidő nélküli változata, nem a jelen idejű. Nem lehet megállapítani a különbséget a „go” esetében, de figyelje meg a „Be good!” felszólítás helyes alakját (a „be” igeidő nélküli alakját használva), amely nem „Are good!”. Annak biztosítására, hogy a helyes igeidőt használjuk, kiegészíthetnénk a *VP*-ket egy igeidő kiterjesztéssel, és a felszólítások szabályának jobb oldalára ezt írhatnánk: *VP*(*rel*, *untensed*).

Előfordulhat, hogy egy adott nyelvtan nem tudja kifejezni az  $X \wedge Y$  logikai alakot az  $X$  és  $Y$  bizonyos értékeire, de ki tudja fejezni  $Y \wedge X$ -et; ez azt sugallja, hogy szükségünk lesz valamilyen módszerre a szemantikai alakok kanonizálására, vagy ki kell terjesztenünk az egyesítő eljárást úgy, hogy az  $X \wedge Y$  és az  $Y \wedge X$  egyesíthető legyen.

Komolyabb generálási feladatok bonyolultabb modelleket használnak, melyek különböznek az elemzés nyelvtanától, és pontosabban szabályozzák, hogy a szemantikai komponenseket pontosan hogyan fejezzük ki. A szisztematikus nyelvtan egy olyan megközelítés, amely könnyűvé teszi, hogy a szemantikai alak legfontosabb részeire nagyobb hangsúlyt helyezzünk.

## 22.6. TÖBBÉRTELMI MŰSÉG ÉS FELOLDÁSA

Bizonyos esetekben a hallgatók tudatában vannak egy megnyilatkozás többértelműségek. Íme, néhány példa a napilapok szalagcíméiből:

Squad helps dog bite victim.

Helicopter powered by human flies.

Once-sagging cloth diaper industry saved by full dumps.

Portable toilet bombed; police have nothing to go on.

British left waffles on Falkland Islands.

Teacher strikes idle kids.

Milk drinkers are turning to powder.

Drunk gets nine months in violin case.

(A mondatok fordítását és értelmezését az olvasóra hagyjuk – a *ford.*)

Azonban legtöbb esetben a beszéd, amit hallunk, egyértelműnek tűnik. Ezért amikor a kutatók először kezdték a nyelv elemzésére számítógépeket használni a hatvanas években, igen meglepődve tapasztalták, hogy *majdnem minden megnyilatkozás erősen többértelmű*, bár az alternatív értelmezések egy anyanyelvi beszélő számára lehet, hogy nem nyilvánvalók. Egy nagy nyelvtannal és szókinccsel rendelkező rendszer értelmezések ezreit találhatja egy teljesen átlagos mondatra. Vegyük például a „The batter hit the ball” mondatot, amelynek látszólag egy egyértelmű értelmezése van, miszerint a baseballjátékos megüt egy baseball-labdát. De különböző értelmezést kapunk, ha az ezt megelőző mondat ez: „The mad scientist unleashed a tidal wave of cake mix towards the ballroom.” Ez a példa a lexikális többértelműségen (lexical ambiguity) alapszik, amikor egy szónak egynél több jelentése van. A lexikális többértelműség igen gyakori: a „back” lehet határozószó (go back), melléknév (back door), főnév (the back of the room) és ige (back up your files). A „Jack” lehet név, főnév (kártya, hatszögű fém játékdarab, hajózási zászló, hal, hím szamár, dugasz vagy nehéz objektumok emelésére szolgáló eszköz), illetve lehet ige (felemelni egy autót, fénnyel vadászni vagy erősen megütni a baseball-labdát).

A szintaktikai többértelműség (syntactic ambiguity) (avagy strukturális többértelműség (structural ambiguity)) a lexikális többértelműséggel együtt vagy nélküle is előfordulhat. Például az „I smelled a wumpus in 2,2” füzérnek két értelmezése lehet: az egyikben az „in 2,2” prepozíciós kifejezés a főnevet módosítja, a másikban az igét. A szintaktikai többértelműség szemantikai többértelműséghoz (semantic

**ambiguity**) vezet, mivel az egyik értelmezés szerint a wumpus a 2,2-n van, a másik szerint a szaga van a 2,2-n. Ebben az esetben a rossz értelmezés halálos tévedés lehet.

A szemantikai többértelműség még olyan kifejezésekben is feltűnhet, ahol nincs lexikális vagy szintaktikai többértelműség. Például a „cat person” főnévi kifejezés takarhat olyasvalakit, aki szereti a macskaféléket, vagy az *Attack of the Cat People* (A macskaemberek támadása) c. film szalagcímét is. Egy „coast road” lehet olyan út, ami a partvonala követi, és olyan, amelyik a partra vezet.

Végezetül lehet többértelműség a szó szerinti és az átvitt értelmű jelentések között. A jelképes beszéd fontos a költészettelben, de meglepően gyakori a minden nap beszédben is. A **metonímia** (*metonymy*) egy olyan jelképes beszéd, amikor az egyik objektum helyére egy másikat állítunk. Amikor azt halljuk, hogy „Chrysler announced a new model”, nem értelmezzük akként, hogy a cégek beszélni tudnak; inkább azt értjük ezáltal, hogy a céget képviselő szóvivő tette a bejelentést. A metonímia gyakori, és az emberi hallgatók gyakran észrevételelőnél interpretálják. Sajnálatos módon a nyelvtanunk abban a formában, ahogy leírtuk, nem ennyire rugalmas. Ahhoz, hogy a metonímia szemantikáját megfelelően kezelni tudjuk, a többértelműség teljesen új szintjét kell bevezetnünk. Ezt úgy tessük meg, hogy két objektumot használunk a mondat minden kifejezésének szemantikai értelmezésénél: egyet arra az objektumra, amelyre a kifejezés szó szerint vonatkozik (Chrysler), míg egy másikkal utalunk a fogalomcserére (szóvivő). Ezek után meg kell mondanunk, hogy összefüggés van a kettő között. Jelenlegi nyelvtanunkban a „Chrysler announced” értelmezése a következő:

$$\exists x, e \quad x = \text{Chrysler} \wedge e \in \text{Announce}(x) \wedge \text{After}(\text{Now}, e)$$

Ezt a következőre kell megváltoztatnunk:

$$\exists m, x, e \quad x = \text{Chrysler} \wedge e \in \text{Announce}(m) \wedge \text{After}(\text{Now}, e) \wedge \text{Metonymy}(m, x)$$

E szerint van egy olyan  $x$  entitás, amely azonos a Chryslerrel, és van egy másik  $m$  entitás, amelyik a bejelentést tette, és a kettő metonímiai relációban áll egymással. A következő lépés annak meghatározása, hogy milyen metonímiai relációk fordulhatnak elő. A legegyszerűbb eset az, amikor nincs semmilyen metonímia – az  $x$  szó szerinti objektum és az  $m$  metonímiai objektum azonos:

$$\forall m, x \quad (m = x) \Rightarrow \text{Metonymy}(m, x)$$

A Chrysler példánál helyénvaló általánosítás az, hogy egy vállalat állhat a szóvivője helyén:

$$\forall m, x \quad x \in \text{Organizations} \wedge \text{Spokesperson}(m, x) \Rightarrow \text{Metonymy}(m, x)$$

További fogalomcserék lehetnek a szerző a munkákra (I read *Shakespeare*), vagy általában a létrehozó az alkotásra (I drive a *Honda*) és a rész az egészre (The Red Sox need a strong *arm*). A metonímia egyes példái, mint a „The ham sandwich on Table 4 wants another beer” újszerűbbek, és az adott szituációban megfelelően értelmezik őket.

Az itt vázolt szabályok lehetővé teszik a számunkra, hogy a „Chrysler announced a new model” mondat magyarázatát elkészítük, de a magyarázat nem logikai következetettsé útján jön létre. Valószínűségi vagy nemmonoton következetést kell használnunk ahhoz, hogy magyarázatjelöltekkel állunk elő.

Egy **metafora** (*metaphor*) olyan jelképes beszéd, amikor egy adott szó szerinti jelentéssel bíró kifejezést használunk egy másik jelentés sugalmazására analógia segítségével.

ségevel. A legtöbb ember úgy gondol a metaforára, mint a költők eszközére, mely nem játszik nagy szerepet a minden nap szövegekben. Mindazonáltal nagyon sok alapvető metaforát olyan gyakran használunk, hogy még azt sem vesszük észre, hogy azok metaforák. Egy ilyen metafora annak a gondolata, hogy a „több felfelé van”. Ez a metafora lehetővé teszi számunkra annak közlését, hogy az árak emelkedtek, felmásztak vagy rakéta módjára szárnyalnak, hogy a hőmérséklet csökken vagy esik, hogy a bázalom valakiben csökken, illetve hogy egy sztár népszerűsége megugrott vagy szárnyal.

Az ilyen metaforák megközelítésének két módja lehetséges. Az egyik az, hogy a metafora teljes tudását beépítjük a szótárba – új jelentéseket adunk az emelkedik, esik, mászik stb. szavakhoz, melyek úgy írják le őket, hogy mindenféle mértékre vonatkozhatnak, nem csak a magasságra. Ez a módszer sok alkalmazás számára elegendő, de nem ragadja meg a metafora azon generálóképességét, amely az emberek számára lehetővé teszi új kifejezések használatát – mint például az „alábukik” vagy a „kirobbantja a tetőt” – anélkül hogy félreértestől tartanának. A másik módszer a gyakori metaforákról explicit tudást illeszt be a rendszerbe, és ezeket használja az új kifejezések olvasás közbeni megértéséhez. Például tegyük fel, hogy egy rendszer ismeri a „több felfelé van” metaforáját. Azaz tudja, hogy az olyan logikai kifejezések, melyek egy függőleges skálán értelmezett pontra vonatkoznak, értelmezhetők egy mennyiségi skála megfelelő pontjaira vonatkozóként. Ezek után az „eladások magasak” kifejezéshez tartozna egy *Altitude(Sales, High)* szó szerinti értelmezés, amit metafora segítségével *Quantity(Sales, Much)*-ként értelmezhetünk.

## A többértelműség feloldása



Ahogy korábban mondtuk, a többértelműség feloldása *diagnózis kérdése*. A beszélő szándéka a kommunikációra a megnyilatkozásban szereplő szavak nem megfigyelt oka, és a hallgató dolga, hogy a szavakból és a szituációról való tudásából kiindulva visszafelé előállítsa a beszélő legvalószínűbb szándékát. Más szavakkal, a hallgató a következőt oldja meg:

$$\underset{\text{intent}}{\operatorname{argmax}} \text{ Likelihood}(\text{intent} \mid \text{words, situation})$$

ahol a *Likelihood* lehet valószínűség vagy a preferenciák bármilyen számszerű mértéke. Valamelyen preferenciára szükség van, mivel a szintaktikai és szemantikai értelmezési szabályok önmagukban nem tudnak a kifejezéshez vagy mondathoz egyetlen helyes értelmezést meghatározni. Ezért részekre bontjuk a munkát: a szintaktikai és szemantikai értelmezés felelős a lehetséges értelmezések halmazának előállításáért, a többértelműség feloldásának folyamata pedig kiválasztja a legjobbat.

Vegyük észre, hogy a szólásaktus szándékáról, és nem csak a beszélő által kimondott aktuális kijelentésről beszélünk. Például ha azt halljuk egy politikustól, hogy „I am not a crook”, akkor lehet, hogy csak 50% valószínűséget adunk annak, hogy a politikus nem bűnöző, ugyanakkor 99,999%-ot annak, hogy a beszélő nem egy kampós pástortör (a *crook* másik jelentése – *a ford.*). Mégis nagyobb valószínűséget rendelünk ahhoz az értelmezéshez, hogy

$$\text{Assert}(\text{Speaker}, \neg(\text{Speaker} \in \text{Criminals}))$$

mivel ez a valószínűbb dolog, amit mond.

Vizsgáljuk ismét az „I smelled a wumpus in 2,2.” többértelmű példát! Egy preferenciaheurisztika a jobbra asszociáció (*right association*), amely szerint amikor el kell

döntenünk, hogy az „in 2,2” PP-t hova helyezzük el az elemzési fában, akkor azt preferáljuk, hogy a jobb szélen elhelyezkedő összetevőhöz csatoljuk, ami jelen esetben az „a wumpus” NP. Természetesen ez csak egy heurisztika; az „I smelled a wumpus with my nose” mondatra a heurisztikát felülbírálja az a tény, hogy az „a wumpus with my nose” NP nem valószínű.

A többérműség feloldása a tények kombinálásával oldható meg, felhasználva a könyvben eddig látott tudásábrázolásra és bizonytalanság melletti következtetésre alkalmas minden technikát.

A tudást négy modellre bonthatjuk:

1. **A világ modell (world model):** annak valószínűsége, hogy egy állítás előfordul a világban.
2. **A mentális modell (mental model):** annak valószínűsége, hogy a beszélő létrehozza azt a szándékát, hogy egy adott tényt a hallgatóval közöljön, feltéve, hogy az bekövetkezik. Ez annak modelljét kombinálja, amit a beszélő hisz azzal, amit a beszélő hisz arról, amit a hallgató hisz és így tovább.
3. **A nyelvi modell (language model):** annak valószínűsége, hogy szavak egy bizonyos füzére lesz kiválasztva, azt feltételezve, hogy a beszélőben létezik egy adott tény elmondásának szándéka. A fejezetben ismertetett CFG- és DCG-modellek a valószínűségre logikai (igaz/hamis) modellt használnak: egy füzérnek vagy lehet egy adott értelmezése, vagy nem. A következő fejezetben a CFG olyan valószínűségi változatát látjuk majd, amely a többérműség feloldására egy jobban informált nyelvi modellt készít.
4. **Az akusztikai modell (acoustic model):** annak valószínűsége, hogy hangok egy adott sorozata keletkezik azt feltéve, hogy a beszélő szavak egy adott sorozatát választotta. A beszédfelismerést a 15.6. alfejezet tárgyalta.

## 22.7. SZÖVEGÉRTÉS

Egy szöveg (*discourse*) a nyelv tetszőleges karaktersorozata – tipikusan olyan, amelyik hosszabb, mint egy mondat. A jegyzetek, a regények, az időjárás-jelentések és a párbeszédek mind szövegek. Eddig teljesen figyelmen kívül hagytuk a szövegek problémáit, a nyelv önálló mondatokra bontását helyeztük előtérbe, melyeket *in vitro* tanulmányozhattunk. Ez a fejezet a mondatokat természetes környezetükben tanulmányozza. Két részproblémát fogunk vizsgálni: az utalásfeloldást és a koherenciát.

### Utalásfeloldás

Az **utalásfeloldás (reference resolution)** egy olyan névmás vagy határozott fónévi ki-fejezés értelmezése, amely a világ egy objektumára utal.<sup>16</sup> A feloldás a vilagról szóló tudáson és a szöveg korábbi részein alapszik. Vizsgáljuk a következő szakaszt:

<sup>16</sup> A nyelvészetben az utalást egy olyan dologra, ami már szerepelt, anaforának (*anaphoric reference*) nevezik. Az utalást egy később szereplő dologra kataforának (*cataphoric reference*) nevezik, mint például a „he” névnás a „When he won his first tournament, Tiger was 20”.

„John flagged down the waiter. He ordered a ham sandwich.”

Annak megértéséhez, hogy a „he” a második mondatban Johnra utal, meg kellett értenünk, hogy az első mondat két személyt említ, valamint azt, hogy John játssza a vendég szerepét, és ezért valószínűleg rendel, míg a pincér nem. Általában az utalásfeloldás egy hivatkozott kiválasztása a jelöltek listájáról, de néha magában foglalja új jelöltek létrehozását is. Vegyük például a következő mondatot:

„After John proposed to Marsha, they found a preacher and got married. For the honeymoon, they went to Hawaii. (Miután John megkérte Marshát, találtak egy papot, és összeházasodtak. Hawaiira mentek nászútra.)”

Itt a „honeymoon” határozott főnévi kifejezés egy olyan dologra utal, amire csak implicit módon utal a „married” ige. A „they” névmás egy olyan csoportra utal, amit explicit módon nem említettek előtte: John és Marsha (de *nem* a pap).

A legjobb hivatkozott dolog kiválasztása egy olyan többérműség feloldási folyamat, amely többféle szintaktikai, szemantikai és pragmatikus értelmezési információ kombinálásán alapszik. Bizonyos nyomravezetők kényszerek formájában adottak. Például a névmásoknak nemben és számosságban egyezniük kell a hivatkozottakkal: a „he” például utalhat Johnra, de nem Marshára; a „they” utalhat egy csoportra, de nem egyetlen személyre. A névmásoknak a reflexivitás szintaktikai kényszereinek is meg kell felelniük. Például a „he saw him in the mirror” mondatban a két névmásnak két különböző emberre kell vonatkoznia, míg a „he saw himself” kifejezésben ugyanarra az emberre kell utalniuk. Vannak a szemantikai konzisztenciára vonatkozó kényszerek is. A „he ate it” kifejezésben a „he” névmásnak olyan dologra kell utalnia, ami eszik, míg az „it” olyanra, amit meghetnek.

Bizonyos vezérfonalak olyan preferenciák, amik nem állnak mindenkor mindenkor előnyös, ha a névmási utalások követik ezt a struktúrát. Így a

„Marsha flew to San Francisco from New York. John flew there from Boston.”

mondatokban azt preferáljuk, hogy a „there” San Franciscóra utal, mivel az ugyanazt a szintaktikai szerepet tölti be. Párhuzamos struktúrája van, olyankor előnyös, ha a névmási utalások követik ezt a struktúrát. Így a

„Marsha gave Sally the homework assignment. Then she left.”

szövegben „Marsha” az első mondat alanya, így ő a „she” preferált előzménye. Egy másik preferencia annak az entitásnak az előtérbe helyezése, amelynek a tárgyalása a legnyilvánvalóbb. Ha a következő mondatpárt önállóan vizsgáljuk

„Dana dropped the cup on the plate. It broke.”

akkor felmerül egy probléma: nem világos, hogy az „it” a pohárra vagy a tányéra utal. Azonban egy nagyobb kontextusban a többérműség feloldódik:

„Dana was quite fond of the blue cup. The cup had been a present from a close friend. Unfortunately, one day while setting the table, Dana dropped the cup on the plate. It broke.”

Itt a pohár van a figyelem középpontjában, ezért ez a preferált hivatkozott dolog.

Sokféle utalásfeloldó algoritmust találtak ki. Az egyik első (Hobbs, 1978) figyelemre méltó, mivel egy olyan mértékű statisztikai ellenőrzést végeztek rajta, ami abban az időben szokatlan volt. Három különböző szövegfajtát használva Hobbs 92%-os pontosságot mutatott ki. A módszer azt feltételezte, hogy egy elemző egy helyes elemzést készített, ami azonban nem állt rendelkezésére, ezért Hobbs kézzel készítette el az elemzéseket. A Hobbs algoritmus keresésként működik: az aktuális mondattól indulva visszafelé keres a mondatokban. Ez a technika biztosítja, hogy a legutóbbi jelölteket vizsgálja először. Egy mondaton belül szélességi keresést végez, balról jobbra haladva. Ez azt biztosítja, hogy az alanyakat a tárgyak előtt vizsgálja. Az algoritmus kiválasztja az első jelöltet, amely a most vázolt kényszereknek eleget tesz.

## Egy koherens szöveg struktúrája

Nyissa fel ezt a könyvet tízszer véletlenszerűen, és másolja le az első mondatot minden oldalról. Az eredmény szükségszerűen inkoherens lesz. Hasonlóképpen, ha veszünk egy koherens, tiz mondatból álló szöveget, majd permutáljuk a mondatokat, az eredmény inkoherens lesz. Ez azt demonstrálja, hogy a természetes nyelvű szövegek mondati nagyon különböznek a logika állításaitól. A logikában, bármilyen sorrendben is KÖZÖLJÜK az  $A$ ,  $B$ , és  $C$  állításokat egy tudásbázissal, az  $A \wedge B \wedge C$  konjukciót kapjuk. Természetes nyelvben a mondatok sorrendje számít; vegyük például a „Go two blocks. Turn right” és a „Turn right. Go two blocks” szövegek közötti különbséget.

A szövegek mondatok feletti struktúrával rendelkeznek. Ezt a struktúrát a szöveg következő nyelvtanának segítségével vizsgálhatjuk meg:

$$\text{Segment}(x) \rightarrow S(x)$$

$$\text{Segment}(\text{CoherenceRelation}(x, y)) \rightarrow \text{Segment}(x) \text{ Segment}(y)$$

E nyelvtan szerint a szöveg szegmensekből áll, ahol minden szegmens vagy egy mondat, vagy mondatok egy csoportja, és a szegmenseket **koherenciarelációk (coherence relations)** kötik össze. A „Go two blocks. Turn right” szövegen a koherenciareláció az, hogy az első mondat engedélyezi a másodikat: a hallgatónak akkor kell jobbra fordulnia, miután ment két háztömbnyit. Különböző kutatók koherenciarelációk különböző készletét javasolták; a 22.21. ábra felsorol egy reprezentatív halmazt. Most vizsgáljuk még a következő történetet:

- (1) A funny thing happened yesterday.
- (2) John went to a fancy restaurant.
- (3) He ordered the duck.
- (4) The bill came to \$50.
- (5) John got a shock when he realized he had no money.
- (6) He had left his wallet at home.
- (7) The waiter said it was all right to pay later.
- (8) He was very embarrassed by his forgetfulness.

Itt, az (1) mondat az *Evaluation* relációban áll a szöveg hátralevő részével; (1) a beszélő megjegyzése a szöveghez. A (2) mondat engedélyezi a (3)-at, és a (2–3) párt együtt

- **Lehetővé teszi vagy okozza:**  $S_1$  előidéz egy olyan állapotváltozást (amely implicit is lehet), amely okozza, vagy lehetővé teszi  $S_2$ -t. Példa: „Kimentem az iskolába vezettem.” (A kimenetel implicit módon lehetővé teszi, hogy beüljünk az autóba.)
- **Magyarázat:** a lehetővé térellel ellentettje:  $S_2$  okozza vagy lehetővé teszi  $S_1$ -et, és így egy magyarázat rát. Példa: „Elkéstem az iskolából. Elaludtam.”
- **Háttér:**  $S_1$  leír egy szituációt vagy háttérét  $S_2$  számára. Példa: „Sötét és viharos éjszaka volt. A történet folytatódik...”
- **Kiértékelés:**  $S_2$ -ből következik, hogy  $S_1$  része a beszélő azon tervének, hogy a szövegrészletet szólásaktusként végrehajtsa. Példa: „Egy mulatságos dolog történt. A történet folytatódik...”
- **Szemléltetés:**  $S_2$  az  $S_1$ -ben levő általános dolog egy példája. Példa: „Az algoritmus megfordít egy listát. Az [A, B, C] bemenetet leképezi a [C, B, A]-ra.”
- **Általánosítás:**  $S_1$  az  $S_2$ -ben levő általános dolog egy példája. Példa: „Az [A, B, C] bemenetet leképezi a [C, B, A]-ra. Általánosságban ez az algoritmus megfordít egy listát.”
- **Elvárással ellentétes:**  $S_2$ -ből  $\neg P$ -t következteti ki, negálva az  $S_1$ -ből szokásos  $P$  következetet. Példa: „Ez a cikk gyenge. Másrészt viszont érdekes.”

**22.21. ábra.** Koherenciarelációk listája (Hobbs, 1990) alapján. Mindegyik reláció két, egymást követő szövegrészlet,  $S_1$  és  $S_2$  között áll fenn.

okozza (4)-et, azzal az implicit köztes állapottal, hogy János megette a kacsát. Ezek után (2–4) szolgál a szöveg hátralevő részének alapjául. A (6) mondat az (5) magyarázata, és (5–6) engedélyezi (7)-et. Vegyük észre, hogy ez egy *Enable*, nem *Cause*, mivel a pincérnek lehetett volna más reakciója is. Az (5–7) együtt okozza (8)-at. A 22.13. feladat arra kéri majd, hogy rajzolja fel e szöveg elemzési fáját.

A koherenciarelációk szolgálnak egy szöveg összekötésére. Vezetik a beszélőt annak eldöntésében, hogy mit mondjon és mit tekintsen magától értetődőnek, és segítik a hallgatót a beszélő szándékának megértésében. A koherenciarelációk a mondatok többértelműségének szűrőjeként szolgálhatnak: önállóan a mondatok lehetnek többértelműek, de ezen többértelmű értelmezések többsége nem illeszthető össze egy koherens szöveggé.

Eddig az utalásfeloldást és a szövegstruktúrát külön vizsgáltuk. Azonban a kettő valójában összefügg. Grosz és Sidner (1986) elmélete például azzal foglalkozik, hogy a beszélő és hallgató figyelme mire koncentrál a szöveg során. Elméletük tartalmaz egy vermet, amely **fókusztereket (focus spaces)** tartalmaz. Bizonyos megnyilvánulások a fókusz elmozdulását okozzák egy elem verembe helyezésével vagy kivételevel. Például az éttermi történetben a „John went to a fancy restaurant” mondat új fókuszt helyez a verem tetejére. Ezen a fókuszon belül a beszélő egy határozott *NP*-t használhat a „the waiter”-re történő utalásra (az „a waiter” általános *NP* helyett). Ha a történet folytatódna egy „John went home” mondattal, akkor a fókuszteret kivenné a verem tetejéről, és a szöveg nem utalhatna tovább a pincérre a „the waiter” vagy a „he” kifejezésekkel.

## 22.8. A NYELVTAN INDUKCIÓS TANULÁSA

A nyelvtan indukciós (grammar induction) tanulása annak feladata, hogy adatokból nyelvtant tanulunk. Nyilvánvaló, hogy meg kell kísérelni, hiszen bebizonysodott, hogy nagyon nehéz egy nyelvtant kézzel elkészíteni, és az interneten mintaként használható kijelentések milliárdjai állnak ingyenesen rendelkezésre. Ez egy nehéz feladat, mivel a lehetséges nyelvtanok tere végtelen, és azért is, mivel annak ellenőrzése, hogy egy adott nyelvtan generálja-e a mondatok egy adott halmazát, számításigényes feladat.

Egy érdekes modell a SEQUITUR-rendszer (Nevill-Manning és Witten, 1997). Nincs szüksége más bemenetre, csak egy szövegre (amit nem szükséges előzetesen mondatokra bontani). Egy nagyon specializált formátumú nyelvtant állít elő: egy olyan nyelvtant, amely csak egy karakterfüzérét generál, nevezetesen az eredeti szöveget. Másképp tekintve a SEQUITUR épp csak arra elegendő nyelvtant tanul, hogy a szöveget elemesse. Íme, itt van az a zárójelezés, amit egy nagyobb, hírekkel tartalmazó szövegen levő mondatra felfedezett:

[Most Labour] [sentiment [[would still] [favor the] abolition]] [[of [the House]] [of Lords]]

Helyesen választott ki olyan összetevőket, mint például az „of the House of Lords” PP, bár a tradicionális elemzéssel ellentétesen dolgozik akkor például, amikor a „the”-t a megelőző igével és nem a követő fónévvvel csoportosítja.

A SEQUITUR azon az ötleten alapszik, hogy egy jó nyelvtan egyben tömör is. A következő két kényszer betartása a legfontosabb: (1) Egymást követő szimbólumok egyetlen párra sem szerepelhet egynél többször a nyelvtanban. Ha az A B szimbólumpár több szabály jobb oldalán is szerepel, akkor ki kell cserélnünk a pár egy új, nem záró szimbólummal, amit C-nek nevezünk, és egy új szabályt kell bevezetnünk, miszerint C → A B. (2) minden szabályt legalább kétszer kell használni. Ha egy C nem záró, csak egyszer szerepel a nyelvtanban, akkor törölünk kell a C-re vonatkozó szabályt, és fel kell cserélnünk egyetlen előfordulását a szabály jobb oldalával. Ezt a két kényszert egy mohó keresés alkalmazza, amely a beérkező szöveget balról jobbra végignézi, menet közben inkrementálisan építi a nyelvtant, és alkalmazza a kényszereket, amint lehetséges. A 22.22. ábra mutatja az algoritmus működését az „abcdabcde” szövegen. Az algoritmus egy optimálisan tömör nyelvtant határoz meg a szöveghez.

A következő fejezetben további nyelvtant tanuló algoritmusokat is látni fogunk, amelyek valószínűségi alapú, környezetfüggetlen nyelvtanokkal működnek. Most azonban egy olyan nyelvtan tanulásának problémáját vizsgáljuk meg, amelyet szemantikával terjesztünk ki. Mivel egy kiterjesztett nyelvtan egyben Horn-kláz logikai program is, ezért az induktív logikai programozás technikái megfelelők lesznek. A CHILL (Zelle és Mooney, 1996) egy induktív logikai programozás (ILP), amely példákból megtanul egy nyelvtant és egy arra specializált elemzőt. A célterület természetes nyelvű adatbázis-lekérdezések. A tanító példák szófüzérek, és a nekik megfelelő lekérdezések párait tartalmazzák – például:

What is the capital of the state with the largest population?

*Answer(c, Capital(s, c) ∧ Largest(p, State(s) ∧ Population(s, p)))*

A CHILL feladata, hogy megtanuljon egy Parse(words, query) predikátumot, amely konzisztens a példákkal, és remélhetőleg más példákra is jól általánosít. Az ILP közvet-

Bemenet	Nyelvtan	Megjegyzések
1 <i>a</i>	$S \rightarrow a$	
2 <i>ab</i>	$S \rightarrow ab$	
3 <i>abc</i>	$S \rightarrow abc$	
4 <i>abcd</i>	$S \rightarrow abcd$	
5 <i>abcdb</i>	$S \rightarrow abcdb$	
6 <i>abcdbc</i>	$S \rightarrow abcdcb$ $S \rightarrow aAdA; A \rightarrow bc$	<i>bc</i> kétszer
7 <i>abcdcba</i>	$S \rightarrow aAdAa; A \rightarrow bc$	
8 <i>abcdcab</i>	$S \rightarrow aAdAab; A \rightarrow bc$	
9 <i>abcdbcabc</i>	$S \rightarrow aAdAabc; A \rightarrow bc$ $S \rightarrow aAdAaA; A \rightarrow bc$ $S \rightarrow BdAB; A \rightarrow bc; B \rightarrow aA$	<i>bc</i> kétszer <i>aA</i> kétszer
10 <i>abcdbcabed</i>	$S \rightarrow BdABd; A \rightarrow bc; B \rightarrow aA$ $S \rightarrow CAC; A \rightarrow bc; B \rightarrow aA; C \rightarrow Bd$ $S \rightarrow CAC; A \rightarrow bc; C \rightarrow aAd$	<i>Bd</i> kétszer <i>B</i> csak egyszer

**22.22. ábra.** Az „*abcdbcabed*” bemeneti szöveget elemző nyelvtant tartalmazó SEQURUR-rendszer futási lépései. Egy *S*-re vonatkozó szabályból indulunk ki, és ezen szabály végéhez sorban minden egyes szimbólumot hozzáadunk. Miután a hatodik szimbólumot is hozzáadtuk, megkapjuk egy ismétlődő pár első előfordulását: *bc*. Ezért a *bc* minden két előfordulását lecseréljük egy új nem záróra, *A*-ra, és egy új szabályt veszünk fel: *A* → *bc*. Miután három újabb szimbólumot hozzáraktunk a bemenethez, a kilencedik a *bc* egy újabb ismétlését eredményezi, így újra *A*-val helyettesítjük. Ez az *aA* két előfordulásához vezet, ezért egy új nem záróval, *B*-vel helyettesítjük. Miután a tizedik, utolsó szimbólumot is hozzáraktuk, *Bd* két előfordulását kapjuk, ezér lecseréljük őket egy új nem záróval, *C*-vel. De *B* immár csak egyszer szerepel, a *C* szabály jobb oldalán, ezért *B*-t a neki megfelelő bővítéssel, *aA*-val cseréljük le.

len alkalmazása ezen predikátum megtanulására gyenge teljesítményt hoz: a kikövetkeztetett elemző csak körülbelül 20%-os pontosságú. Szerencsére az ILP tanuló algoritmusok fejleszthetők tudás bevitelével. Ebben az esetben a Parse predikátum nagy részét logikai programként definiálták, és a CHILL feladatát a vezérlő szabályok kikövetkeztetésére redukálták, amelyek segítik az elemzőt a különböző elemzések közötti választásban. Ezzel a hozzáadott háttértudással a CHILL 70–85% közötti pontosságot ér el különböző adatbázis-lekérdezési feladatokon.

## 22.9. ÖSSZEFoglalás

A természetes nyelv megértése az MI egyik legfontosabb részterülete. Ötleteket merít a filozófiából és a nyelvészettől, valamint a logikai programozás, valóságúniségi tudás-reprezentáció és a következtetés területének technikáiból is. Más MI-területektől eltérő módon a természetes nyelv megértése a tényleges emberi viselkedés empirikus vizsgálatát igényli – amely komplexnek és érdekesnek bizonyul.

- Az ágensek jelzéseket küldenek egymásnak bizonyos célok elérése érdekében: informálni, figyelmeztetni, segítséget szerezni, tudást megosztani vagy megígérni valamit.

A jelzések ily módon való küldését **szólásaktusnak (speech act)** nevezik. Végső soron minden szólásaktus kísérlet arra, hogy egy másik ágens elhiggyen vagy megtegyen valamit.

- A nyelv megállapodáson alapuló **jelzések ből (signs)** áll, melyek jelentést közvetítenek. Sok állat használ ilyen értelemben jelzéseket. Az ember tűnik az egyetlen állatnak, aki **nyelvtant (grammar)** használ strukturált üzenetek végételen számú variációinak előállítására.
- A kommunikáció három lépést igényel a beszélőtől, a szándékot egy elképzelés átadására, a szavak mentális előállítását és a szavak fizikai szintézisét. Ezek után a hallgatónak négy lépést kell végrehajtania: észlelés, analízis, többértelműség feloldása és a jelentés beépítése. minden nyelvhasználat **beágyazott (situated)** abban az értelemben, hogy a megnyilatkozások jelentése függhet attól a szituációból, amiben előállnak.
- A formális nyelvészeti és a **kifejezésstruktúra (phrase structure)** nyelvtanok (külnösen a **környezetfüggetlen nyelvtan (context-free grammar)**) hasznos eszközök a természetes nyelv egyes aspektusainak kezelésére.
- Környezetfüggetlen nyelv mondatai  $O(n^3)$  időben elemezhetők a **diagramelemző (chart parser)** segítségével.
- Kézenfekvő egy nyelvtan **kiterjesztése (augment)** annak érdekében, hogy olyan problémákat kezeljünk, mint az alany-ige egyeztetése vagy a névmási esetek. A **definit kláz nyelvtan (definite clause grammar, DCG)** egy olyan formalizmus, amely lehetővé teszi a kiterjesztéseket. A DCG segítségével az elemzés és szemantikai értelmezés (sőt még a generálás is) logikai következtetés segítségével elvégezhető.
- A **szemantikus értelmezés (semantic interpretation)** szintén kezelhető egy kiterjesztett nyelvtannal. Egy kvázilogikai forma jó közvetítő lehet a szintaktikai fák és a szemantika között.
- A **többértelműség (ambiguity)** nagyon fontos probléma a természetes nyelv megértésében; a mondatok többségének több lehetséges értelmezése van, de általában csak egy megfelelő. A többértelműség feloldása a vilagról, az adott szituációról és a használt nyelvről szóló tudáson alapszik.
- A legtöbb nyelv több mondat kontextusában létezik, nem csak egyben. A **szövegértes (discourse)** a kapcsolódó szövegek vizsgálata. Láttuk, hogyan oldunk fel mondatok közötti névmási utalásokat, és a mondatok hogyan kapcsolódnak koherens szegmensekké.
- A **nyelvtan indukciós (grammar induction)** tanulása példákból tud nyelvtant tanulni, bár vannak határai annak, hogy a nyelvtan milyen jól általánosít.

## Irodalmi és történeti megjegyzések

A jelek és szimbólumok mint nyelvi elemek tanulmányozását John Locke (1690) **jeltudománynak (semiotics)** nevezte, bár a 20. századig nem művelték (Peirce, 1902; De Saussure, 1993). A legutóbbi áttekintő munkák között van Eco (Eco, 1979) és Cobley (Cobley, 1997) műve.

A nyelv mint cselekvés ötlete a 20. századi nyelvészeti beállítottságú filozófiából ered (Wittgenstein, 1953; Grice, 1957; Austin, 1962), leginkább a *Speech Acts* c. könyvből (Searle, 1969). A szólásaktus ötletének előfutára Protagorasz négyféle mondatkategóriája

volt: ima, kérdés, válasz és ítélet (kb. i. e. 430-ból). A szólisták tervalapú modelljét először Cohen és Perrault (Cohen és Perrault, 1979) javasolta. A nyelv cselekvéshez kapcsolását tervfelismerés segítségével történetek megértéséhez Wilensky tanulmányozta (Wilensky, 1983). Cohen, Morgan és Pollack (Cohen, Morgan és Pollack, 1990) gyűjtött össze újabb munkákat e területen.

A szemantikus hálóhoz hasonlóan a környezetfüggetlen nyelvtanok (amelyek kifejezősstruktúra nyelvtanokként is ismertek) a sásztra szanszkrit nyelvet tanulmányozó ósi indiai nyelvészek (külsőnösen Pánini, kb. i. e. 350) által használt technika felelevenítései (Ingerman, 1967). A modern időkben Noam Chomsky fedezte fel újra őket az angol szintaxis elemzésére (Chomsky, 1956), és tőle függetlenül John Backus az Algol-58 szintaxis analízisére használta. Naur kiterjesztette Backus jelölésrendszerét (Naur, 1963), ma pedig őhozzá rendelik az „N” betűt a BNF-ben, ami eredetileg a Backus Normál Forma volt (Backus Normal Form) (Backus, 1996). Knuth egyfajta kiterjesztett nyelvtant definiált, amelyet attribútumnyelvtannak (**attribute grammar**) nevezünk, és programozási nyelvek esetén hasznos (Knuth, 1968). A definit klóz nyelvtanokat Colmerauer vezette be (Colmerauer, 1975), majd Pereira és Warren fejlesztette tovább és népszerűsítette (Pereira és Warren, 1980). A Prolog programozási nyelvet Alain Colmerauer talált ki, elsősorban a francia nyelv elemzésének problémájára. Colmerauer voltaképpen kidolgozott egy metamorfózis-nyelvtannak (metamorphosis grammar) nevezett formalizmust, amely tövből a definit klázoknál, de a DCG hamarosan követte.

Számos kísérlet volt természetes nyelvek formális nyelvtanának leírására, mind a „tiszta”, mind a számítógépes nyelvészethez. A gépek számára készült nyelvtanok között van a New York University Linguistic String projektje (Sager, 1981) és a University of Pennsylvania XTAG projektje (Doran és társai, 1994). A modern DCG-rendszerek jó példája a Core Language Engine (Alshawi, 1992). Több átfogó, de nem formális nyelvtan létezik az angol nyelvről (Jespersen, 1965; Quirk és társai, 1985; McCawley, 1998; Huddleston és Pullum, 2002). A nyelvészetről szóló jó jegyzetek között van a (Sag és Wasow, 1999) bevezetése a szintaxisba, valamint a (Chierchia és McConnell-Ginet, 1990; Heim és Kratzer, 1998) szemantikáról szóló jegyzetek. McCawley nyelvészük számára szóló anyaga a logikára koncentrál (McCawley, 1993).

A nyolcvanas évek közepétől létező trend szerint egyre több információt tesznek a szókincsbe és kevesebbet a nyelvtanba. A lexikai-funkcionális nyelvtan (lexical-functional grammar), avagy LFG (Bresnan, 1982) volt az első jelentős nyelvtani formalizmus, amihez nagy szókincset gyűjtötték. Ha a szélsőséggel fokozzuk a lexikálizálást, akkor a kategorikus nyelvtant (**categorial grammar**) kapjuk, amelyben lehet, hogy csak két nyelvtani szabály van, vagy a függőségi nyelvtant (**dependency grammar**) (Mel'čuk és Polguere, 1988), amelyben nincsenek kifejezések/frázsik, csak szavak. Sleator és Temperley leír egy függőségi nyelvtant használó népszerű elemzőt (Sleator és Temperley, 1993). A TAG, avagy Tree-Adjoining Grammar (Joshi, 1985) nem szigorúan lexikális, de nő a népszerűsége lexikálizált alakban is (Schabes és társai, 1988). A Wordnet (Fellbaum, 2000) egy körülbelül 100 000 szót és kifejezést tartalmazó szabadon elérhető szótár, melyet a beszéd részei szerint kategorizáltak, és olyan szemantikai relációkkal egészítettek ki, mint például a szinonima, antónia és része.

Az első gépesített elemző algoritmusokat Yngve mutatta be (Yngve, 1955). Hatékony algoritmusokat a hatvanas évek végén fejlesztettek ki, néhány kisebb újítással azóta (Kasami, 1965; Younger, 1967; Graham és társai, 1980). A diagramelemzőnk Early

rendszeréhez áll a legközelebb (Early, 1970). Egy jó összefoglaló Aho és Ullman elemzésről és fordításról szóló tanulmányában található (Aho és Ullman, 1972). Maxwell és Kaplan mutatja meg, hogy a kiterjesztett diagramelemző algoritmus hogyan tehető hatékonyá az átlagos esetekben (Maxwell és Kaplan, 1993). Church és Patil foglalkozik a szintaktikai többérműség feloldásával (Church és Patil, 1982).

A természetes nyelvek formális szemantikai értelmezése a filozófiából és a formális logikából ered, és különösen szorosan kapcsolódik Alfred Tarskinak a formális nyelvek szemantikájáról szóló munkájához (Tarski, 1935). Bar-Hillel volt az első, aki a pragmatikai elemzés problémáját áttekintette, és azt javasolta, hogy formális logikával kezeljék. Például vezette C. S. Pierce *referenciális index (indexical)* fogalmát a nyelvészettel (Bar-Hillel, 1954). Richard Montague esszéje, az „English as a formal language” a nyelv logikai analízisének egyfajta manifestuma (Montague, 1970), azonban Dowty és társainak könyve (Dowty és társai, 1991), valamint Lewis cikke (Lewis, 1972) sokkal olvasmányosabb. Thomason szerkesztette Montague munkáinak teljes kollekcióját (Thomason, 1974). A mesterséges intelligenciában McAllester és Givan munkája folytatja a montague-i hagyományt sok új technikai részlet megvilágításával (McAllester és Givan, 1992).

Egy közbülső, vagy kvázilogikai forma ötlete a kvantorok érvényességi köréhez hasonló problémák kezelésére Woodsig nyúlik vissza (Woods, 1978), és sok jelenlegi rendszerben is megtalálható (Alshawi, 1992; Hwang és Schubert, 1993).

Az első valódi feladatot megoldó NLP-rendszer valószínűleg a BASEBALL kérdésválaszoló rendszer volt (Green és társai, 1961), amely baseball-statistikák adatbázisával kapcsolatos kérdéseket kezelt. Nem sokkal utána következett Woods LUNAR rendszere, amely az Apolló program által visszahozott kövekkel kapcsolatos kérdéseket válaszolt meg (Woods, 1973). Robert Schank és hallgatói több programot is készítettek (Schank és Abelson, 1977; Wilensky, 1978; Schank és Riesbeck, 1981; Dyer, 1983), amelyek mindegyikének a nyelv megértése volt a feladata. A hangsúly azonban kevésbé volt magán a nyelven, sokkal inkább a reprezentáció és következetesen. A problémák között volt a sztereotíp szituációk reprezentálása (Cullingford, 1981), az emberi memória-szervezés leírása (Rieger, 1976; Kolodner, 1983), valamint tervez és célok megértése (Wilensky, 1983).

A természetes nyelv generálását az ötvenes években, a gépi fordítás legelső napjaitól kezdve figyelembe vették, de a hetvenes évekig nem merült fel egynyelvű problémaként. Simmons és Slocum (Simmons és Slocum, 1972), valamint Goldman (Goldman, 1975) munkája reprezentatív. A PENMAN (Bateman és társai, 1989) volt az első teljes generáló rendszer, amely a Szisztematikus Nyelvtanra (Systematic Grammar) (Kasper, 1988) épült. A kilencvenes években két fontos szabadon elérhető generáló rendszer vált elérhetővé, a KPM (Bateman, 1997) és a FUF (Elhadad, 1993). A generálásról szóló fontos könyvek között szerepel (McKeown, 1985; Hovy, 1988; Patten, 1988; Reiter és Dale, 2000).

A többérműség feloldásával foglalkozó egyik legkorábbi munka Wilks elmélete a **preferenciaszemantikáról (preference semantics)** (Wilks, 1975), amelyik azokat az értelmezéseket próbálta megtalálni, amik minimalizálják a szemantikai anomáliák számát. Hirst egy olyan hasonló célú rendszert ír le, amelyik közelebb van a fejezetben ismertetett kompozíciós szemantikához (Hirst, 1987). Hobbs és társai a szintaktikai és szemantikai reprezentáció minőségét mérő kvantitatív keretrendszert írnak le (Hobbs és társai, 1993). Azóta a Bayes-hálók használata vált elterjedtebbé (Charniak és Goldman,

1992; Wu, 1993). A nyelvészetiben az optimalitás elmélete (Kager, 1999) a puha kényeszerek nyelvtanba építésén alapszik, amely természetes súlyt ad az értelmezéseknek ahelyett, hogy a nyelvtan az összes lehetőséget egyformára súlyralapozva generálná. Norvig tárgyalja azokat a problémákat, amelyek egy maximálisan valószínű értelmezés helyett több párhuzamos értelmezés figyelembevételéből adódnak (Norvig, 1988). Az irodalmi kritika nem egyértelmű a tekintetben, hogy a többértelműség megoldandó probléma vagy üdvözlendő dolog (Empson, 1953; Hobbs, 1990).

Nunberg a metonímia (metonymy) formális modelljét vázolja (Nunberg, 1979). Lakoff és Johnson az angol nyelv gyakori metaforáit katalogizálja és magával ragadó módon elemzi (Lakoff és Johnson, 1980). Ortony metaforáról szóló cikkgyűjteményt mutat be (Ortony, 1979); Martin a metafora értelmezésének számítógépes megközelítését adja meg (Martin, 1990).

Az utalásfeloldás általunk bemutatott kezelése a (Hobbs 1978)-at követi. A (Lappin és Leass, 1994) által bemutatott összetettebb megoldás egy kvantitatív pontozási módszeren alapszik. Újabb munkák (Kehler, 1997; Ge és társai, 1998) gépi tanulást használnak a kvantitatív paraméterek hangolására. Az utalásfeloldásról szóló két kitűnő áttekintés Hirst és Mitkov könyvei (Hirst, 1981; Mitkov, 2002).

1758-ban David Hume *Enquiry Concerning the Human Understanding* c. műve amellett érvelt, hogy a nyelvi szövegeket „három, az elgondolások közötti kapcsolatot leíró elv köti/fogja össze, nevezetesen a *Hasonlóság*, az idő- vagy térfelvéről *Összefüggés* és az *Ok* vagy *Okozat*”. Így kezdődött a koherenciarelációk meghatározásának hosszú története. Hobbs (Hobbs, 1990) adja meg nekünk a fejezetben használt halmazt; Mann és Thompson (Mann és Thompson, 1983) egy jobban kifejtett halmazt nyújt, amely magában foglalja a következőket: megoldás-összetartozás, bizonyíték, igazolás, motiváció, következetettség, sorozat, engedélyezés, kidolgozás, újrafogalmazás, feltétel, körülmény, ok, engedmény, háttér, valamint tézis-antitézis. Ez a modell fejlődött a retorikai struktúra elméletté (rhetorical structure theory, RST), amely valószínűleg napjaink legkiemelkedőbb elmélete (Mann és Thompson, 1988). Ez a fejezet átvesz egyes példákat (Jurafsky és Martin, 2000) Andrew Kehler által írt fejezetéből.

Grosz és Sidner bemutat egy fókuszteltoláson alapuló szövegkoherencia-elméletet (Grosz és Sidner, 1986), és a (Grosz és társai, 1995) egy középpontba állítására alapuló hasonló elméletet kínál. Joshi, Webber és Sag fontos korai munkákat gyűjt össze a szövegértésről (Joshi, Webber és Sag, 1981). Webber bemutatja olyan szintaxis- és szövegkényszerek egymással kapcsolatban álló modelljét, amely kényszerek a szöveg egy adott pontján állítható dolgokra vonatkoznak (Webber, 1983), továbbá bemutatja az igeidők és a szöveg kölcsönhatási módjának modelljét (Webber, 1988).

Az első fontos eredmény a nyelvtan indukciós (*grammar induction*) tanulásának területén egy negatív eredmény volt: Gold (Gold, 1967) megmutatta, hogy nem lehetséges megbiztosítani egy helyes környezetfüggetlen nyelvtan megtanulása annak szövegei alapján. Az ötlet lényege szerint ha adott egy  $s_1, s_2, \dots, s_n$  füzérhalmaz, akkor a helyes nyelvtan lehet a minden magába foglaló ( $S \rightarrow word^*$ ), vagy a bemenet másolata, az ( $S \rightarrow s_1 | s_2 | \dots | s_n$ ), vagy bármi a kettő között. Jeles nyelvészek, mint például Chomsky (Chomsky, 1957; 1980) és Pinker (Pinker, 1989; 2000) felhasználták Gold eredményeit annak megmutatására, hogy lennie kell egy öröklött univerzális nyelvtannak (**universal grammar**), amellyel minden gyermek születésénél fogva rendelkezik. Az úgynevezett Ingerszegénység (*Poverty of the Stimulus*) érvelés szerint a gyerekeknek nincs más nyelvi

példájuk, mint pozitív: szüleik és társaik nyelvük legtöbbször pontos példáit állítják elő, és igen ritkán javítanak ki hibákat. Ezért, mivel Gold bebizonyította, hogy pozitív példákból nem lehet CFG-t tanulni, a gyerekeknek már „tudniuk” kell a nyelvtant, és pusztán ezen öröklött nyelvtan paramétereit állítják, illetve a szókészletet tanulják. Bár ez az érvelés még befolyással bír a Chomskyt követő nyelvészek körében, egyes nyelvészek (Pullum, 1996; Elman és társai, 1997) és a legtöbb számú tudományos foglalkozó elvette. Igen hamar, már 1969-ben Horning megmutatta, hogy *meg lehet tanulni* PAC értelemben egy *valószínűségi* környezetfüggetlen nyelvtant. Azóta számos megyőző, kizárolag pozitív példákból tanuló empirikus demonstráció jelent meg, mint például Mooney, illetve Muggleton és De Raedt ILP-munkái (Mooney, 1999; Muggleton és De Raedt, 1994), valamint Schütze és de Marcken emlékezetes PhD-értekezései (Schütze, 1995; de Marcken, 1996). Más nyelvtani formalizmusok, mint például a reguláris nyelvek (Oncina és Garcia, 1992; Denis, 2001), reguláris fa nyelvek (Carrasco és társai, 1998), valamint a véges automaták (Parekh és Honavar, 2001) tanulása is lehetséges.

A SEQUITUR-rendszert Nevill-Manning és Witten készítette (Nevill-Manning és Witten, 1997). Érdekes, hogy ők, valamint de Marcken is megjegyezte, hogy nyelvtant kikövetkeztető megoldásaiak tömörítésre is jól használhatók. Ez összhangban van a minimális leíró hosszúság kódolásának elvével: egy jó nyelvtan olyan, amely minimalizál két méretet: a nyelvtan és a szövegek elemzési fájának nagyságát.

A nyelv tanulására szolgáló induktív logikai programozási munkák között van a CHILL rendszer (Zelle és Mooney, 1996), valamint Mooney és Califf programja (Mooney és Califf, 1995), amely jobban tanult igék múlt idejű alakjára vonatkozó szabályokat, mint a korábbi neurális hálók és a döntési fa rendszerek. Cussens és Džeroski több cikkgyűjteményt is szerkesztett, amelyek a nyelvtanulásról szóltak a logikában (Cussens és Džeroski, 2000).

Az Association for Computational Linguistics (ACL) rendszeresen rendez konferenciákat, és kiadja a *Computational Linguistics* folyóiratot. Ezenkívül van egy nemzetközi konferencia a témaiban, az International Conference on Computational Linguistics (COLING). A *Readings in Natural Language Processing* (Grosz és társai, 1986) a terület sok fontos korai cikkét tartalmazó antológia. A (Dale és társai, 2000) az NLP-rendszerek építésére szolgáló gyakorlati eszközökre helyezi a hangsúlyt. Jurafsky és Martin jegyzete alapos bevezetőt nyújt a területhez (Jurafsky és Martin, 2000). Az (Allen, 1995) egy kicsit régebbi munka. Pereira és Sheiber, valamint Covington Prolog-implementációkon alapuló tömör áttekintést ad (Pereira és Sheiber, 1987; Covington, 1994). Az *Encyclopedia of AI*-ben sok hasznos cikk található a területről; kiemelendő a „Computational Linguistics” és a „Natural Language Understanding”.

## Feladatok

### 22.1. Olvassa el az alábbi szöveget egyszer, és próbáljon meg amennyit csak lehet megjegyezni belőle. Később lesz egy teszt.

Az eljárás valójában igen egyszerű. Először különböző csoportokba rendezi a dolgokat. Természetesen egy halom elegendő lehet attól függően, hogy mennyi teendő van. Ha valahova máshova kell mennie a felszerelés hiánya miatt, akkor az a következő lépés, különben már nagyjából előkészült. Fontos, hogy ne vigye túlzásba a dolgot. Ez azt jelenti, hogy jobb egyszerre keveseb-

bel foglalkozni, mint többel. Rövid távon ez nem tűnhet fontosnak, de könnyen jelentkezhetnek komplikációk. A hiba is költséges. Első ránézésre a teljes eljárás bonyolultnak tűnik. Azonban hamarosan az élet megszokott dolgává válik. Nehéz a feladat szükségességének bármilyen elmúlását is előre látni, de sosem lehet tudni. Miután az eljárás befejeződött, ismét különböző csoportokba kell rendezni az anyagokat. Ezek után a megfelelő helyükre pakolhatók a dolgok. Végső soron újra használni fogják őket, és a teljes ciklust meg kell majd ismételni. Mindazonáltal, ez az élet velejárója.

- 22.2.** Írjon egy nyelvtant a DCG-jelölésrendszerrel, amely ugyanolyan, mint az  $\mathcal{E}_1$ , azaz a kivétellel, hogy kikénszeríti a mondat alanya és igéje közötti egyeztetést, így nem állítja elő az „I smells the wumpus”-t!
- 22.3.** Terjessze ki az  $\mathcal{E}_1$  nyelvtant úgy, hogy kezelje a névelő-főnév egyeztetéseket! Azaz biztosítsa, hogy az „agents” *NP*, de az „agent” és az „an agents” nem az!
- 22.4.** Foglalja össze a legfontosabb különbségeket a Java (vagy bármely más számítógépes nyelv, amelyet ismer) és a magyar között, megjegyezve a „megértés” problémáját mindenktőre! Gondoljon olyan dolgokra, mint a nyelvtan, a szintaxis, a szemantika, a pragmatikus elemzés, az összetételek, a környezetfüggőség, a lexikális többértelműség, a szintaktikai többértelműség, az utalások megtalálása (a névmásokat is ideérte), a háttértudás és leginkább arra, hogy mit is jelent „megérteni”.
- 22.5.** Melyek indokolják a következők közül a kvázilogikai forma bevezetését?
- Könnyebben leírható egyszerű összetétes nyelvtani szabályok.
  - A szemantikai leíró nyelv kifejezőképességének kiterjesztése.
  - A kvantorok érvényességi köre többértelműségének (többek között) tömör formában történő ábrázolása.
  - A szemantikai egyértelműsítés egyszerűbbé tétele.
- 22.6.** Határozza meg, hogy az ebben a fejezetben található nyelvtan alapján milyen szemantikai értelmezés adható a következő mondatokhoz:
- It is a wumpus.
  - The wumpus is dead.
  - The wumpus is in 2,2.
- Jó ötlet lenne az „It is a wumpus” szemantikai értelmezésének egyszerűen a  $\exists x \in Wumpuses$ ? Fontoljon meg alternatív mondatokat, mint például az „It was a wumpus”.
- 22.7.** Anélkül hogy megnézne a 22.1. feladatot, válaszolja meg a következő kérdéseket:
- Mi a négy említett lépés?
  - Melyik lépés maradt ki?
  - Mi az „az anyag”, amit a szöveg említi?
  - Milyen hiba lenne drága?
  - Jobb túl keveset, vagy túl sokat tenni? Miért?

- 22.8.** Ez a gyakorlat nagyon egyszerű nyelvek nyelvtanaival foglalkozik.
- Írjon egy környezetfüggetlen nyelvtant az  $a^n b^n$  nyelv számára!
  - Írjon egy környezetfüggetlen nyelvtant a tükrömondatok (palindroma) számára: az összes olyan füzérre, amelyek második fele épp az első fél tükrépe!
  - Írjon egy környezetfüggő nyelvtant a duplázó nyelv számára: az összes olyan füzérre, amelyek második fele megegyezik az első felével.

- 22.9.** Vizsgálja meg a „Someone walked slowly to the supermarket (Valaki lassan a szupermarkethöz sétált)” mondatot és az alábbi szókincset:

<i>Pronoun</i> → <b>someone</b>	<i>V</i> → <b>walked</b>
<i>Adv</i> → <b>slowly</b>	<i>Prep</i> → <b>to</b>
<i>Det</i> → <b>the</b>	<i>Noun</i> → <b>supermarket</b>

A következő átíró szabályhalmazok melyike állítja elő a fenti mondatot az előbbi szókincset használva? Adja meg a megfelelő elemzési fá(ka)t!

(A):	(B):	(C):
$S \rightarrow NP VP$	$S \rightarrow NP VP$	$S \rightarrow NP VP$
$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$
$NP \rightarrow Det Noun$	$NP \rightarrow Noun$	$NP \rightarrow Det NP$
$VP \rightarrow VP PP$	$NP \rightarrow Det NP$	$VP \rightarrow V Adv$
$VP \rightarrow VP Adv Adv$	$VP \rightarrow V Vmod$	$Adv \rightarrow Adv Adv$
$VP \rightarrow V$	$Vmod \rightarrow Adv Vmod$	$Adv \rightarrow PP$
$PP \rightarrow Prep NP$	$Vmod \rightarrow Adv$	$PP \rightarrow Prep NP$
$NP \rightarrow Noun$	$Adv \rightarrow PP$	$NP \rightarrow Noun$
	$PP \rightarrow Prep NP$	

Írjon három szintaktikailag helyes és három helytelen angol mondatot, amelyet a három előbbi nyelvtan állít elő! Lényegében különbözzenek egymástól, legalább hat szó hosszúak legyenek, és egy teljesen új szókincsre kell épülniük (amit lehetőleg Ön határozzon meg). Javasoljon minden hármon nyelvtanra olyan újítást, amely elkerüli a nem helyes mondatok előállítását!

- 22.10.** Valósítsa meg a diagramelemző algoritmus egy olyan változatát, amely a teljes  bemenetet lefedő összes él tömörített fáját adja vissza!

- 22.11.** Valósítsa meg a diagramelemző algoritmus egy olyan változatát, amely a leg-hosszabb bal oldali él tömörített fáját adja vissza, és amennyiben ez az él nem fedi le a teljes fát, folytatja az elemzést annak az élnek a végén! Mutassa meg, hogy miért lesz szükség a PREDICT eljárás meghívására a folytatás előtt! A vég-eredmény tömörített fák egy olyan listája, ahol a teljes lista lefedi a bemenetet.

- 22.12.** (Barton és társai, 1987) alapján. Ez a gyakorlat az általunk *Buffalo*<sup>n</sup>-nek nevezett nyelvvel foglalkozik, ami nagyon hasonlatos az angolhoz (legalábbis az  $\mathcal{E}_0$ -hoz), azzal a kivétellel, hogy a szókincsben csak egy szó található: a *buffalo*. Íme, két mondat a nyelvből:

Buffalo buffalo buffalo Buffalo buffalo.

Buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

Arra az esetre, ha nem hinné el, hogy ezek mondatok, álljon itt két angol mondat ugyanezzel a szintaktikai struktúrával:

Dallas cattle bewilder Denver cattle.

Chefs London critics admire cook French food.

Írjon egy nyelvtant a *Buffalo*" számára! A lexikális kategóriák a város, a többes számú főnév és a (tranzitív) ige, és egy szabálynak kell lennie a mondat, egynek az igei kifejezés és háromnak a következő főnévi kifejezések számára: többes számú főnév, főnévi kifejezés, amelyet a város mint módosító előz meg, és főnévi kifejezés, amelyet redukált relatív klóz követ. A redukált relatív klóz egy olyan klóz, amelyből hiányzik a relatív névmás. Emellett a klóz egy alanyi főnévi kifejezést követő tárgy nélküli igéből áll. Egy példa a redukált relatív klózra a „London critics admire” a fenti példában. Foglalja táblázatba a *Buffalo*" lehetséges elemzéseinek számát  $n = 1\dots 10$ -re! Külön pontért: Carl de Marcken kiszámolta, hogy 121 030 872 213 055 159 681 184 485 olyan *Buffalo*" mondat van, amely 200 hosszú (az általa használt nyelvtanra). Hogyan csinálta?

- 22.13.** Rajzolja fel a 939. oldalon található „John egy elegáns étterembe megy” történet szövegelemzési fáját! Használja a Segment-re vonatkozó két szabályt, megadva a helyes CoherenceRelation-t minden egyes csomópontra! (Nem kell az egyes mondatok elemzéseit megmutatnia.) Most tegye meg ugyanezt egy Ön által választott tételezés 5–10 mondat hosszúságú szövegre!
- 22.14.** Elfelejtettük megemlíteni, hogy a 22.1. feladat szövegének címe: „Ruhamosás”. Olvassa újra a szöveget, és válaszolja meg a 22.7. feladat kérdéseit újra! Ezúttal jobban ment? Bransford és Johnson (1973) ezt a szöveget használta egy jobban ellenőrzött kísérletben, és azt tapasztalta, hogy a cím sokat segített. Mit mond ez Önnek a szövegértésről?

# 23. VALÓSZÍNŰSÉGI NYELV-FELDOLGOZÁS

*Ebben a fejezetben meglátjuk, hogyan lehet egyszerű, statisztikailag tanított nyelvi modelleket szavak millióinak feldolgozására használni ahelyett, hogy csak egyes mondatok feldolgozására használnánk*

A 22. fejezetben láttuk, hogyan képes egy ágens kommunikálni egy másik (szoftver vagy emberi) ágennel közös nyelvi megnyilatkozások segítségével. A megnyilatkozások teljes szintaktikai és szemantikai elemzése szükséges a jelentésük teljes kinyeréséhez, ami azért lehetséges, mert a megnyilatkozások rövidek és körülhatárolt tárgyterületre korlátozottak.

Ebben a fejezetben a nyelvmegértés **korpuszalapú** (*corpus-based*) megközelítését tárgyaljuk. A korpusz egy nagy szöveggyűjtemény, például az a több milliárd oldal, ami a világhálót épít fel. A szövegeket emberek írják embereknek, és a szoftverek feladata az, hogy megkönnyítsék az embereknek az információkeresést. Ez a megközelítés magában foglalja statisztikák és tanulás használatát a korpusz kihasználására, és jellemzően olyan valószínűségi nyelvi modelleket von maga után, amelyek az adatokból tanulhatók, és amelyek egyszerűbbek, mint a 22. fejezet kibővített DCG-i. A legtöbb feladat esetén az adatmennyiség kárpolt azért, hogy egyszerűbb nyelvi modellt használunk. Hárrom specifikus feladatot fogunk áttekinteni: az **információkeresést** (*information retrieval*) (23.2. alfejezet), az **információkinyerést** (*information extraction*) (23.3. alfejezet) és a **gépi fordítást** (*machine translation*) (23.4. alfejezet). Elsőként azonban a valószínűségi nyelvi modelleket (*probabilistic language model*) mutatjuk be röviden.

## 23.1. VALÓSZÍNŰSÉGI NYELVI MODELLEK

A 22. fejezet a nyelv *logikai* modelljét adta meg: CFG-ket és DCG-ket használtunk, hogy egy adott karakterfüzérrel előntsük, eleme-e vagy sem egy nyelvnek. Ebben az alfejezetben számos *valószínűségi* modellt vezetünk be. A valószínűségi modellek számos előnyvel rendelkeznek. Kényelmesen taníthatók adatok alapján: a tanulás minden össze az előfordulások megszámolásából áll (némi tűréssel a kis mintaméret okozta hibák miatt). Továbbá robusztusabbak (mivel *bármely* karakterfüzérét elfogadnak, habár kis valószínűséggel), visszatükrözik azt a tényt, miszerint nem a beszélők 100%-a ért egyet abban, hogy mely mondatok részei a nyelvnek; valamint alkalmasak a többéterműség feloldására: a valószínűségre támaszkodva kiválasztható a legvalószínűbb értelmezés.

A **valószínűségi nyelvi modell** (*probabilistic language model*) valószínűségi eloszlást definiál egy (esetleg végtelen) karakterfüzér-halmaz felett. Példaként állhatnak a 15.6. alfejezetben beszédmegértésre használt bi- és trigram nyelvi modellek. Az unigram modell  $P(w)$  valószínűséget rendel a szókincs minden egyes szavához. A modell feltételezi, hogy a szavak függetlenül lettek kiválasztva, azaz a karakterfüzér valószínűsége

egyszerűen a szavak valószínűségének szorzata:  $\prod_i P(w_i)$ . A következő húszszavas szekvenciát véletlen módon generáltuk, a könyv szavainak unigram modellje alapján:<sup>1</sup>

logical are as confusing a may right tries agent goal the was diesel more object then information-gathering search is

A bigram modell egy  $P(w_i|w_{i-1})$  valószínűséget rendel minden egyes szóhoz, adott előző szó esetén. A 15.21. ábra néhány bigram-valószínűséget mutatott be. A könyv bigram modellje a következő véletlen szekvenciát generálja:

planning purely diagnostic expert system are very similar computational approach would be represented compactly using tic tac toe a predicate

Általában, egy  $n$ -gram modell az előző  $n - 1$  szó alapján szabja meg a  $P(w_i|w_{i-(n-1)}...w_{i-1})$  valószínűséget. A könyv trigram modellje ezt a véletlen szekvenciát generálja:

planning and scheduling are integrated the success of naive bayes model is just a possible prior source by the time

Még a fenti kis példa alapján is láthatónak kell lennie, hogy a trigram modell jobb, mint a bigram (amely pedig jobb, mint az unigram), mind az angol nyelv, mind egy MIT-könyv témájának közelítésében. A modellek maguk is egyetértenek ezzel: a trigram modell a véletlen módon generált sztringjéhez  $10^{-10}$  valószínűséget rendel, a bigram  $10^{-29}$ -et, az unigram pedig  $10^{-59}$ -et.

Ez a könyv félmillió szávával nem tartalmaz elég adatot ahhoz, hogy jó minőségű bigram modellt lehessen előállítani belőle, nem is beszélve a trigram modellről. A könyv szókincse körülbelül 15 ezer különböző szót tartalmaz, tehát a bigram modell  $15\ 000^2 = 225$  millió szópárt tartalmaz. Világos, hogy a szópárok legalább 99,8%-a 0 gyakoriságú, de nem akarjuk, hogy a modell azt állítsa, hogy ezek a szópárok lehetetlenek. Szükségünk van valamilyen simításra (*smoothing*) a nulla gyakoriságok felett. A legegyszerűbb megoldás az **adj-hozzá-egyet simítás (add-one smoothing)**: minden lehetséges bigram gyakoriságához hozzáadunk egyet. Azaz amennyiben a korpuszban  $N$  szó és  $B$  lehetséges bigram található, akkor minden  $c$  gyakoriságú bigramhoz egy  $(c + 1)/(N + B)$  értékű valószínűség-beclsőt rendelünk. Ez a módszer megszünteti a nulla valószínűségű  $n$ -gramok problémáját, de az a követelmény, hogy minden gyakoriságot pontosan eggyel kell növelni, kétséges, és rossz beclsésekhez vezethet.

A másik megközelítés a **lineáris interpoláció alapuló simítás (linear interpolation smoothing)**, ami lineárisan kombinálja a trigam, bigram és unigram modelleket. A valószínűség-beclsőt a következőképpen definiáljuk:

$$\hat{P}(w_i|w_{i-2}w_{i-1}) = c_3 P(w_i|w_{i-2}w_{i-1}) + c_2 P(w_i|w_{i-1}) + c_1 P(w_i)$$

ahol  $c_3 + c_2 + c_1 = 1$ . A  $c_i$  paraméterek lehetnek rögzítettek vagy EM algoritmussal taníthatók. Az is lehetséges, hogy a  $c_i$  értékeit az  $n$ -gram gyakoriságuktól függőenek választjuk, azaz nagyobb súlyt adunk azoknak a valószínűségi beclsőknek, amelyeket nagyobb gyakoriságokból számítunk.

<sup>1</sup> Mivel nem lett volna értelme lefordítani magyarra az egyes szavakat, ezért meghagyuk az eredeti angol nyelvű példát. (A ford.)

A nyelvi modell egyik lehetséges *kiértékelési* módja a következő: először válasszuk szét a korpuszt egy tanító és egy teszthalmazra. Határozzuk meg a modell paramétereit a tanító halmaz alapján. Ezután számítsuk ki a valószínűséget a teszthalmazra a modell alapján; minél nagyobb a valószínűség, annál jobb. A megközelítés egyik problémája az, hogy hosszú karakterfüzérek esetén a  $P(\text{szavak})$  nagyon kicsi; a kis számok lebegő-pontos alulesordulást okozhatnak, vagy egyszerűen túl nehéz lenne elolvasni őket. Tehát a valószínűség helyett a modell összetettségét (*perplexity*) számíthatjuk ki a teszt karakterfüzéren:

$$\text{Összetettség}(\text{szavak}) = 2^{-\log_2(P(\text{szavak}))/N}$$

ahol  $N$  a *szavak* száma. Minél kisebb az összetettség, annál jobb a modell. Az az  $n$ -gram modell, amely minden szóhoz  $1/k$  valószínűséget rendel,  $k$  összetettségű; az összetettséget úgy is lehet értelmezni, mint átlagos elágazási tényezőt.

Példaként arra, hogy mire képesek az  $n$ -gram modellek, vegyük a **szegmentáció (segmentation)** feladatát, ami szóhatárok megtalálása szóköz nélküli szövegben. Ez a feladat elengedhetetlen a japán és kínai nyelv esetén; ezek olyan nyelvek, amelyek nem rakkák szóközöt a szavak közé, feltételezzük azonban, hogy a legtöbb olvasónak az angol nyelv<sup>2</sup> jobban megfelel. Az

*It is easy to read words without spaces*<sup>3</sup>

mondatot tényleg könnyű elolvasni. Az olvasó arra gondolhat, hogy ezt az teszi lehetővé, hogy teljesen ismerjük az angol szintaktikát, szemantikát és pragmatikát. Meg fogjuk mutatni, hogy a mondatot egy egyszerű unigram modellel is könnyen dekódolni lehet.

Korábban láthattuk, hogy a Viterbi algoritmus (15.9) hogyan használható a legvalószínűbb szekvencia megtalálására egy szó-valószínűségi hálóban. A 23.1. ábrán látható a Viterbi algoritmus olyan változata, amelyet specifikusan a szegmentációs probléma megoldására tervezünk. Bemenete a  $P(\text{szó})$  unigram valószínűségi eloszlás és egy karakterfüzér. Ezután az algoritmus a karakterfüzér minden egyes i pozíciójára a *legjobb[i]* elemben eltárolja a legvalószínűbb i-ig tartó karakterfüzér valószínűségét. Emellett a *szavak[i]* elemben eltárolja azt az i-edik pozícióban végződő szót, ami a legnagyobb valószínűséget adta. Miután felépítette a *legjobb* és a *szavak* tömbököt dinamikus programozási módon, hátrafelé mozogva feldolgozza a *szavak* tömböt, hogy megtalálja a legjobb utat. Ebben az esetben, a könyv unigram modellje alapján a legjobb szekvencia ténylegesen az „*It is easy to read words without spaces*”,<sup>4</sup>  $10^{-25}$  valószínűséggel. A szekvencia részeinek összehasonlítása során látható, hogy az „easy” unigram valószínűsége  $2,6 \times 10^{-4}$ , miközben az alternatív „as” es „y”<sup>5</sup> valószínűsége sokkal kisebb,  $9,8 \times 10^{-12}$  annak ellenére, hogy a könyv képleteiben viszonylag gyakran előfordul az „e” és az „y”. Hasonlóképpen:

$$P(\text{„without”}) = 0,0004$$

$$P(\text{„with”}) = 0,005; P(\text{„out”}) = 0,0008$$

$$P(\text{„with out”}) = 0,005 \times 0,0008 = 0,000004$$

<sup>2</sup> Mivel nem lehetett volna hasonló magyar példát valószínűségekkel együtt elkészíteni, ezért meghagyottuk az eredeti angol nyelvű példát. (A *ford.*)

<sup>3</sup> Könnyűszavakat szóközök nélkül olvasni (A *ford.*)

<sup>4</sup> Könnyű szavakat szóközök nélkül olvasni (A *ford.*)

<sup>5</sup> *easy* – könnyű; *as* – mint (A *ford.*)

```

function VITERBI-SZEGMENTÁCIÓ(szöveg, P) returns legjobb szavak és valószínűségek
  inputs: szöveg, szóköz nélküli karakterfüzér
            P, szavak unigram modellje

  n  $\leftarrow$  Hossz(szöveg)
  szavak  $\leftarrow$  n + 1 hosszúságú üres vektor
  legjobb  $\leftarrow$  n + 1 hosszúságú vektor, 0,0-ra inicializálva
  legjobb[0]  $\leftarrow$  1,0
  /* Legjobb és szavak tömbök feltöltése dinamikus programozási módon */
  for i = 0 to n do
    for j = 0 to i - 1 do
      szó  $\leftarrow$  szöveg[j:i]
      w  $\leftarrow$  Hossz(szó)
      if P[szó]  $\times$  legjobb[i - w]  $\geq$  legjobb[i] then
        legjobb[i]  $\leftarrow$  P[szó]  $\times$  legjobb[i - w]
        szavak[i]  $\leftarrow$  szó
    /* Legjobb szó szekvencia kigyűjtése */
    szekvencia  $\leftarrow$  üres lista
    i  $\leftarrow$  n
    while i > 0 do
      szavak[i] a szekvencia elő
      i  $\leftarrow$  i - Hossz(szavak[i])
    /* Legjobb szó szekvencia és valószínűségének visszaadása */
  return szekvencia, legjobb[i]

```

**23.1. ábra.** Viterbi-alapú szószegmentáló algoritmus. Egy szóközöket nem tartalmazó szófüzért feldolgozva megadja a legvalószínűbb szavakra történő szegmentációt.

Ennek következtében az unigram modell szerint a „without” százszor nagyobb valószínűségű, mint a „with out”.<sup>6</sup>

Ebben a bekezdésben a szavak feletti *n*-gram modellekéről értekeztünk, azonban az *n*-gram modelleket számos egyéb egység – például karakterek vagy **beszédrészek** (*parts of speech*) – felett is lehet értelmezni.

## Valószínűségi környezetfüggetlen nyelvtanok

Az *n*-gram modellek a korpuszon belüli közös előfordulási statisztikát használják ki, azonban nincs semmilyen információjuk a nyelvtanról *n*-nél nagyobb távolságra. A **valószínűségi környezetfüggetlen nyelvtan** – PCFG<sup>7</sup> (*probabilistic context-free grammar*) – egy alternatív nyelvi modell, ami egy olyan CFG, melyben minden átfirási szabályhoz valószínűséget rendelünk. Az azonos bal oldallal rendelkező szabályok valószínűségének összege 1. A 23.2. ábrán az  $\mathcal{E}_0$  nyelvtan egy részletének PCFG-je látható.

A PCFG-modellben egy karakterfüzér valószínűsége – a  $P(\text{szavak})$  – egyszerűen az elemzési fa valószínűségeinek összege. Egy adott fa valószínűsége a fa csomópontjait

<sup>6</sup> without – nélküli; with – vele; out – kint (A ford.)

<sup>7</sup> A PCFG-k másik megnevezése a sztochasztikus környezetfüggetlen nyelvtan (stochastic context-free grammar), avagy SCFG.

$$S \rightarrow NP\ VP [1,00]$$

$$NP \rightarrow Pronoun [0,10]$$

- | *Neme*[0,10]
- | *Noun* [0,20]
- | *Article Noun* [0,50]
- | *NP PP* [0,10]

$$VP \rightarrow Ige [0,60]$$

- | *VP NP* [0,20]
- | *VP PP* [0,20]

$$PP \rightarrow Preposition\ NP [1,00]$$

$$Noun \rightarrow \text{breeze} [0,10] \mid \text{wumpus} [0,15] \mid \text{agent} [0,05] \mid \dots$$

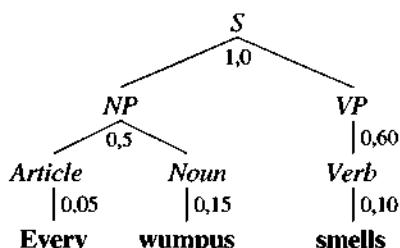
$$Verb \rightarrow \text{sees} [0,15] \mid \text{smells} [0,10] \mid \text{goes} [0,25] \mid \dots$$

$$Pronoun \rightarrow \text{me} [0,05] \mid \text{you} [0,10] \mid \text{I} [0,25] \mid \text{it} [0,20] \mid \dots$$

$$Article \rightarrow \text{the} [0,30] \mid \text{a} [0,35] \mid \text{every} [0,05] \mid \dots$$

$$Preposition \rightarrow \text{to} [0,30] \mid \text{in} [0,10] \mid \text{on} [0,20] \mid \dots$$

**23.2. ábra.** Az  $\mathcal{E}_0$  nyelvtan egy részletének valószínűségi környezetfüggetlen nyelvtana (PCFG) és szókincse. A szögletes zárójelben levő szám jelzi a valószínűségét annak, hogy az adott bal oldali szimbólumot a megfelelő szabály szerint írjuk át.



**23.3. ábra.** A „Minden wumpus bázlik” mondat elemzési fája, megadva minden egyes részfa valószínűségét. A teljes fa valószínűsége  $1,0 \times 0,5 \times 0,05 \times 0,15 \times 0,60 \times 0,10 = 0,000225$ . Mivel a mondatnak ez az egyetlen elemzése, ezért ennyi a mondat valószínűsége is.

felépítő szabályok valószínűségeinek szorzata. A 23.3. ábra bemutatja, hogyan lehet kiírni egy mondat valószínűségét. A valószínűség kiszámítható egy CFG diagramm elemző alkalmazásával, amely megadja az összes lehetséges elemzést, majd egyszerűen össze kell adni a valószínűségeket. Azonban ha csak a legvalószínűbb elemzés érdekel minket, akkor a nem valószínű elemzések meghatározása pazarlás. A legvalószínűbb elemzés hatékony megtalálására használhatunk egy Viterbi algoritmus variációt vagy egy legjobbat-először keresési technikát (mint például az  $A^*$ -ot).

A PCFG-vel az a probléma, hogy környezetfüggetlen. Ez azt jelenti, hogy a különbség a  $P(\text{„eat a banana”})$  és a  $P(\text{„eat a bandanna”})$ <sup>8</sup> között minden össze a  $P(\text{„banana”})$

<sup>8</sup> „egy banánt eszik” – „egy selyemkendőt eszik” (*A ford.*)

és  $P(\text{„bandanna”})$  közti különbségtől függ, és nem az „enni” ige és a megfelelő objektumok közti kapcsolattól. Hogy megkapjuk ezt a kapcsolatot, szükség van valamilyen környezetfüggő modellre, mint például a **szókincsel ellátott PCFG-re (lexicalized PCFG)**, amelyben a kifejezés feje<sup>9</sup> szerepet játszhat a kifejezés valószínűségének meghatározásában. Elegendő tanító adat esetén a  $VP \rightarrow VP\ NP$  szabály kondicionáltható a beágyazott  $VP$  fejére („enni”) és az  $NP$  fejére („banán”). Ezáltal a szókincsel ellátott PCFG-k képesek az  $n$ -gram modellek közös előfordulási megszorításai egy részének a megragadására, megtartva a CFG-modellek nyelvtani megszorításait.

A PCFG másik problémája az, hogy erősen preferálja a rövid mondatokat. Egy olyan korpuszban, mint a *Wall Street Journal*, az átlagos mondathossz körülbelül 25 szó. Azonban egy PCFG általában úgy végez, hogy viszonylag magas valószínűséget rendel olyan szabályokhoz, mint az  $S \rightarrow NP\ VP$ , az  $NP \rightarrow Pronoun$  és a  $VP \rightarrow Verb$ . Ez azt jelenti, hogy a PCFG viszonylag nagy valószínűséget fog rendelni sok rövid mondathoz, mint pl. „ő aludt”, azonban a *Journalban* sokkal valószínűbben találkozunk olyan mondatokkal, mint „Egy megbízható kormányzati forrás jelentése szerint az az állítás, hogy aludt, hihető”. Úgy tűnik, hogy a *Journal* mondatai igazából nem környezetfüggetlenek, hanem az fróknak van elképzelésük az elvárt mondathosszról, amit fel is használnak lágy globális kényszerként a mondatok írásakor. Ezt nehéz visszatükrözni egy PCFG-ben.

## PCFG-valószínűségek tanulása

Egy PCFG-modell létrehozásához a CFG konstrukciójának összes nehézségével szembesülünk, ehhez hozzátevődik még az egyes szabályok valószínűséggel való ellátása. Ez azt sugallja, hogy a nyelvtan adatokból való **tanulása (learning)** hasznosabb lehet, mint a tudásmérnöki megközelítés. Csakúgy, mint a beszédmegértés esetén is, kétféle adat áll rendelkezésre: elemzett és nem elemzett. A feladat sokkal egyszerűbb, ha az adatok **elemzési fáját** nyelvészük (vagy legalábbis képzett anyanyelvi beszélők) készítették el. Egy ilyen korpusz elkészítése óriási feladat, a legnagyobb korpuszok „mindössze” körülbelül egymilliós szót tartalmaznak. Az elemzési fa korpusz alapján a PCFG-t egyszerűen számlálással (és simítással) készítjük el: minden egyes nem záró szimbólumra megnézzük az összes olyan csomópontot, amelynek ez a szimbólum a gyökere, és előállítjuk a csomópontok gyermekinek összes különböző kombinációt lefró szabályokat. Például ha az  $NP$  szimbólum 100 000-szer fordul elő, és ebből 20 000 esetben a gyermek listája [ $NP$ ,  $PP$ ], akkor a következő szabályt állítjuk elő:

$$NP \rightarrow NP\ PP \quad [0,20]$$

A feladat sokkal nehezebb, ha csak elemzetlen szöveggel rendelkezünk. Először is két problémával szembesülünk: a nyelvtani szabályok struktúrájának és az egyes szabályok valószínűségének megtanulásával. (Ugyanezt a megkülönböztetést tesszük neurális hálózatok, valamint Bayes-hálók tanulása esetén is.)

Pillanatnyilag feltételezzük, hogy a szabályok struktúrája adott, és csak a valószínűségeket próbáljuk megtanulni. Alkalmazhatunk egy várhatóérték-maximalizálás (**expectation-**

<sup>9</sup> A kifejezés feje a legfontosabb szó, például a főnév a főnévi szerkezetben.

**maximization**, EM) módszert, úgy, mint az RMM-ek tanulásánál. A paraméterek – amelyeket tanulni próbálunk – a szabály-valószínűségek. A rejtett változók az elemzési fák: nem tudjuk, hogy a  $w_1, \dots, w_t$  szavakból álló karakterfüzér ténylegesen az  $X \rightarrow \alpha$  szabály generálja-e, vagy sem. Az E lépés megbecsüli az egyes részszekvenciák egyes szabályok által történő generálásának valószínűségét. Ezután az M lépés megbecsüli az egyes szabályok valószínűségét. Az egész számítást el lehet végezni dinamikus programozási módon, az ún. **belső–külső (inside–outside)** algoritmussal, ami a HMM tanulás előre–hátra algoritmusának analógiája.

A belső–külső algoritmus varázslatosnak tűnik, hiszen elemzetlen szövegekből állít elő nyelvtant. Azonban számos hátrányaival rendelkezik. Először is lassú: ahol  $n$  a mondatbeli szavak,  $t$  pedig a nem záró szimbólumok száma. Másodsorban, a valószínűségi hozzárendelések tere nagyon nagy, és a tapasztalatok alapján a lokális maximumokban való bennragadás súlyos probléma. Alternatív módszerek – például szimulált lehűtés – megpróbálhatók ugyan, de ezek még nagyobb számítási igényűek. Harmadsorban, a kapott nyelvtanok által elvégzett elemzések gyakran nehezen érthetők, és nem elégítik ki a nyelvészket. Ez megnehezíti a manuálisan előállított tudás kombinálását az automatikus indukcióval.

## PCFG-szabálystruktúrák tanulása

Most pedig tételezzük fel, hogy a nyelvtani szabályok struktúrája nem ismert. Az első probléma, amivel szembesülünk, az, hogy a lehetséges szabályhalmazok tere végletesen, azaz nem tudjuk, hogy hány szabályt vegyük figyelembe, és azt sem, hogy az egyes szabályok milyen hosszúak lehetnek. A probléma egyik lehetséges megkerülése az, hogy a nyelvtani szabályokat Chomsky normál alakban (**Chomsky normal form**) tanuljuk, ami azt jelenti, hogy minden szabály a következő két alak egyike lehet:

$$\begin{aligned} X &\rightarrow YZ \\ X &\rightarrow t \end{aligned}$$

ahol  $X$ ,  $Y$  és  $Z$  nem záró, míg  $t$  záró szimbólum. minden környezetfüggetlen nyelvtant át lehet írni Chomsky normál alakra, amely pontosan ugyanazt a nyelvet fogadjá el. Ezután önhatalmúlag  $n$  nem záró szimbólumra szorítkozhatunk, ezáltal  $n^3 + nv$  szabályt kapunk, ahol  $v$  a záró szimbólumok száma. A gyakorlatban ez a módszer csak kis nyelvtanok esetén bizonyult hatékonynak. A **bayesi modellösszevonás** (**Bayesian model merging**) alternatív megközelítés hasonló a SEQUITUR modellhez (lásd 22.8. alfejezet). A módszer mondatonkénti lokális modellek (nyelvtanok) építésével kezd, majd a minimális leíróhossz felhasználásával összevonja a modelleket.

## 23.2. INFORMÁCIÓKERESÉS

Az **információkeresés** (**information retrieval**) feladata olyan dokumentumok megtalálása, amelyek relevánsak a felhasználó információigényére nézve. Az információkereső rendszerek legjobban ismert példái a világháló keresőgépei. A felhasználó megadhat egy lekérdezést – mint például [MI-könyv] – a keresőgépnek, majd megnéz-

heti a releváns oldalak listáját. Ebben az alfejezetben bemutatjuk, hogyan épülnek fel ezek a rendszerek. Egy információkereső (ezentúl IR-) rendszer a következő módokon jellemezhető:

- **Dokumentumgyűjtemény.** minden rendszernek el kell döntenie, hogy mit kezel dokumentumként: egy bekezdést, egy oldalt vagy egy többoldalas szöveget.
- **Lekérdezőnyelven megadott kérdés.** A lekérdezés (*query*) írja le, hogy a felhasználó mit szeretne megtudni. A lekérdezőnyelv (*query language*) lehet egyszerűen szavak listája (pl. [MI-könyv]); vagy meg lehet adni egymás mellett álló szavakból álló kifejezést (pl. [„MI-könyv”]); logikai operátorokat tartalmazhat (pl. [MI ÉS könyv]), nem logikai operátorokat (például [MI KÖZELBEN könyv] vagy [MI könyv CÍM:www.aaai.org]) foglalhat magában.
- **Eredményhalmaz.** Ez a dokumentumok azon részhalmaza, amit az IR a keresés alapján relevánsnak (*relevant*) ítélt. *Releváns* azt értjük, hogy valószínűleg hasznos lesz a kérdést feltevő személy számára, arra a bizonyos információigényre, amelyet a lekérdezésben fogalmazott meg.
- **Az eredményhalmaz megjelenítése.** Ez lehet olyan egyszerű, mint a dokumentum-címek rendezett rangsorolt listája, vagy olyan bonyolult, mint az eredményhalmaz háromdimenziós térbe vetített, forgó színes térképe.

Az előző fejezet olvasása után gondolhatunk arra, hogy egy információkereső rendszert úgy is felépíthetünk, hogy a dokumentumhalmazt elemzés után leképezzük logikai mondatok tudásbázisába, majd elemezzük a kérdéseket, és MEGKÉRDEZ-zük a tudásbázist, hogy a válaszokat megkapjuk. Sajnos senkinék sem sikerült így működő, nagyméretű IR-rendszert készítenie. Egyszerűen túl bonyolult olyan szókincset és nyelvtant építeni, amely képes nagy dokumentumhalmazokat lefedni, így minden IR-rendszer egyszerűbb nyelvi modellt használ.

A legkorábbi IR-rendszerek a **Boole-kulcsszó modell** (**Boolean keyword model**) szerint működtek. A dokumentumgyűjtemény minden egyes szavát úgy kezelik, mint egy Boole-tulajdonságot, amely igaz, ha a szó előfordul a dokumentumban, és hamis, ha nem. Azaz a „visszakeresés” tulajdonság igaz erre a fejezetre, de hamis a 15. fejezetre. A lekérdezőnyelv a tulajdonságok feletti logikai kifejezések nyelve. Például az [információkeresés ÉS lekérdezés] lekérdezés igaz erre a fejezetre, de nem igaz a 15. fejezetre.

Ez a modell rendelkezik azzal az előnnyel, hogy könnyű elmagyarázni és megvalósítani. Azonban rendelkezik néhány hátránnal is. Először, a dokumentum relevanciája egyetlen bit, így nincs semmilyen iránymutatás arra, hogy hogyan rendezzük a megjelenítés során a releváns dokumentumokat. Másodszor, a logikai kifejezések szokatlanok lehetnek a nem programozó vagy nem logikával foglalkozó felhasználók számára. Harmadszor, még egy gyakorlott felhasználó számára is nehéz lehet a megfelelő lekérdezést megfogalmazni. Tételezzük fel, hogy az [információkeresés ÉS modellek ÉS optimálás] kérdést tesszük fel, és üres eredményhalmazt kapunk vissza. Megpróbálhatjuk az [információkeresés VAGY modellek VAGY optimálás] lekérdezést, azonban ha túl sok eredményt ad vissza, akkor nehéz megmondani, mit kell utána kipróbálni.

A legtöbb IR-rendszer a szavak előfordulási statisztikájára (és esetleg más alacsony szintű jellemzőkre) épít. Bemutatunk egy valószínűségi keretrendszeret, amely jól illeszkedik a tárgyalt nyelvi modellekhez. Az alapötlet az, hogy egy adott lekérdezéshez

meg akarjuk találni azokat a dokumentumokat, amelyek relevánsak. Más szavakkal, ki akarjuk számolni a:

$$P(R = \text{igaz} | D, Q)$$

értéket, amelyben  $D$  a dokumentum,  $Q$  a lekérdezés,  $R$  pedig egy véletlen logikai változó, amely a relevanciát fejezi ki. Ha meghatároztuk ezt az értéket, alkalmazhatjuk a valószínűségi rendezési elvet, amely szerint amennyiben be kell mutatnunk az eredményhalmazt, akkor azt csökkenő valószínűségű relevancia szerint kell tennünk.

Számos lehetőség létezik a  $P(R = \text{igaz} | D, Q)$  együttes eloszlás dekomponálására. Itt az ún. **nyelvi modellezés (language modeling)** megközelítést fogjuk bemutatni, amely minden egyes dokumentumra egy nyelvi modellt becsül, majd az adott dokumentum nyelvi modellje alapján minden egyes lekérdezésre kiszámítja a lekérdezés valószínűségét. Az  $R = \text{igaz}$  érték jelölésére  $r$ -t használva, a következő alakra írhatjuk át a valószínűséget:

$$\begin{aligned} P(r|D, Q) &= P(D, Q|r)P(r)/P(D, Q) && \text{(a Bayes-szabály alapján)} \\ &= P(Q, D|r)P(D|r)P(r)/PP(D, Q) && \text{(a láncszabály alapján)} \\ &= \alpha P(Q|D, r)P(r|D)/P(D, Q) && \text{(a Bayes-szabály alapján, rögzített } D\text{-re)} \end{aligned}$$

Azt mondta, hogy a  $P(r|D, Q)$  értékét akarjuk maximalizálni, azonban ezzel ekvivalens, ha a  $P(r|D, Q)/P(\neg r|D, Q)$  valószínűségi arányt maximalizáljuk. Azaz a dokumentumokat a következő pontszám alapján rangsorolhatjuk:

$$\frac{P(r|D, Q)}{P(\neg r|D, Q)} = \frac{P(Q|D, r)P(r|D)}{P(Q|D, \neg r)P(\neg r|D)}$$

Ennek az az előnye, hogy kiküszöböli a  $P(D, Q)$  tagot. Most pedig feltételezzük, hogy az irreleváns dokumentumokra a dokumentum független a lekérdezéstől. Más szavakkal, amennyiben egy dokumentum irreleváns egy adott lekérdezésre, akkor a dokumentum ismerete nem fog segíteni a lekérdezés meghatározásában. Ezt a lekérdezést a következő egyenlet írja le:

$$P(D, Q|\neg r) = P(D|\neg r)P(Q|\neg r)$$

Ezzel a feltételezéssel azt kapjuk, hogy:

$$\frac{P(r|D, Q)}{P(\neg r|D, Q)} = P(Q|D, r) \times \frac{P(r|D)}{P(\neg r|D)}$$

A  $P(r|D)/P(\neg r|D)$  tényező a dokumentum relevanciájának lekérdezésfüggetlen valószínűsége. Ez a dokumentum minőségének mértéke: egyes dokumentumok *bármely* lekérdezéshez relevánsak, mert a dokumentum egyszerűen magas színvonalú. Akadémiai környezetben született folyóiratcikkek esetén a relevancia a hivatkozások száma alapján becsülhető, míg weboldalak esetén az oldalra mutató hiperhivatkozások számát használhatjuk. minden esetben nagyobb súlyt adhatunk azoknak a hivatkozásoknak, amelyek maguk is magas színvonalúak. A dokumentum kora szintén szerepelhet a lekérdezésfüggetlen relevancia becslésében.

Az első tényező – a  $P(Q|D, r)$  – a lekérdezés valószínűsége, feltéve egy adott releváns dokumentumot. Hogy megbecsülhessük ezt a valószínűséget, egy nyelvi modellt kell

választanunk, amely megadja, hogy milyen kapcsolatban állnak a lekérdezések a releváns dokumentumokkal. Az egyik népszerű választás a dokumentumok unigram szómodellel történő reprezentálása. Ez az információkeresésben úgy is ismert, mint a **szószák** (*bag of words*) modell, mivel a szavak dokumentumon belüli előfordulási gyakorisága az, ami számít, nem a sorrendjük. Ebben a modellben a „man bites dog” és a „dog bites man” (nagyon rövid) dokumentumok azonosan fognak viselkedni.<sup>10</sup> Világos, hogy *elterő* jelentésűek, azonban az is igaz, hogy mindenketen relevánsak a kutyákat és a harapásokat tartalmazó lekérdezésekre. Ezek után, hogy kiszámolhassuk egy lekérdezés valószínűségét egy adott dokumentum esetében, egyszerűen össze kell szoroznunk a lekérdezésben található szavak valószínűségeit a dokumentum unigram modellnek megfelelően. Ez a lekérdezés **naiv Bayes-** (*naive Bayes*) modellje.  $Q_j$ -vel jelölve a lekérdezés *j*-edik szavát, azt kapjuk, hogy:

$$P(Q|D, r) = \prod_j P(Q_j|D, r)$$

Ez lehetővé teszi a következő egyszerűsítést:

$$\frac{P(r|D, Q)}{P(\neg r|D, Q)} = \prod_j P(Q_j|D, r) \frac{P(r|D)}{P(\neg r|D)}$$

Végre készen állunk, hogy ezeket a matematikai modelleket egy példára alkalmazzuk. A 23.4. ábra a [Bayes informational retrieval model] ([Bayes információkeresés modell]) lekérdezés szavainak ennek a könyvnek öt kiválasztott fejezetéből álló dokumentumgyűjtemény feletti unigram statisztikáját adja meg. Feltesszük, hogy a fejezetek azonos minőségűek, így csak azt kell kiszámolnunk, hogy mennyi a lekérdezés valószínűsége az adott dokumentum esetén, minden egyes dokumentumra. Két alkalommal tesszük ezt meg, egyszer egy  $D_i$  simítatlan maximum-likelihood becslővel, majd egy  $D'_i$

Szavak	Lekérdezés	1. fejezet Bevezetés	13. fejezet Bizonytalanság	15. fejezet Idő	22. fejezet NLP	23. fejezet Aktuális
Bayes	1	5	32	38	0	7
information	1	15	18	8	12	39
retrieval	1	1	1	0	0	17
model	1	9	7	160	9	63
$N$	4	14 680	10 941	18 186	16 397	12 574
$P(Q D_i, r)$		$1,5 \times 10^{-14}$	$2,8 \times 10^{-13}$	0	0	$1,2 \times 10^{-11}$
$P(Q D'_i, r)$		$4,1 \times 10^{-14}$	$7,0 \times 10^{-13}$	$5,2 \times 10^{-13}$	$1,7 \times 10^{-15}$	$1,5 \times 10^{-11}$

**23.4. ábra.** A [Bayes information retrieval model] lekérdezés-valószínűségi IR-modellje a könyv első öt fejezetét tartalmazó dokumentumgyűjtemény felett. Megadjuk a szógyakoriságot minden egyik dokumentum-szó párra, és a szavak számát ( $N$ ) az összes dokumentumra. Két dokumentummodellt alkalmazunk,  $D_i$  az *i*-edik dokumentumon alapuló simítatlan unigram szómodell, míg  $D'_i$  ugyanaz a modell adj-hozzáegyet simítással, majd kiszámítjuk a lekérdezés valószínűségét minden dokumentumra minden modellrel. A jelen (23.) fejezet az egyértelmű győztes, minden modell esetén több mint kétszázszor valószínűbb, mint bármely más dokumentum.

<sup>10</sup> „Egy kutya megharapott egy embert”, illetve „egy ember megharapott egy kutyát”, jellemző újságírói stílus. (A ford.)

adj-hozzá-egyet simító becslőjű modellel. Azt tételeznénk fel, hogy egy ilyen keresésénél ez a fejezet lesz elsőnek rangsorolva, és valóban ez így is van minden modell szerint.

A simított modell rendelkezik azzal az előnyvel, hogy kevésbé érzékeny a zajra, és hogy nemzérus relevancia-valósínűséget képes rendelni olyan dokumentumokhoz, amelyek nem tartalmazzák az összes szót. A simítatlan modellnek az az előnye, hogy könnyű sok dokumentumot tartalmazó gyűjteményekre kiszámolni: ha elkészítünk egy olyan indexet, amely megadja, hogy az adott szót mely dokumentumok tartalmazzák, akkor gyorsan elő tudjuk állítani az eredményhalmazt ezeknek a listáknak a metszettel, és a  $P(Q|D_i)$  értékeit csak a metszethen szereplő dokumentumokra kell kiszámolni, nem pedig mindenre.

## Az IR-rendszerek értékelése

Honnan tudjuk, hogy egy IR-rendszer jól teljesít? Elvégzünk egy kísérletet, amelyben a rendszer kap egy lekérdezéshalmazt, az eredményhalmazokat pedig pontozzuk az emberi relevanciamegítélés szerint. Tradicionálisan két mértéket használunk a pontozásra: a **felidézést (recall)** és **pontosságot (precision)**. Ezeket egy példán keresztül fogjuk bemutatni. Képzeljük el, hogy egy IR-rendszer visszaadott egy eredményhalmazt egy olyan lekérdezésre, amelyre tudjuk, hogy egy 100 dokumentumot tartalmazó korpuszból mely dokumentumok relevánsak, és melyek nem. Az egyes kategóriákba tartozó dokumentumok számát az alábbi táblázat adja meg:

	Az eredményhalmazban	Nem az eredményhalmazban
Releváns	30	20
Nem releváns	10	40

A **pontosság** az eredményhalmaz dokumentumai közül a ténylegesen relevánsak arányát méri. A példánkban a pontosság  $30/(30 + 10) = 0,75$ . A hamis pozitív arány  $1 - 0,75 = 0,25$ . A **felidézés** a gyűjtemény releváns dokumentumaiból az eredményhalmazban megjelenő hárnyadát méri. A példánkban a felidézés  $30/(30 + 20) = 0,60$ . A hamis negatív arány  $1 - 0,60 = 0,40$ . Egy nagyon nagy dokumentumgyűjteményben, mint például a világhálón, a felidézés nehezen számítható, mivel nincs egyszerű módszer a web összes oldalának a relevancia szempontjából történő elemzésére. A legjobb, amit tehetünk, a felidézés becslése mintavételezéssel, vagy pedig teljesen figyelmen kívül hagyjuk a felidézést, és csak a pontosság alapján ítélnünk.

A rendszer kompromisszumot köthet a pontosság és felidézés között. Extrém esetben a rendszer visszaadhatja az összes dokumentumot a dokumentumgyűjteményből az eredményhalmazban, 100%-os felidézést garantálva, azonban a pontossága kicsi lesz. Alternatívaként, a rendszer visszaadhat egyetlen dokumentumot, így a felidézés alacsony lesz, azonban tűrhető esélye van 100%-os pontosságra. A kompromisszum összefoglalásának egyik lehetséges módja az **ROC-görbével (ROC curve)** történhet. Az „ROC” a „vevő működési karakterisztika” (**receiver operating characteristics**) rövidítése (ami nem túlzottan felvilágosító név). Ez egy olyan grafikon, amely a hamis negatív arányt méri az

y tengelyen és a hamis pozitív arányt az x tengelyen, ábrázolva az egyes kompromisszumos pontokat. A görbe alatti terület az IR-rendszer hatékonyságának összefoglalása.

A felidézést és pontosságot akkor definiálták, amikor az IR-kereséseket elsődlegesen könyvtárosok végezték, akik alapos, pontos találatokban voltak érdekeltek. Manapság a legtöbb (napi több százmillió) lekérdezést az internetfelhasználók végzik, akik kevésbé érdekeltek az alaposságban, sokkal inkább abban, hogy azonnal választ kapjanak. Számukra jó mérték az első releváns találat átlagos **reciprokrangja** (**reciprocal rank**). Azaz, amennyiben a rendszer első találata releváns, 1-es pontszámot kap a lekérdezésre, és amennyiben az első kettő nem releváns, de a harmadik az, akkor 1/3-ot. Egy alternatív mérték a **válaszidő** (**time to answer**), ami azt méri, hogy mennyi ideig tart a felhasználónak a problémára kívánt választ megtalálni. Ez kerül a legközelebb ahoz, amit méni szeretnénk, azonban azzal a hátránnal rendelkezik, hogy minden egyes kísérlethez új emberi tesztalanycsoporthoz van szükség.

## Az IR-rendszerk továbbfejlesztése

Az unigram modell az összes szót függetlenként kezeli, azonban mi tudjuk, hogy bizonyos szavak korreláltak: a „dívány” közeli kapcsolatban áll mind a „díványok”-kal, mind a „kanapé”-val. Számos IR-rendszer próbálja figyelembe venni ezeket a korrelációkat.

Például, amennyiben a lekérdezés [dívány], gyalázatos lenne kihagyni az eredményhalmazból azokat a dokumentumokat, amelyek a „DÍVÁNY” vagy a „díványok” szavakat tartalmazzák, de a „dívány”-t nem. A legtöbb IR-rendszer **kisbetű-nagybetű konverziót** (**case folding**) alkalmaz, hogy a „DÍVÁNY”-t „dívány” alakká alakítsa, számos rendszer pedig **szótövesítő** (**stemming**) algoritmusokat, hogy a „díványok” alakkot a „dívány” szótöré redukálja. Ez tipikusan a felidézés kismértékű növekedését eredményezi (az angol nyelv esetén kb. 2%-ot). Azonban ronthatja a pontosságot. Például a „stocking” szótövesítése „stock”<sup>11</sup> alakra valószínűleg csökkenteni fogja a ruházati, valamint pénzügyi eszközökre irányuló lekérdezések pontosságát, azonban növelheti a raktározásra irányuló lekérdezések felidézését. Szabályalapú szótövesítők (például az „-ing” végződés eltávolítása az angolban) nem képesek megkerülni ezt a problémát, de a szótárakon alapuló újabb algoritmusok igen (nem kell eltávolítani az „-ing” képzőt, ha a szó már szerepel a szótárban). Bár az angol nyelv esetén a szótövesítésnek csak kis hatása van, sokkal fontosabb más nyelvek esetén. A német nyelvben gyakran találkozhatunk olyan szavakkal, mint „Lebensversicherungsgesellschaftsangestellter” (életbiztosító cég alkalmazottja). Az olyan nyelvek, mint a finn, a török, az inuit vagy a jupik rekurzív morfológiai szabályokkal rendelkeznek, amelyek elméletileg korlátlan hosszúságú szavakat generálnak.

A következő lépés a **szinonimák** (**synonyms**) felismerése, mint amilyen a „dívány” és a „kanapé”. A szótövesítéshoz hasonlóan ez is a felidézés kismértékű növekedését eredményezheti, azonban a pontosságot veszélyezteti, ha túl agresszíven alkalmazzuk. Akik Tim Couch futballistára kíváncsiak, nem szeretnék átvergődni a kanapékről (couch – kanapé) szóló dokumentumokon. Az a probléma, hogy „a nyelv ugyanúgy írtózik az abszolút szinonimáktól, mint ahogy a természet retteg a vákuumtól” (Cruse,

<sup>11</sup> Harisnya, illetve raktárkészlet. (A ford.)

1986). Azaz, amennyiben két szó azonos dolgot jelent, a nyelv beszélői törekednek a jelentés módosítására, hogy megszüntessék a zúrzavart.

Számos IR-rendszer bizonyos mértékig szó **bigramokat** használ, azonban csak néhányuk valósít meg egy teljes valószínűségi bigram modellt. A **helyesírás-javító (spelling correction)** eljárások alkalmazhatók minden dokumentumok, minden lekérdezések hibáinak javítására.

Végző finomításként az IR-rendszerek tökéletesíthetők **metaadatok (metadata)** figyelembevételével, amelyek a dokumentum szövegén kívül álló adatok, mint például emberek által megadott kulcsszavak vagy dokumentumok közötti hypertext-hivatkozások.

## Az eredményhalmaz prezentálása

A valószínűségi rendezési elv szerint vegyünk egy eredményhalmazt, és a relevancia valószínűségének megfelelően sorba rendezve prezentáljuk a felhasználónak. Ennek akkor van értelme, ha a felhasználó az összes releváns dokumentum minél hamarabb történő megtalálásban érdekelte. Azonban bajba kerül, mert nem veszi figyelembe a *hasznosságot*. Például ha a legrelevánsabb dokumentum két példányban szerepel a gyűjteményben, akkor az első megnézése után a második azonos relevanciájú, de zérus hasznosságú. Számos IR-rendszer rendelkezik mechanizmusokkal, amelyek eliminálnák az előző találatokhoz tulajdonított eredményeket.

Az IR-rendszerek teljesítménynövelésének egyik leghatékonyabb módja a **relevancia-visszacsatolás (relevance feedback)**, amely a felhasználó visszajelzése, hogy az eredeti eredményhalmazból mely dokumentumok voltak relevánsak. A rendszer ezután egy második eredményhalmazt prezentálhat, amelynek dokumentumai hasonlók a megadottakhoz.

Egy másik lehetséges megközelítés az eredményhalmaznak egy rendezett lista helyett egy *címkézett fák* által előállított prezentálása. A **dokumentumosztályozás (document classification)** során az eredményeket egy előre definiált téma- és taxonómiai hierarchiának megfelelően osztályozzuk. Például újsághírekkel álló gyűjteményt a következő kategóriákba lehet sorolni: külföldi, belföldi, üzleti hírek, szórakozás és sport. **Dokumentumklaszterezés (document clustering)** esetén minden eredményhalmazra teljesen új kategóriaifa készül. Az osztályozás akkor használható, ha a gyűjteményben kevés számú téma- és kategória található, míg a klaszterezést a világhálóhoz hasonló széles körű gyűjtemények esetén érdemes használni. Mindkét esetben, miután a felhasználó megad egy lekérdezést, az eredményhalmazt a kategóriáknak megfelelő mappákba rendezve kapja meg.

Az osztályozás felügyelt tanítási probléma, és mint ilyen, a 18. fejezetben ismertetett bármelyik módszerrel megtámadható. Az egyik népszerű megközelítés a döntési fák alkalmazása. Amennyiben rendelkezünk a megfelelő kategóriákkal címkézett dokumentumokból álló tanító halmazzal, építhetünk egyetlen döntési fát, amelynek levelei a dokumentumot a megfelelő kategóriához rendelik. Ez akkor működik jól, ha minden össze néhány kategória van; nagyobb kategóriahalmazok esetén minden egyes kategóriára külön döntési fát építünk, amelynek a levelei megadják, hogy a dokumentum az adott kategóriába tartozik-e vagy sem. Általában az egyes csomópontokban tesztelt tulajdonságok egyedi szavak. Például a „Sport” kategóriában az egyik csomópont tesztelheti a „kosárlabda” szó meglétét. Javított teljesítményű döntési fák, naiv Bayes-modellek,

valamint szupport vektor gépek mindegyikét használták szövegossztályozásra, sok esetben a hitelesség 90–98%-os volt bináris osztályozás esetén.

A klaszterezés egy felügyelet nélküli tanítási módszer. A 20.3. alfejezetben láthattuk, hogy hogyan alkalmazható az EM algoritmus a klaszterezés eredeti becslésének javítására, Gauss-modellek keverékét használva. A dokumentumok klaszterezése nehezebb feladat, mert nem tudjuk, hogy az adatokat egy barátságos Gauss-modell generálta-e, és mert egy sokkal több dimenziós térrrel kell elbánnunk. Számos megközelítést dolgoztak ki.

**Az agglomeratív klaszterezés (agglomerative clustering)** klaszterekből álló fát épít, lemenve egészen az egyedi dokumentumok szintjére. A fa bármely szinten nyeshető, hogy kevesebb kategóriát kapunk, de ezt az algoritmuson kívül vesszük figyelembe. Az elején minden dokumentumot külön klaszternek tekintünk. Ezután megkeressük azt a két klasztert, melyek egy bizonyos távolságmérték szerint legközelebb állnak egymáshoz, és összevonjuk őket. Addig ismételjük a folyamatot, amíg csak egy klaszter marad. A két dokumentum távolságát meghatározó mérték a dokumentumok szavai közti átfedés valamelyen mértéke. Például reprezentálhatjuk a dokumentumot szógyakoriságok vektoraként, ahol a távolságot a két vektor euklideszi távolságaként értelmezzük. Két klaszter távolsága a klaszterek mediánjainak távolságaként értelmezhetjük, vagy a klaszterek elemeinek átlagos távolságát vehetjük figyelembe. Az agglomeratív klaszterezés időigénye  $O(n^2)$ , ahol  $n$  a dokumentumok száma.

A **k-közép klaszterezés (k-means clustering)** pontosan  $k$  darab kategória halmazát állítja elő. A következő elven működik:

1. Vegyük véletlenszerűen  $k$  dokumentumot a  $k$  kategória reprezentálására.
2. Rendeljünk minden dokumentumot a legközelebbi kategóriához.
3. Számoljuk ki minden egyes kategória átlagát, és használjuk a  $k$  átlagot a  $k$  kategóriák új értékeinek reprezentálására.
4. Ismételjük a 2-es és 3-as lépéseket, amíg konvergálnak.

A **k-közép módszer**  $O(n)$  időigényű, ez az egyetlen előnye az agglomeratív klaszterezéshez képest. Általában kevésbé pontos, mint az agglomeratív klaszterezés, bár egyesek szerint majdnem olyan hatékony (Steinbach és társai, 2000).

Az alkalmazott klaszterezési módtól függetlenül van még egy feladat, amit el kell végezni, mielőtt a klaszterezést az eredményhalmaz bemutatására használhatjuk: a klaszter jó leírásának megtalálása. Az osztályozás esetén a kategóriák előre definiáltak (például „jövedelmek”), míg a klaszterezés esetén ki kell találnunk a kategórianeveket. Az egyik választás a klaszter szempontjából reprezentatív szavak listájának használata. A másik lehetőség a klaszter középpontjához közel levő egyik dokumentum címének a használata.

## Az IR-rendszerek megvalósítása

Az eddigiekben csupán absztrakt módon definiáltuk az IR-rendszerek működését, azonban nem magyaráztuk el, hogy hogyan lehet olyan hatékonnyá tenni őket, hogy egy webes keresőgép visszaadhassa a legfelső találatokat egy több milliárd oldalas gyűjteményből egyszerűen másodperc alatt. minden IR-rendszer számára két kulcsfontosságú adatstruktúra szükséges: a szókincs, amely felsorolja a dokumentumok szavait, és az invertált index, amely megadja, hogy az egyes szavak hol szerepelnek a dokumentumgyűjteményben.

**A szóklincs (lexicon)** egy olyan adatstruktúra, amely egy műveletet támogat: megadja, hogy egy adott szó hol szerepel az invertált indexben, amely a szó előfordulásait tárolja. Egyes megvalósítások esetén azt is visszaadja, hogy összesen hány dokumentum tartalmazza a szót. A szókincset hash-tábla vagy valamilyen hasonló adatstruktúrával ajánlott megvalósítani, amely lehetővé teszi a gyors kikeresést. Egyes esetekben kis információtartalmú gyakori szavakat kihagynak a szókincsből. Ezek a **tíltólistás szavak (stop words)** (például „a”, „egy”, „ez” stb.) helyet foglalnak az indexben, és nem javítják az eredmény rangsorolását. Az egyetlen jó indok arra, hogy mégis megtartsuk őket a szókincsben a kifejezéslekérdezést támogató rendszerek esetén áll fenn: a tiltólistás szavak szókincsbeli tárolása szükséges olyan lekérdezések kiszolgálására, mint pl. a „to be or not to be”.<sup>12</sup>

**Az invertált index**<sup>13</sup> (**inverted index**), a könyünk végén található szójegyzékhez hasonlóan, **találati listák (hit lists)** halmazából – az egyes szavak előfordulási helyeiből – áll. A Boole-kulcsszó modell esetén a találati lista nem más, mint a dokumentumok listája. Az unigram modell esetén egy (dокументum, gyakoriság) párokat tartalmazó lista. A kifejezéslekérdezés támogatásához a találati listának az adott szó minden egyes dokumentumon belüli előfordulási helyeit is tartalmaznia kell.

Amennyiben a lekérdezés egyetlen szó (Silverstein szerint ilyen az esetek 26%-a (Silverstein és társai, 1998)), akkor a feldolgozás igen gyors. Egyszerűen kikeressük a szót a szókincsből, hogy megkapjuk a találati lista címét, majd egy üres prioritási sorat hozunk létre. Ezután egyesével végigmegyünk a találati lista dokumentumain, és megnézzük a szó gyakoriságát a dokumentumban. Amennyiben a prioritási sor  $R$ -nél kevesebb elemet tartalmaz (ahol  $R$  az eredményhalmaz elvárt mérete), akkor hozzáadjuk a (dокументum, gyakoriság) párt a sorhoz. Ellenkező esetben, ha a gyakoriság nagyobb, mint a prioritási sor legkisebb elemének gyakorisága, akkor töröljük a legkisebb elemet, és hozzáadjuk az új (dокументum, gyakoriság) párat. Ezáltal a lekérdezés megválasztása  $O(H + R \log R)$  időt vesz igénybe, ahol  $H$  a találati lista dokumentumainak száma. Amennyiben a lekérdezés  $n$  szót tartalmaz, akkor  $n$  találati listát kell összevonni, ami  $O(nH + R \log R)$  időt vesz igénybe.

Azért mutattuk be az IR-rendszerek elméletét a valószínűségi modellen keresztül, mert ez a modell hasznosítja az ötleteket, amelyeket más témaikban is alkalmaztunk. A jelenlegi gyakorlati IR-rendszerek sokkal inkább egy másik megközelítést alkalmaznak, amelyet **vektortér modellnek (vector space model)** hívunk. Ez a modell ugyanúgy a szózsák-megközelítést alkalmazza, mint a valószínűségi modell. minden egyes dokumentumot unigram szógyakoriságok vektoraként ábrázolunk. A lekérdezést is így ábrázoljuk, például a [Bayes information retrieval model] lekérdezést a

[0, ..., 1, 0, ..., 1, 0, ..., 1, 0, ..., 1, 0, ...]

vektor írja le, amelyben az ötlet az, hogy minden egyes szóhoz külön dimenzió tartozik, és a lekérdezés vektorában minden dimenzió 0 értékű, kivéve azt a négy szót, amely a lekérdezésben előfordul. A releváns dokumentumokat úgy kapjuk meg, hogy

<sup>12</sup> lenni vagy nem lenni (A ford.)

<sup>13</sup> Az „invertált index” kifejezés redundáns, sokkal jobb lenne egyszerűen az „index” kifejezést használni. Azért invertált, mert más sorrendben van, mint a szöveg szavai, de ilyen minden index. Azonban az „invertált index” a hagyományos IR-kifejezés.

megkeressük azokat a dokumentumokat, amelyek a lekérdezés vektorának legközelebbi szomszédai a vektortérben. A hasonlóság egyik lehetséges mértéke a lekérdezés- és dokumentumvektor skaláris szorzata: minél nagyobb ez a szorzat, annál közelebb van egymáshoz a két vektor. Algebrailag ez a mérték nagy pontszámot ad azoknak a szavaknak, amelyek mind a dokumentumokban, mind a lekérdezésben gyakran előfordulnak. Geometriailag a két vektor skaláris szorzata az általuk bezárt szög koszinusza, azaz ha két ilyen vektor koszinuszát maximalizáljuk (amennyiben azonos kvadránsban találhatók), akkor az általuk bezárt szög nullához közeli lesz.

A vektortér modell ennél sokkal többre képes. A gyakorlatban számos jellemzővel, finomítással, javítással és kiegészítéssel lett kibővítve. Az az alapötlet, miszerint a dokumentumokat a vektortérbeli hasonlóságuk alapján lehet rangsorolni, lehetővé teszi, hogy új ötleteket építsünk be a numerikus sorrendező rendszerbe. Egyesek azt állítják, hogy a valószínűségi modell ezeket a módosításokat sokkal tisztább elvi alapokon állva tenné lehetővé, azonban az IR-kutatók nem fognak váltani, amíg nem látnak egyértelmű teljesítményjavulást a másik modellhez képest.

Hogy egy átlagos IR-feladat indexelési problémájának nagyságrendjét érzékeljessük, vegyük egy szabványos TREC (Text REtrieval Conference – Szöveg-visszakeresési Konferencia) dokumentumgyűjteményt, amely 750 ezer dokumentumot tartalmaz, összesen 2 GB (gigabajt) szöveggel. A szókincs körülbelül 500 ezer szót tartalmaz, szótövesítés és kisbetű-nagybetű konverzió után; ennyi szó tárolása 7 és 10 MB közötti tárterületet igényel. Az invertált index a (dокументum, gyakoriság) párokkal 324 MB területet igényel, bár tömörítési technikák alkalmazásával csak 83 MB-ot. A tömörítés tárhelyet takarít meg, a feldolgozási követelmények kismértékű növelése árán. Azonban ha a tömörítés lehetővé teszi, hogy az egész indexet a memóriában – és nem hártertárion tároljuk –, akkor jelentős eredő teljesítménynövekedést kapunk. A kifejezés-lekérdezés támogatása a tárígenyt 1200 MB-ra növeli tömörítetlen, illetve 600 MB-ra tömörített esetben. A webes keresőgépek háromezerszer ekkora feladattal dolgoznak. A legtöbb probléma az esetükben is hasonló, azonban nem praktikus terabajtnyi mennyiségi adatot kezelní egyetlen számítógépen, ezért az indexet  $k$  szegmensre vágják, és minden szegmenst külön számítógép tárol. A lekérdezést mindegyik számítógép megkapja, majd a  $k$  eredményhalmazt egy eredményhalmazba vonják össze, amelyet megjelenítenek a felhasználó számára. A webes keresőgépeknek röviden másodpercenként több ezer lekérdezést kell kiszolgálniuk, így a  $k$  számítógép  $n$  másolatára van szükség. Az idő folyamán  $k$  és  $n$  folyamatosan növekszik.

### 23.3. INFORMÁCIÓKINYERÉS

Az **információkinyerés** (*information extraction*) adatbázis-bejegyzések előállításának a folyamata, amely során egy szöveget egy adott osztályba tartozó objektum vagy esemény előfordulásait, valamint az objektumok és események közti relációkat keresve átfutunk. Megpróbálhatjuk konkrét címek kinyerését weboldalakról, adatbázismezőkkel az utca, az állam és az irányítószám számára, vagy konkrét viharok adatainak a kinyerését időjárás-jelentésekben, mezőkkel a hőmérséklet, a szélsebesség és a csapadék számára. Az információkinyerő rendszerek félúton állnak az információkereső rendszerek és a teljes nyelvi elemzők között, hiszen többet kell tenniük, mint egyszerűen szószáknak

tekinteni a dokumentumot, de kevesebbet, mint minden egyes mondat teljes nyelvtani elemzését.

Az információkinyerő rendszerek legegyszerűbb típusát **attribútumalapú (attribute-based)** rendszernek nevezik, mert feltételezi, hogy a teljes szöveg egy objektumról szól, és a feladat ezen objektum attribútumainak kinyerése. Például a 10.5. alfejezetben megemlítettük a „17 hüvelykes SXGA Monitor mindössze 249,99 \$-ért” szövegből az alábbi adatbázis-reláció kinyerésének problémáját:

$$\exists m \in \text{SzámítógépMonitorok} \wedge \text{Méret}(m, \text{Hüvelyk}(17)) \wedge \text{Ár}(m, \$249,99) \\ \wedge \text{Felbontás}(m, 1280 \times 1024)$$

Ennek az informáciának egy része **reguláris kifejezésekkel (regular expression)** kezelhető, amelyek reguláris nyelvtant definiálnak egy karakterfüzérben. Reguláris kifejezéseket használnak olyan Unix-parancsokban, mint a grep, olyan programozási nyelvekben, mint a Perl, valamint olyan szövegszerkesztőkben, mint a Microsoft Word. A részletek eltérők az egyes eszközökben, és legjobban a megfelelő útmutatóból tanulhatók meg, azonban itt bemutatjuk, hogyan lehet reguláris kifejezéseket építeni dollárban megadott áráakra, bemutatva a közös részkifejezéseket:

[0-9]	megfelel bármely 0 és 9 közti számjegynek
[0-9] +	megfelel egy vagy több számjegynek
. [0-9] [0-9]	megfelel számjegyek által követett pontnak
( . [0-9] [0-9] ) ?	megfelel számjegyek által követett pontnak vagy üres füzérnek
\$ [0-9] + ( . [0-9] [0-9] ) ?	megfelel 249,99 \$-nak, vagy 1,23 \$-nak, vagy 1 000 000 \$-nak, vagy...

Az attribútumalapú információkinyerő rendszerek reguláris kifejezések sorozatából építhetők fel, ahol minden egyes attribútumhoz egy reguláris kifejezés tartozik. Ha a reguláris kifejezés pontosan egyszer illeszkedik a szövegre, akkor kivehetjük a szöveg illeszkedő részét, amely az attribútum értéke lesz. Ha nincs illeszkedés, akkor nem tudunk mit tenni, azonban ha több illeszkedés is van, akkor szükségünk van egy eljáráusra, hogy hogyan válasszunk közülük. Az egyik megoldás szerint minden egyes attribútumhoz több reguláris kifejezést rendelünk, prioritás szerint sorba rendezve. Például a legnagyobb prioritású árra vonatkozó reguláris kifejezés a dolláreljel előtti „ár:” karakterfüzérre kereshet, ha ezt nem találja meg, akkor átlépünk egy kevésbé megbízható reguláris kifejezésre. Egy másik stratégia az összes találat kinyerése, majd valamilyen módon választani közülük. Például vehetjük azt a legalacsonyabb árat, amely a legnagyobbnak legalább fele. Ez a megoldás kezelní tudja az olyan szövegeket, mint a „Listaár 99,00 \$, akciós ár 78,00 \$, szállítás 3,00 \$”.

Az attribútumalapú rendszerekkel együtt összetettebbek a relációs alapú információkinyerő rendszerek, amelyeknek több mint egy objektummal kell foglalkozniuk, ráadásul a köztük levő relációkat is figyelembe kell venniük. Azaz, amikor egy ilyen rendszer a „249,99 \$” szöveget látja, akkor nem csak azt kell megállapítania, hogy ez egy ár, hanem azt is, hogy mely objektumnak az ára. A FASTUS egy tipikus relációs alapú információkinyerő rendszer, amely cégegyesülésekrol és felvásárlásokról szóló híreket képes kezelni. El tudja olvasni a következő hírt:

„Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.”<sup>14</sup>

és egy a következőhöz hasonló adatbázisrekordot tud létrehozni:

$e \in \text{VegyesVállalat} \wedge \text{Termék}(e, \text{„Golfütők"}) \wedge \text{Dátum}(e, \text{„Péntek"})$   
 $\wedge \text{Entitás}(e, \text{„Bridgestone Sports Co.”}) \wedge \text{Entitás}(e, \text{„egy helyi konszern”})$   
 $\wedge \text{Entitás}(e, \text{„egy japán kereskedőház”})$

Relációs információknyerő rendszereket gyakran építenek **kaszkádosított véges állapotú átalakítók** (*cascaded finite-state transducer*) segítségével. Ez azt jelenti, hogy egy sor véges állapotú automatából (*finite-state automaton*, FSA) áll, ahol minden egyes automata egy szöveget kap bemenetként, amelyet átalakít egy más formába, majd továbbadja a következő automatának. Ez azért megfelelő, mert az egyes véges automaták hatékonyak lehetnek, és együtt képesek lehetnek a szükséges információ kinyerésére. A FASTUS egy tipikus rendszer, amely a következő öt fokozatból áll:

1. Tokenizálás
2. Komplex szavak kezelése
3. Alapcsoport-kezelés
4. Komplex kifejezések kezelése
5. Struktúra-összevonás

A FASTUS első lépése a **tokenizálás** (*tokenization*), amely a karakterfüzéreket tokenekbe (szavak, számok és írásjelek) szegmentálja. Az angol nyelv esetén a tokenizálás elég egyszerű lehet, pusztán a szóközök (white space) vagy írásjelek mentén történő szegmentálás viszonylag jó hatékonyságú. A japán nyelv esetén a tokenizálásnak szegmentálást kell végeznie, valami olyasmit használva, mint a Viterbi algoritmus (lásd 23.1. ábra). Egyes tokenizálók az olyan jelölőnyelveket is kezelik, mint a HTML-, az SGML és az XML.

A második lépés **komplex szavak** (*complex words*) kezelése, ideértve az olyan szókapcsolatokat, mint a „set up” és a „joint venture”, valamint olyan tulajdonneveket, mint a „Prime Minister Tony Blair” és a „Bridgestone Sports Co.”.<sup>15</sup> Ezeket szótárbejegyzések és véges állapotú nyelvtani szabályok kombinációjaként ismeri fel. Például egy cégnévét a közvetkező szabállyal lehet felismerni:

NagybetűvelKezdődőSzó + („Company” | „Co” | „Inc” | „Ltd”)

A szabályokat gondosan kell megalkotni, majd ellenőrizni kell a felidézést és a pontos-ságot. Az egyik kereskedelmi rendszer az „Intel Chairman Andy Grove”<sup>16</sup> szöveget személy helyként ismerte fel, az egyik szabály miatt:

NagybetűvelKezdődőSzó + („Grove” | „Forest” | „Village” | ...)

<sup>14</sup> „A Bridgestone Sports Vállalat pénteken azt nyilatkozta, hogy vegyes vállalatot hozott létre egy helyi konszernnal és egy japán kereskedőházzal Tajvanon, hogy Japánba szánt golfütőket gyártsanak.” A *golf club* kétértelmű, itt golfütő. (A ford.)

<sup>15</sup> „vegyes vállalat”, „Tony Blair Miniszterelnök”, „Bridgestone Sports Vállalat” (A ford.)

<sup>16</sup> Az Intel elnöke, Andy Grove – a grove magyarul ligetet jelent. (A ford.)

A harmadik lépés az **alapcsoportok (basic groups)** kezelése, amelyek főnévi és igei csoportokat jelentenek. A lépés lényege, hogy ezeket darabokra kell vágni, hogy a későbbi lépések kezelhessék őket. A főnévi csoport a fejszerepet betöltő főnévből áll, amelyet opcionálisan megelőzhetnek névelők és egyéb módosítók. Mivel a főnévi csoport nem tartalmazza az *NP*  $\mathcal{E}_1$ -beli teljes komplexitását, ezért nincs szükség rekurzív környezetfüggetlen nyelvtani szabályokra, a véges automaták által megengedett reguláris nyelvtanok is elégsgesek. Az igei csoport az igéből és a hozzá kapcsolódó segédigékből, valamint határozókból áll, azonban nem tartalmazza a közvetlen és közvetett tárgyakat és az előjárói kifejezéseket. Az előbbi példamondat a következő módon kerülne ki ebből a lépésből:

1 NG:	Bridgestone Sports Co.	10	NG: a local concern
2 VG:	said	11	CJ: and
3 NG:	Friday	12	NG: a Japanese trading house
4 NG:	it	13	VG: to produce
5 VG:	has set up	14	NG: golf clubs
6 NG:	a joint venture	15	VG: to be shipped
7 PR:	in	16	PR: to
8 NG:	Taiwan	17	NG: Japan
9 PR:	with		

Az NG főnévi csoportot, a VG igei csoportot, a PR előjárószót, a CJ pedig kötőszót jelent.

A negyedik lépés az alapcsoportokat **komplex kifejezésekkel** (**complex phrases**) kombinálja. Ismét az a cél, hogy véges állapotú szabályokat használunk a gyors feldolgozás érdekében, és a szabályok olyanok legyenek, hogy (közel) egyértelmű kimeneti kifejezéseket kapjunk. A kombinációs szabályok egyik típusa tárgyterületfüggő eseményekkel foglalkozik. Például a

Cég+ VegyesVállalatLétrehozása („with” Vállalat+)?

szabály a vegyes vállalat létrehozásának egyik lehetséges leírása. A kaszkádban ez az előző lépés, amely eredményét nemcsak a kimenetre, hanem adatbázissémákba is elhelyezi.

Az utolsó lépés az előző lépés által felépített struktúrákat vonja össze (**merges structures**). Amennyiben a következő mondat azt állítja, hogy „A vegyes vállalat januárban kezdi meg a termelést”, akkor ez a lépés felismeri, hogy két hivatkozás történt a vegyes vállalatra, amelyeket össze kellene vonni.

Általánosságban az információknyerés jól működik egy olyan korlátozott tárgyterület esetén, amelyben a tárgyat témaikat előre meg lehet állapítani, továbbá azt is lehet tudni, hogy hogyan írnak róluk. Számos tárgyterületen használhatónak bizonyult ez a technika, azonban nem lehet a teljes természetes nyelvi elemzés helyettesítője.

## 23.4. GÉPI FORDÍTÁS

A gépi fordítás (machine translation) egy természetes nyelvű szöveg egyik nyelvről (forrás) egy másik nyelvre (cél) történő automatikus fordítása. Ez a folyamat számos feladatra alkalmasnak bizonyult, beleértve a következőket:

- Nyersfordítás (rough translation)**, amelyben a cél pusztán az, hogy egy passzus lényegét megkapjuk. Nyelvtanilag helytelen és nem elegáns mondatokat is elfogadunk, mindenkor, amíg a jelentés világos. Például a webszörfözés során a felhasználó gyakran örül az idegen nyelvű weboldal nyersfordításának. Az esetek egy részében egy egynyelvű ember utólagosan szerkesztheti a kimenetet, az eredeti szöveg elolvásásának szükségessége nélkül. A géppel támogatott fordítás ezen típusa azért takarít meg pénzt, mert az ilyen szerkesztőknek kevesebbet kell fizetni, mint a kétnyelvű fordítóknak.
- Korlátozott forrás fordítása (restricted-source translation)**, amelyben a forrás-szöveg téma és formátuma szigorúan korlátozott. Az egyik legsikeresebb példa a TAUM-METEO rendszer, amely időjárás-jelentéseket fordít angolból franciára. Azért működik, mert az időjárás-jelentések nyelvezete erősen stilizált és reguláris.
- Előre szerkesztett fordítás (preedited translation)**, amelyben egy ember úgy szerkeszti meg előre a forrásként dokumentumot, hogy megfeleljen az angol nyelv (vagy bármely forrásnyelv) egy korlátozott részhalmazának. Ez a megközelítés különösen akkor költséghatékony, amikor egyetlen dokumentumot sok nyelvre kell lefordítani, mint például jogi szövegeket az Európai Közösségnél, vagy olyan vállalatok esetén, amelyek ugyanazt a terméket sok országban értékesítik. A korlátozott nyelveket néha „Caterpillar-angolnak” nevezik, mert a Caterpillar vállalat volt az első, amelyik az útmutatóit ilyen formában próbálta megírni. A Xerox olyan nyelvet definiált a javítási útmutatók számára, amely annyira egyszerű volt, hogy géppel le lehetett fordítani minden olyan nyelvre, amellyel a Xerox érintkezésbe kerül. További haszonként az eredeti angol nyelvű útmutatók is világosabbá váltak.
- Irodalmi fordítás (literary translation)**, amely a forrásszöveg legkisebb árnyalatát is megőrzi. Ez jelenleg túlmutat a legmodernebb gépi fordítók képességén is.

A nyersfordítás példájaként álljon itt jelen fejezetünk első bekezdésének a SYSTRAN fordító szolgáltatás által történt fordítása előbb olaszra, majd vissza angolra:<sup>17</sup>

OLASZ: In capitolo 22 abbiamo visto come un agente potrebbe comunicare con un altro agente (essere umano o software) che usando le espressioni in un linguaggio reciprocamente accordato. Completare sintattico e l'analisi semantica delle espressioni necessaria da estrarre il significato completo del utterances ed è possibile perché le espressioni sono corte e limitate ad un settore limitato.

ANGOL: In chapter 22 we have seen as an agent could communicate with an other agent (to be human or software) that using the expressions in a language mutual come to an agreement. Complete syntactic and the semantic analysis of the expressions is necessary to extract the complete meant one of the utterances and is possible because the expressions short and are limited to a dominion.

<sup>17</sup> A fejezet első bekezdésének eredeti angol szövege: *In Chapter 22, we saw how an agent could communicate with another agent (human or software), using utterances in a common language. Complete syntactic and semantic analysis of the utterances is necessary to extract the full meaning of the utterances, and is possible because the utterances are short and restricted to a limited domain.* Látható, hogy a két angol nyelvű szöveg között jelentős eltérések vannak, azonban a gépi fordítás eredményeként kapott szöveg nagyjából értetődő. (A ford.)

A fordítás azért nehéz, mert általában a szövegek mély megértését igényli, amihez pedig a közölt szituáció mély megértése szükséges. Ez még nagyon egyszerű szövegek esetén is fennáll, sőt egyszavas „szövegek” esetén is. Vegyük a „Open” szót egy üzlet ajtaján.<sup>18</sup> Ez azt az információt közvetíti, hogy az üzlet pillanatnyilag fogad vevőket. Most vegyük ugyanezt a szót egy újronnan felépített üzleten található hatalmas transzparensten. Ez azt jelenti, hogy az üzlet megnyílt, de az olvasók nem éreznék becsapva magukat, ha éjszaka a transzparenst eltávolítása nélkül lenne zárva a bolt. A két felirat ugyanazt a szót használja különböző jelentések közvetítésére. Német nyelvű országban a felirat az ajtón „Offen”, míg a transzparensten „Neu eröffnet” lenne.<sup>19</sup>

Az a probléma, hogy az egyes nyelvek eltérő módon kategorizálják a világot. Például a francia „doux” szó jelentések széles körét fogja át, amely megközelítőleg a következő angol szavakkal adható meg: „soft”, „sweet”, „gentle”.<sup>20</sup> Hasonlóképpen az angol „hard” szó gyakorlatilag a német „hart” szó összes jelentését (fizikailag ellenálló, durva) lefedи, emellett a „schwierig” („nehéz”, fáradtságos, bonyolult értelemben) szó néhány jelentését is magában foglalja. A német „heilen” (meggyógyít) ige az angol „cure” (gyógyít) szó orvosi használatát fedi le, továbbá a „heal” (meggyógyul) tranzitív és intranzi-tív jelentéseit. Emiatt az adott mondat jelentésének ábrázolása sokkal nehezebb a fordítás, mint az egynyelvi megértés esetén. Egy egynyelvű elemzőrendszer használhatna olyan predikátumokat, mint *Open(x)*, azonban a fordítás céljából a reprezentációs nyelvnek esetleg több megkülönböztetést kellene tennie, például az „Offen” jelentést reprezentáló *Open<sub>1</sub>(x)* predikáttummal és a „Neu eröffnet” jelentést reprezentáló *Open<sub>2</sub>(x)* predikáttummal. Azt a reprezentációs nyelvet, amely a lefedett nyelvekhez szükséges összes megkülönböztetést lehetővé teszi, köztes nyelvnek (*interlingua*) nevezzük.

Folyékony fordításhoz az szükséges, hogy a fordító (ember vagy gép) elolvassa az eredeti szöveget, megértsse a szituációt, amiről szól, és találjon egy megfelelő szöveget a célnyelvben, amely jól leírja ugyanazt, vagy egy hasonló szituációt. Ez gyakran választást jelent. Például amennyiben az angol „you” szó egy személyre vonatkozik, akkor vagy a magyar hivatalos „ön” vagy a közvetlen „te” alakra fordítható. Egyszerűen nem lehet a „you” fogalomról beszélni magyarul anélkül, hogy előntenénk, hogy hivatalos vagy közvetlen módon szeretnénk használni. A fordítók (mind a gépi, mind az emberi) néha nehezen tudják ezt a döntést meghozni.

## Gépi fordító rendszerek

A gépi fordító rendszerek erősen különböznek abban, hogy milyen szinten elemzik a szöveget. Egyes rendszerek a bemeneti szöveget egészen a köztes nyelv szintjéig próbál-ják elemezni (mint ahogy azt a 22. fejezetben tettük), majd a célnyelven ezen reprezen-táció alapján generálnak mondatokat. Ez azért nehéz, mert részproblémaként tartalmazza a teljes nyelvi megértés problémáját, amihez még a köztes nyelv kezeléséből fakadó nehézségek is hozzáadódnak. A megközelítés azért törékeny, mert ha az elemzés siker-telen, akkor nincs kimenet. Azonban az az előnye, hogy a rendszerben nincs olyan kom-

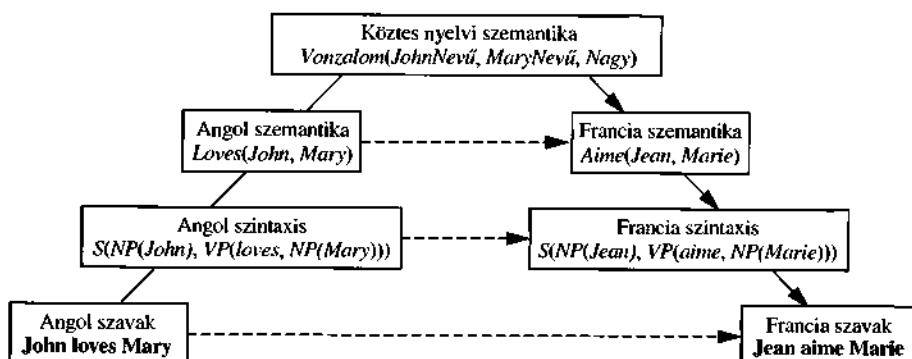
<sup>18</sup> Martin Kay példája.

<sup>19</sup> Nyitva – Kinyitottunk (*A ford.*)

<sup>20</sup> doux – édes, enyhe, lágy, soft – lágy, sweet – édes, gentle – finom, gyengéd (*A ford.*)

ponens, amelynek két (természetes) nyelvet egyszerre kell ismernie. Ez azt jelenti, hogy köztes nyelv használatával  $n$  nyelv közti fordítás  $O(n)$  és nem  $O(n^2)$  nehézségű.

Más rendszerek az átvitelen (transfer) alapulnak. Fordítási szabályok (vagy példák) adatházisát tartalmazzák, és amikor egy szabály (vagy példa) illeszkedik, akkor közvetlenül fordítanak. Az átvitel lexicai, szintaktikai vagy szemantikai szinten történhet. Például egy szigorúan szintaktikai szabály az angol [Melléknév Főnév] szekvenciát a francia [Főnév Melléknév] szekvenciába képezi le. Egy kevert szintaktikai és lexikai szabály a francia [ $S_1$  „et puis”  $S_2$ ] szekvenciát az angol [ $S_1$  „and then”  $S_2$ ] szekvenciába képezi le.<sup>21</sup> Azt az átvitelt, amely egy mondatot közvetlenül egy másikba visz át, memóriaalapú fordításnak (memory-based translation) nevezünk, mivel nagyszámú (angol, francia) pár memorizálására támaszkodik. Az átviteli módszer robusztus, mert minden esetben generál valamilyen kimenetet, és legalább a szavak egy része feltétlenül helyes. A 23.5. ábra bemutatja az egyes átviteli pontokat.



23.5. ábra. Egy gépi fordító rendszer választási lehetőségeit ábrázoló sematikus diagram. A fenn található angol szöveggel kezdjük. A köztes nyelven alapuló rendszer a folytonos vonalakat követi, az angol szöveget először szintaktikai elemzések veti alá, majd szemantikai és köztes nyelvi reprezentációt állít elő, végül szemantikai, szintaktikai és lexikai formákon keresztül francia szöveget állít elő. Az átvitel-alapú rendszerek a szaggatott vonalak által jelölt rövidzárakat használják. Az egyes rendszerek eltérő szinteken végezik az átvitelt, egyes rendszerek több szinten is.

## Statisztikai gépi fordítás

Az 1960-as évek elején nagy reményeket fűztek ahhoz, hogy a számítógépek képesek egyik természetes nyelvből a másikba fordítani csakúgy, mint ahogy Turing projektje képes volt kódolt német üzeneteket értelmes német szövegbe fordítani. 1966-ra világossá vált, hogy a folyékony fordítás igényli az üzenet jelentésének a megértését, míg a kódfejtés nem.

Az elmúlt évtizedben elmozdulás volt megfigyelhető a statisztikai alapú gépi fordító rendszerek irányába. Természetesen, a 23.5. ábra bármely lépéseként javára szolgálnak statisztikai adatok, valamint egy olyan egyértelmű valószínűségi modell használata,

<sup>21</sup> Mindkét esetben magyarul [ $S_1$  „és utána”  $S_2$ ] (A ford.)

amely megadja, hogy mi egy jó analízis vagy átvitel. Azonban a „statisztikai gépi fordítás” az egész fordítási probléma olyan megközelítésének a megnevezésévé vált, amely a mondat legvalószínűbb fordításának kétnyelvű korpuszon alapuló megtalálását jelenti. A kétnyelvű korpuszok egyik példája a *Hansard*,<sup>22</sup> amely parlamenti viták naplója. Kanada, Hongkong és más országok kétnyelvű Hansardokat tesznek közzé, az Európai Unió 11 nyelven<sup>23</sup> publikálja hivatalos dokumentumait, míg az Egyesült Nemzetek Szervezete többnyelvű dokumentumokat ad ki. Ezek a statisztikai gépi fordítás számára felbecsülhetetlenül értékes forrásoknak bizonyultak.

Egy angol nyelvű mondat ( $E$ ) például francia<sup>24</sup> ( $F$ ) mondatra történő fordításának problémáját a Bayes-szabály következő alkalmazásaként írhatjuk le:

$$\begin{aligned} \operatorname{argmax}_F P(F|E) &= \operatorname{argmax}_F P(E|F)P(F)/P(E) \\ &= \operatorname{argmax}_F P(E|F)P(F) \end{aligned}$$

Ez a szabály azt állítja, hogy minden lehetséges  $F$  francia mondatot figyelembe kell vennünk, és azt kell választanunk, amelyik maximalizálja  $P(E|F)P(F)$ -t. A  $P(E)$  tényezőt nem kell figyelembe venni, hiszen minden  $F$  esetén azonos. A  $P(F)$  tényező a francia **nyelvi modell (language model)**, azt adja meg, hogy milyen valószínűségű egy adott francia mondat. A  $P(E|F)$  a **fordítási modell (translation model)**, azt adja meg, hogy milyen valószínűségű egy adott angol mondat, mint az adott francia mondat fordítása.

Az ügyes olvasók bizonyára csodálkoznak, hogy mit nyertünk a  $P(F|E)$  definíálásával a  $P(E|F)$  segítségével. A Bayes-szabály más alkalmazásainál azért tettük ezt, mert kauzális modellt akartunk használni. Például a  $P(\text{Szimptómák}|\text{Betegség})$  kauzális modellt használtuk  $P(\text{Betegség}|\text{Szimptómák})$  kiszámolására. A fordítás esetében azonban egyik irány sem kauzálisabb, mint a másik. Jelen esetben a Bayes-szabály alkalmazásának az az oka, hogy azt hisszük, képesek leszünk egy olyan  $P(F)$  nyelvi modellt megtanulni, amely pontosabb, mint a  $P(E|F)$  fordítási modell (és pontosabb, mint  $P(F|E)$  közvetlen becslése). Lényegében a problémát két részre osztottuk: először a  $P(F|E)$  fordítási modellt alkalmazzuk, hogy olyan francia mondatokat találunk, amelyek visszaadják az angol mondat lényegét, de amelyek nem feltétlenül folyékony francia mondatok, majd a  $P(F)$  nyelvi modellt (amelyre sokkal jobb valószínűségi becslésünk van) használjuk a legjobb jelölt kiválasztására.

A  $P(F)$  **nyelvi modell** bármilyen olyan modell lehet, amely egy mondathoz valószínűséget rendel. Nagyon nagy korpusz esetén  $P(F)$ -et közvetlenül becsülhetnénk az egyes mondatok korpuszbeli előfordulási száma alapján. Például ha a webről összegyűjtünk 100 millió francia mondatot, és a „*Clique ici*”<sup>25</sup> mondat 50 ezerszer fordul elő, akkor  $P(\text{Clique ici}) = 0,0005$ . Azonban még 100 millió példa esetén is a legtöbb mondat előfordulási száma nulla lenne.<sup>26</sup> Emiatt az ismert bigram modellt fogjuk alkalmazni, amelyben az  $f_1 \dots f_n$  szavakból álló francia nyelvű mondat valószínűsége:

<sup>22</sup> William *Hansard* után elnevezve, aki 1811-ben elsőként publikálta a brit parlamenti vitákat.

<sup>23</sup> Az angol nyelvű kiadás idején. (*A ford.*)

<sup>24</sup> Ebben a bekezdésben az angolról francia nyelvre történő fordítás problémájával foglalkozunk. Ne zavarja az olvasót, hogy a Bayes-szabály alkalmazása a  $P(E|F)$ , és nem a  $P(F|E)$  figyelembe vételéhez vezet, ami annak tönök, mintha franciáról fordítanánk angolra.

<sup>25</sup> Kattints ide (*A ford.*)

<sup>26</sup> Amennyiben csak 100 ezer szó fordulna elő a lexikonban, akkor az összes lehetséges háromszavas mondatok 99,9999%-ának előfordulási száma nulla lenne a 100 milliós korpuszban. Hosszabb mondatok esetén még rosszabb a helyzet.

$$P(f_1 \dots f_n) = \prod_{i=1}^n P(f_i | f_{i-1})$$

Ismertünk kell az olyan bigram valószínűségeket, mint például a  $P(\text{Eiffel}|\text{tour}) = 0,02$ .<sup>27</sup> Ez a szintaxis mindenkor nagyon lokális jellemzőit képes leírni, ahol a szó csak az elő megelőző szótól függ. A nyersfordításhoz azonban többnyire ez is elegendő.<sup>28</sup>

A  **$P(E|F)$  fordítási modellt (translation model)** nehezebb meghatározni. Egyrészt nem áll rendelkezésünkre (angol, francia) mondatpárokból álló kész gyűjtemény, amely alapján taníthatnánk. Másrészt a modell komplexitása nagyobb, mivel mondatok kereteszszorozatára, és nem pedig különálló mondatokra alapul. Egy túlzottan leegyszerűsített fordítási modellel fogunk kezdeni, és felépítünk valamit, ami az „IBM Model 3”-at (Brown és társai, 1993) közelíti, ami továbbra is a végletekig leegyszerűsítettnek tűnik, azonban az esetek körülbelül felében elfogadható fordításokat generált.

A végletekig leegyszerűsített modell arról szól, hogy „a mondat fordításához egyszerűen forditsuk le a szavakat egyesével és egymástól függetlenül, balról jobbra”. Ez egy unigram szóválasztási modell. Lehetővé teszi, hogy egyszerűen kiszámítsuk egy fordítás valószínűségét:

$$P(E|F) = \prod_{i=1}^n P(E_i | F_i)$$

Néhány esetben ez a modell jól működik. Vegyük például a következőt:

$$P(\text{the dog}| \text{le chien}) = P(\text{the}| \text{le}) \times P(\text{dog}| \text{chien})$$

Bármely elfogadható valószínűségi érték halmaz esetén a „the dog”<sup>29</sup> lenne a „le chien” maximum-likelihood becslője. A legtöbb esetben azonban a modell megbukik. Az egyik probléma a szórend. Franciában a „dog” megfelelője a „chien”, a „brown”-é a „brun”, azonban a „brown dog”-é a „chien brun”.<sup>30</sup> A másik probléma az, hogy a szóválasztás nem egy az egyes leképezés. Az angol „home” szót gyakran „à la maison”-nak<sup>31</sup> fordítják, ami egy hármas leképezés (illetve három az egybe a másik irányban). Ezen problémák ellenére az IBM Model 3 makacsul ragaszkodik az alap unigram modellhez, bár, hogy javítson rajta, hozzátesz néhány kiegészítést.

A modell, hogy képes legyen kezelní azt a tényt, hogy a szavakat nem egy az egyben fordítjuk, bevezeti a szó **termékenységének (fertility)** fogalmát. Egy  $n$  termékenységű szót  $n$ -szer lemásolja, és az  $n$  másolat mindenike függetlenül fordítódik. A modell az összes francia szóra tartalmazza a  $P(\text{termékenység} = n | \text{szó})$  paramétereket. Az „à la maison” „home”-ra történő fordításához a modell 0 termékenységet választana az „à” és „la” szavakra,<sup>32</sup> míg egyet a „maison”-ra, és utána az unigram modellt alkalmazná a „maison” szó „home”-ra történő fordításához. Ez eléggyé indokoltanak tűnik, az „à” és a „la” kis információtartalmú szavak, amelyeket ésszerű üres karakterfüzérre fordítani.

<sup>27</sup> Eiffel-torony (*A ford.*)

<sup>28</sup> Világos, hogy a fordítás finomabb részleteihez  $P(f_i | f_{i-1})$  nem elegendő. Híres példaként álljon itt Marcel Proust 3500 oldalas regénye, a „A la recherche du temps perdu”, amely ugyanazzal a szóval kezdődik és végződik, így egyes fordítók úgy döntöttek, hogy ugyanezt teszik, hogy egy szó fordítását egy olyan szóval alapozták, amely durván 2 millió szóval korábban fordult elő.

<sup>29</sup> a kutya (*A ford.*)

<sup>30</sup> barna kutya (*A ford.*)

<sup>31</sup> home – otthon, à la maison – otthon lenni, hazára menni értelemben (*A ford.*)

<sup>32</sup> „à” – nagyjából a magyar -ban, -ben, -ba, -be tagoknak felel meg, „la” – röviden határozott névelő (*A ford.*)

A másik irányba történő fordítás már kétségesebb. A „home” szóhoz hármas termékenységet rendelne a modell, „home home home” szekvenciát kapva. Az első „home” „à”-ra, a második „la”-ra, míg a harmadik „maison”-ra fordulna. A fordítási modell szempontjából az „à la maison” és a „maison à la” pontosan azonos valószínűséget kapna. (Pont ez a kétséges rész.) A nyelvi modellre lenne bízva, hogy eldöntse, melyik a jobb. Értelmesebbnek tűnhet a „home”-ot közvetlenül „à la maison”-ra fordítani, mint közvetett módon a „home home home”-on keresztül, azonban ehhez sokkal több paraméterre lenne szükség, amelyeket nehéz lenne a rendelkezésre álló korpuszból meg-határozni.

A fordítási modell végső része a szavak megfelelő sorrendbe történő permutálása. Ez egy eltolási modellel történik, amely során a szó az eredeti pozíciójából a végleges pozícióba mozog. Például a „chien brun” „brown dog”-ra történő fordítása során a „brown” szó +1 eltolási értéket kap (azaz jobbra egy pozíciót mozog), míg a „dog” –1 értéket. Az olvasó elképzelheti, hogy az eltolásnak függnie kellene a szótól: az olyan melléknevek, mint a „brown” többnyire pozitív eltolási értékkel rendelkeznének, mert a francia nyelv általában a főnév után helyezi el a mellékneveket. Azonban az IBM Model 3 úgy döntött, hogy a szótól független a szótól, és csak a mondatbeli pozíciótól és a mondat mindenbeli nyelvbeli hosszától függ. Azaz a modell a következő paramétereket becsüli:

$$P(\text{Eltolás} = o | \text{Pozíció} = p, \text{AngolHossz} = m, \text{FranciaHossz} = n)$$

Azaz a „brown” szó „brown dog”-beli eltolásának meghatározásához megnézzük a  $P(\text{Eltolás}|1,2,2)$  értékét, amely mondjuk +1 értéket adna 0,3, és 0 értéket 0,7 valószínűséggel. Az eltolási modell még kétségesebbnek látszik, olyan, mintha egy olyan személy agyalta volna ki, aki sokkal jobban ért a mágnesbetűk hűtőn történő tologatásához, mint a valódi természetes nyelven történő beszédehez. Rövidesen bemutatjuk, hogy nem azért terveztek ilyen módon, mert jól modellezik a természetes nyelvet, hanem azért, mert a rendelkezésre álló adatokat ésszerűen hasznosítja. Mindenesetre arra szolgál, hogy élénken emlékeztessen bennünket arra, hogy egy középszerű fordítási modellt meg lehet menteni egy jó francia nyelvi modellel. Íme, egy példa egy mondat fordításának lépéseire.

Forrás francia:	Le	chien	brun	n'	est	pas	allé	à	la	maison	<sup>33</sup>
Termékenységi modell:	1	1	1	1	1	0	1	0	0	1	
Transzformált francia:	Le	chien	brun	n'	est		allé			maison	
Szóválasztási modell	The	dog	brown	not	did		go			home	
Eltolási modell:	0	+1	-1	+1	-1		0			0	
Cél angol:	The	brown	dog	did	not		go			home	

Most már tudjuk, hogyan kell kiszámítani a  $P(F|E)$  valószínűséget bármely (francia, angol) mondatpárra. Azonban amit igazából tenni akarunk az az, hogy egy adott angol mondat esetén megtaláljuk azt a francia mondatot, amely maximalizálja ezt a valószínűséget. Nem sorolhatunk fel egyszerűen mondatokat:  $10^5$  francia szó esetén  $10^{5n}$  hosszúságú mondat lehetséges, és számos összerendelés mindegyikre. Még ha csak a 10 leggyakoribb szó-szó fordítást tekintjük minden egyes szóra, és csak 0 és  $\pm 1$  eltolást veszünk figyelem-

<sup>33</sup> A barna kutya nem ment haza. (A ford.)

be, akkor is  $2^n/2 \cdot 10^n$  mondatot kapunk, ami azt jelenti, hogy  $n = 5$  esetén még fel tudnánk őket sorolni, de  $n = 10$  esetén már nem. Ehelyett meg kell *keresnünk* a legjobb megoldást. Az  $A^*$  keresés hatékonyak bizonyult a feladatra (Germann és társai, 2001).

## Valószínűségek tanulása gépi fordításhoz

Körvonalaztunk egy olyan  $P(F|E)$  modellt, amely négy paraméterhalmazt tartalmaz:

Nyelvi modell:	$P(\text{szó}_i \text{szó}_{i-1})$
Termékenységi modell:	$P(\text{Termékenység} = n \text{szó}_F)$
Szóválasztási modell:	$P(\text{szó}_E \text{szó}_F)$
Eltolási modell:	$P(\text{Eltolás} = o \text{pozíció}, \text{hossz}_E, \text{hossz}_F)$

Ennek a modellnek még egy 1000 szót tartalmazó, szerény szókincs esetén is paramétereik millióira van szüksége. Nyilvánvalóan adatokból kell megtanulnunk őket. Feltételezzük, hogy az egyetlen rendelkezésünkre álló adat egy kétnyelvű korpusz. Lássuk a felhasználási módját:

**Mondatokra történő szegmentáció:** a fordítási egység a mondat, tehát a korpuszt mondatokra kell tördelnünk. A pont a mondatvég jellemző indikátora, de vegyük a „Dr. J. R. Smith of Rodeo Dr. megérkezett.” példát, amelyben csak az utolsó pont jelenti a mondat végét. A mondatokra történő szegmentációt körülbelül 98%-os pontossággal lehet elvégezni.

**A  $P(\text{szó}_i|\text{szó}_{i-1})$  francia nyelvi modell becslése:** vegyük a korpusz francia nyelvű részét, számoljuk meg a szópárok előfordulási gyakoriságát, és végezzük simítást, hogy megkapjuk a  $P(\text{szó}_i|\text{szó}_{i-1})$  becslójét. Például megkaphatjuk, hogy  $P(\text{Eiffel|tour}) = 0,02$ .

**Mondatok illesztése:** minden egyes angol nyelvű mondatra határozzuk meg az(oka)t a francia nyelvű mondat(ok)a)t, amely(ek) megfelel(nek) neki a francia változatban. Általában a szövegben következő angol mondat megfelel a következő francia mondatnak 1:1 megfeleltetéssel, azonban néha előfordulnak variációk: az egyik nyelvű mondatot 2:1 megfeleltetésbe vágjuk, vagy két mondat sorrendjét felcséréljük, 2:2 megfeleltetést kapva. Mindössze a mondathosszakat tekintve lehetséges az illesztésük (1:1, 1:2, 2:2 stb.) 90% és 99% közötti pontossággal, a Viterbi szegmentációs algoritmus (lásd 23.1. ábra) megfelelő variációjával. Még jobb illeszkedés is elérhető amennyiben olyan iránypontokat használunk, amelyek minden két nyelvben közösek, mint például számok, tulajdonnevek, illetve olyan szavak, melyeknek – egy kétnyelvű szótár alapján – tudjuk, hogy egyértelmű a fordításuk.

**A  $P(\text{Termékenység} = n|\text{szó}_F)$  kezdeti termékenységi modell becslése:** legyen adott egy  $m$  hosszúságú francia mondat, amely egy  $n$  hosszúságú angol mondathoz illeszkedik, ekkor vegyük ezt annak alátámasztásaként, hogy mindegyik (a mondatban előforduló) francia szó termékenysége  $n/m$ . Vagyuk figyelembe ezeket az összes mondatra az egyes szavak termékenységi eloszlásának meghatározásához.

**A  $P(\text{szó}_E|\text{szó}_F)$  kezdeti szóválasztási modell becslése:** vegyük az összes olyan mondatot, amely tartalmazza például a „brun” szót. Azok a szavak, amelyek a leggyakrabban fordulnak elő az illeszkedő angol mondatokban, valószínűleg a „brun” szó-szó fordításai.

**A  $P(\text{Eltolás} = o|\text{pozíció}, \text{hossz}_E, \text{hossz}_F)$  kezdeti eltolási modell becslése:** most, hogy rendelkezünk egy szóválasztási modellel, használjuk fel azt az eltolási modell becslésére. Egy  $n$  hosszúságú angol mondat esetén, amely egy  $m$  hosszúságú francia

mondathoz illeszkedik, vegyük minden egyes francia szót ( $i$  pozícióban) és minden olyan angol szót ( $j$  pozícióban), amely valószínű választás lenne a francia szóra, és vegyük ezt  $P(\text{Eltolás} = i - j|i, n, m)$  bizonyítékául.

**A becslések javítása:** használjuk az EM (elvárásmaximalizáló) algoritmust a becslések javítására. A rejtett változó az illeszkedő mondatpárok közti **szóilleszkedési vektor** (**word alignment vector**). A vektor megadja minden egyes angol szóra a megfelelő francia szó francia mondatbeli pozícióját. Például megkaphatjuk a következőt:

Forrás francia:	Le	chien	brun	n'	est	pas	allé	à	la	maison
Cél angol:	The	brown	dog	did	not		go			home
Szóilleszkedés:	1	3	2	5	4		7			10

Először a paraméterek aktuális becslőit használva elkészítjük a szóilleszkedési vektort minden egyik mondatpárra. Ez jobb becslő megalkotását fogja számunkra lehetővé tenni. A termékenységi modellt az alapján becsülik, hogy a szóilleszkedési vektor egy adott tagja hányszor képződik le több szóra, vagy egyetlenegyre sem. A szóválasztási modellnek a mondat összes szava helyett most már csak olyan szavakat kell figyelembe vennie, amelyek egymáshoz lettek illesztve. Az eltolási modell a mondat egyes pozícióinak elmozdulását vizsgálja a szóilleszkedési vektornak megfelelően. Sajnos nem tudjuk, hogy mi a helyes illeszkedés, és túl sok van belőlük ahhoz, hogy minden felsoroljuk. Emiatt arra vagyunk kényszerítve, hogy megkeressük néhány nagy valószínűségű illeszkedést, és a valószínűségekkel súlyozzuk őket, miközben bizonyítékot gyűjtünk az új paraméterbecslőkhöz. Mindössze ennyire van szükségünk az EM algoritmushoz. A kezdeti paraméterek alapján illeszkedéseket számolunk, majd az illeszkedések alapján javítjuk a paraméterbecslőket. Ismétljük, amíg konvergál.

## 23.5. ÖSSZEFOGLALÁS

A fejezet főbb pontjai a következők:

- Az  $n$ -gram alapú valószínűségi nyelvi modellek meglepő mennyiséggű információt képesek visszaadni egy nyelvről.
- A CFG-keket ki lehet egészíteni valószínűségi CFG-kre, könnyebbé téve az adatok alapján történő tanulásukat és a kétértelműségek feloldását.
- Az **információkereső** (**information retrieval**) rendszerek egy nagyon egyszerű, szózsákakra épülő nyelvi modellt használnak, ennek ellenére jó **felidézési** (**recall**) és **pontossági** (**precision**) mutatókkal rendelkeznek nagyon nagy korpuszokon.
- Az **információkinyerő** (**information extraction**) rendszerek bonyolultabb modellt alkalmaznak, amely korlátozottan figyelembe veszi a szintaxist és a szemantikát. Gyakran véges automaták kaszkádjával valósítják meg őket.
- A **gépi fordító** (**machine translation**) rendszereket különböző technikák alkalmazásával valósították meg, a teljes szintaktikai és szemantikai analízistől kezdve egészen a szógyakoriságon alapuló statisztikai módszerekig.
- Egy statisztikai nyelvi rendszer építése során az a legjobb, ha olyan rendszert gondolunk ki, amely a rendelkezésre álló adatokat jól hasznosítja, még akkor is, ha a modell a végletekig leegyszerűsítettnek tűnik.

## Irodalmi és történeti megjegyzések

Az *n*-gram betűmodelleket nyelvi modellezésre Markov javasolta (Markov, 1913). Claude Shannon generált elsőként *n*-gram szómodelleket az angol nyelvre (Shannon és Weaver, 1949). Chomsky mutatta meg a véges állapotú modellek korlátait a környezetfüggetlen modellekhez képest, az alábbi következtetést levonva: „Valószínűsgégi modellek nem adnak részletes bepillantást a szintaktikai struktúrák egyes alapvető problémáiba” (Chomsky, 1956; 1957). Ez igaz, azonban figyelmen kívül hagyja azt a tényt, hogy a valószínűsgégi modellek *lehetővé teszik* a bepillantást néhány más alapvető problémába, olyanokba, amelyekkel a CFG-k nem foglalkoznak. Chomsky észrevételeinek olyan sajnálatos hatása volt, hogy két évtizeden keresztül sokat elijesztett a statisztikai modellektől egészen addig, míg ezek a modellek újra megjelentek a beszédfelismerésben (Jelinek, 1976).

Az adj-hozzá-egyet simítás Jeffreystől származik (Jeffreys, 1948), míg a törölt interpolációs simítás Jelinektől és Mercertől, akik beszédfelismerésre használták (Jelinek és Mercer, 1980). További technikákra példa a Witten–Bell-simítás (Witten–Bell, 1991) és a Good–Turing-simítás (Church és Gale, 1991). Az utóbbit gyakran használják bioinformatikai problémáknál is. A biostatisztika és a valószínűsgégi természetes nyelvfeldolgozás (NLP) közelednek egymáshoz, mivel minden alkotóelemek ábécéjéből felépülő hosszú, strukturált szekvenciákkal foglalkozik.

Az egyszerű *n*-gram betű- és szómodellek nem az egyetlen lehetséges valószínűsgégi modellek. Blei és társai leírják a **rejtett Dirichlet-allokáció**nak (*latent Dirichlet allocation*) nevezett valószínűsgégi szövegmodellt, amely a szövegeket téma keveréknél tekinti, amelyben minden egyik téma saját szóeloszlással rendelkezik (Blei és társai, 2001). Ez a modell a **rejtett szemantikai indexelés** (*latent semantic indexing*) modell (Deerwester és társai, 1990) (lásd még (Papadimitriou és társai, 1998)) kibővítésének és racionalizálásának tekinthető, valamint Sahami és társai többszörös okkeverék (*cause mixture*) modelljéhez is kötődik (Sahami és társai, 1996).

A valószínűsgégi környezetfüggetlen nyelvtanok (PCFG) megválaszolják Chomsky valószínűsgégi modellekkel kapcsolatos összes ellenvertését, és a CFG-khez képest előnyökkel rendelkeznek. A PCFG-ket Booth (Booth, 1969) és Salomaa (Salomaa, 1969) vizsgálták. Jelinek bemutatja a veremdekkódoló algoritmust, amely a Viterbi-keresés olyan variációja, amely arra használható, hogy megtalálja egy PCFG-vel a legvalószínűbb elemzést (Jelinek, 1969). Baker vezette be a belső–külső algoritmust (Baker, 1979), Lari és Young leírta használhatóságát és korlátait (Lari és Young, 1990). Charniak (Charniak, 1996), valamint Klein és Manning (Klein és Manning, 2001) a treebank nyelvtanokkal történő elemzést tárgyalják. Stolcke és Omohundro megmutatták, hogyan lehet nyelvtani szabályokat tanulni Bayes-modellek egyesítésével (Stolcke és Omohundro, 1994). További PCFG-algoritmusokat mutatott be Charniak (Charniak, 1993), valamint Manning és Schütze (Manning és Schütze, 1999). Collins a terület áttekintő tanulmányát kínálja, továbbá az egyik legsikeresebb statisztikai elemzőprogram magyarázatát (Collins, 1999).

Sajnos a PCFG-k rosszabbul teljesítenek, mint az egyszerű *n*-gram modellek számos feladat esetén, mert a PCFG-k nem képesek reprezentálni az egyes szavakhoz társuló információkat. Hogy kijavítsák ezt a hiányosságot, számos szerző (Collins, 1996; Charniak, 1997; Hwa, 1998) bevezették a **szókincssel ellátott valószínűsgégi nyelvtanok** (lexica-

lized probabilistic grammar) különböző verziót, amelyek kombinálják a környezetfüggetlen és a szóalapú statisztikákat.

A Brown-korpusz volt az első próbálkozás, hogy tapasztalati nyelvészeti célokra kiegyensúlyozott korpuszt gyűjtsön (Francis és Kucera, 1967). Körülbelül egymillió szót tartalmazott, szófaji információval ellátva. Eredetileg 100 ezer lyukkártyán tárolták. A Penn treebank körülbelül 1,6 millió szó gyűjteménye, kézi elemzéssel fákba rendezve. Elfér egy CD-n. A brit nemzeti korpusz (British National Corpus) kibővíti ezt 100 millió szóra (Leech és társai, 2001). A világháló több mint egybillió szót tartalmaz, több mint 10 millió szerveren tárolódik.

Az **információkeresés (informational retrieval)** területe iránti érdeklődés újra nő, amelyet az internetes keresés széles körű elterjedése inspirál. Robertson egy korai áttekintést ad, és vezeti a valószínűségi rangsorolási elvet (Robertson, 1977). Manning és Schütze az NLP statisztikai megközelítésének kontextusában tárgyalja röviden az IR-t (Manning és Schütze, 1999). Baeza-Yates és Ribeiro-Neto általános célú áttekintést ad (Baeza-Yates és Ribeiro-Neto, 1999), helyettesítve az olyan régi klasszikusokat, mint Salton és McGill (Salton és McGill, 1983), valamint Frakes és Baeza-Yates (Frakes és Baeza-Yates, 1992). A *Gigabajtok kezelése* (Managing Gigabytes) c. könyv pont azt teszi, amit a címe mond: elmagyarázza hogyan lehet hatékonyan indexelni, tömöríteni és lekérdezni gigabajtos méretű korpuszokat (Witten és társai, 1999). Az amerikai kormány Nemzeti Szabványügyi és Technológiai Intézete (National Institute of Standards and Technology, NIST) által szervezett TREC konferencia minden évben megrendezi az IR-rendszerek versenyét, és kiadványában publikálja az eredményeket. A verseny első hét évében a részt vevő rendszerek teljesítménye körülbelül megduplázódott.

A legnépszerűbb IR-modell a **vektortér modell (vector space model)** (Salton és társai, 1975). Salton munkája uralta a terület korai éveit. Két alternatív valószínűségi modell létezik. Mi a Ponte és Croft munkáján alapulót mutattuk be (Ponte és Croft, 1998). A  $P(D, Q)$  közös valószínűségi eloszlást  $P(Q|D)$  segítségével modellezte. Egy alternatív modell (Maron és Kuhns, 1960; Robertson és Sparck Jones, 1976) a  $P(D|Q)$ -t használja. Lafferty és Zhai megmutatta, hogy a modellek ugyanazon a közös valószínűségi eloszláson alapulnak, azonban a modellválasztásnak következményei vannak a paramétertanításra (Lafferty és Zhai, 2001). Az általunk bemutatottak az ő munkájukon alapul. Turtle és Croft összehasonlítják a különféle IR-modelleket (Turtle és Croft, 1992).

Brin és Page leírják egy világhálón kereső gép implementációját, taglalva a PAGERANK algoritmust is, mely a dokumentumok minőségének weblinkek elemzésén alapuló lekérdezésfüggetlen mértéke (Brin és Page, 1998). Kleinberg leírja, hogyan lehet hiteles forrásokat találni a weben hivatkozáselemezéssel (Kleinberg, 1999). Silverstein és társai megvizsgáltak egy egymilliárd webkeresést tartalmazó logot (Silverstein és társai, 1998). Kukich felmérte a helyesírási hibák javításának irodalmát (Kukich, 1992). Porter leírja a klasszikus szabályalapú szótövesítő algoritmust (Porter, 1980), míg Krovetz egy szótáralapú verziót ír le (Krovetz, 1993).

Manning és Schütze a dokumentumosztályozás és klaszterezés alapos áttekintését adják (Manning és Schütze, 1999). Joachims statisztikai tanulási elméletet és szupport vektor gépeket alkalmazott annak elméleti vizsgálatára, hogy mikor sikeres egy osztályozás (Joachims, 2001). Apté és társai a Reuters-hírek „Jövedelmek” kategóriába történő osztályozásánál 96%-os pontosságot publikált (Apté és társai, 1994). Koller és

Sahami 95%-os pontosságot ért el egy naiv Bayes-osztályozóval, és akár 98,6%-ot is, ha olyan Bayes-osztályozót alkalmazott, amely bizonyos jellemzők közti összefüggéseket is figyelembe vett (Koller és Sahami, 1997). Lewis áttekinti a Bayes-osztályozók negyvenéves alkalmazását szövegesztályzásra és visszakeresésre (Lewis, 1998).

Az *Information Retrieval* újság és az éves *SIGIR* konferenciakiadványai a terület friss fejleményeit fedik le.

A korai információkinyerő programok közé tartozik a **GUS** (Bobrow és társai, 1977) és a **FRUMP** (DeJong, 1982). A modern információkinyerő rendszerek egyes tervezési részletei az 1970–1980-as évekbeli szemantikai nyelvtanokkal foglalkozó kutatásokra vezethetők vissza. Például egy szemantikai nyelvtanokon alapuló repülőgép-foglaló rendszer interfésze olyan kategóriákkal rendelkezne, mint a *Hely* és a *Repül*, az *NP*-k és a *VP*-k helyett. Birnbaum és Selfridge munkája egy szemantikai nyelvtanon alapuló rendszer megvalósítását mutatja be (Birnbaum és Selfridge, 1981).

Az újabb keletű információkinyeréssel foglalkozó munkákat az amerikai kormány által szponzorált Message Understand Conference (MUC) konferenciák mozzatták. A FASTUS-rendszert Hobbs és társai készítették (Hobbs és társai, 1997), és az a publikációgyűjtemény, amely bemutatja, további véges állapotú modellek alkalmazó rendszereket is felsorol (Roche és Schabes, 1997).

Az 1930-as években Petr Troyanskii beadott egy „fordítógépre” vonatkozó szabadalomkérelmet, azonban még nem állt rendelkezésre számítógép ötleteinek megvalósítására. 1947 márciusában a Rockefeller Alapítványnak dolgozó Warren Weaver az írta Norbert Weinernek, hogy a gépi fordítás megvalósítható lehet. A kriptográfiai és információelméleti munkákra alapozva Weaver azt írta, hogy: „Amikor egy orosz nyelvű cikket nézek, azt mondom, hogy »Ezt igazából angolul írták, azonban furcsa szimbólumokkal kódolták. Most pedig nekiállok a dekódolásnak.«” A következő évtizedben a közösség ilyen módon próbálta a dekódolást. Az IBM 1954-ben bemutatott egy kezdetleges rendszert. Bar-Hillel (Bar-Hillel, 1960), valamint Locke és Booth (Locke és Booth, 1955) leírták az ebben az időszakban meglévő lelkesedést. A gépi fordításból való későbbi kiábrándulást Lindsay írta le, aki a gépi fordítás néhány olyan akadályát is bemutatta, amelyek a szintaxis és a szemantika közti kölcsönhatáshoz és a világ ismeretének szükségességehez kötődnek (Lindsay, 1963). Az amerikai kormány csalódott az előrehaladás hiánya miatt, ezt egy jelentés akként összegezte, hogy „a gépi fordításnak nincs közvetlen vagy megjósolható jövője” (ALPAC, 1966). Ennek ellenére korlátozottan ugyan, de folytatódott a munka, és az amerikai légiérő 1970-ben „hadrendbe állította” a SYSTRAN-rendszert, míg az Európai Közösség 1976-ban kezdte azt használni. A TAUM-METEO időjárásjelentés-fordító rendszer használata szintén 1976-ban indult (Quinlan és O'Brien, 1992). Az 1980-as években a rendelkezésre álló számítási teljesítmény eljutott arra a szintre, ahol az ALPAC megállapításai már nem voltak helyesek. Voorhees bemutat néhány Wordneten alapuló, közelműltbeli fordítási alkalmazást (Voorhees, 1993). Bevezető tankönyvet Hutchins és Somers írt (Hutchins és Somers, 1992).

A statisztikai gépi fordítás Weaver 1947-es megjegyzéséig nyúlik vissza, azonban csak az 1980-as évekre jutott el gyakorlati szintre. Az általunk bemutatottak Brown és kollégáinak IBM-beli munkáin alapulnak (Brown és társai, 1988; 1993). Nagyon komoly matematikát alkalmaznak, ezért a Kevin Knight által írt kísérő oktatóanyag sokat segít a megértésben (Knight, 1999). A még frissebb statisztikai gépi fordítás területén

végzett munkák továbblépnek a bigram modellen, és olyan modelleket használnak, amelyek valamennyi szintaxist is figyelembe vesznek (Yamada és Knight, 2001). A mondat-szegmentáció terén Palmer és Hearst végezték a korai munkákat (Palmer és Hearst, 1994). Michel és Plamondon a kétnyelvű mondatilleszsíről értekezik (Michel és Plamondon, 1996).

Két kitűnő könyv szól a valószínűségi nyelvfeldolgozásról: Charniak könyve rövid és lényegre törő (Charniak, 1993), míg Manning és Schütze munkája átfogó és naprakész (Manning és Schütze, 1999). A gyakorlati nyelvfeldolgozás terén végzett munkákat a kétévente megrendezett Applied Natural Language Processing (ANLP) és az Empirical Methods in Natural Language Processing (EMNLP) konferenciák, valamint a *Natural Language Engineering* folyóirat mutatja be. A SIGIR egy hírlevelet és egy információkeresést tárgyaló évenkénti konferenciát támogat.

## Feladatok

- 23.1.** (Jurafsky és Martin, 2000) alapján. Ebben a feladatban szerzőséget osztályozó rendszert fogunk kifejleszteni, amely egy adott szöveg esetén megpróbálja meg-határozni, hogy a két pályázó szerző közül melyik írta a szöveget. Szerezzünk be szövegeket két különböző szerzőtől. Válasszuk szét őket tanító és teszthal-mazokra. Tanítsunk unigram modellt minden szerző tanító halmaza alapján. Végül minden egyes tesztelő szöveget számítsuk ki minden szerző unigram modellre a valószínűséget, és rendeljük a legvalószínűbb modellhez. Határozza meg a technika pontosságát. Tudná növelni a pontosságot további jellemzőkkel? A nyelvészeti ezen részterületét a *stílusmérés tudományának* (*stylometry*) nevezik, sikerei közé sorolható a Federalist Papers (Mosteller és Wallace, 1964), valamint Shakespeare néhány vitatott munkája (Foster, 1989) szerzőjének meg-határozása.
- 23.2.** Ez a feladat az  $n$ -gram modellek minőségét méri fel. Keressen vagy hozzon létre körülbelül 100 ezer szavas egynyelvű korpuszt. Tördelje szavakra, és szá-mítsa ki az egyes szavak előfordulási gyakoriságát. Hány különböző szó található benne? Ábrázolja a gyakoriságot a sorrend (leggyakoribb, második leg-gyakoribb stb.) függvényében log–log skálán. Határozza meg a bigramok (két egymást követő szó) és trigramok (három egymást követő szó) gyakoriságát is. Használja ezeket a gyakoriságokat nyelv generálására: az unigram, bigram és trigram modell alapján a gyakoriságnak megfelelő véletlen választással ge-néraljon 100 szavas szövegeket. Hasonlítsa össze ezeket a szövegeket a tény-legesen beszélt nyelvvel. Végül számolja ki az egyes modellek összetettségét.
- 23.3.** Ez a feladat a spam-felismeréssel foglalkozik. A spam definíciója „kéretlen tö-meget reklám e-mail”. A spam kezelése bosszantó elfoglaltság számos e-mailező számára, tehát megbízható eltávolításuk áldás lenne. Hozzon létre spam és spam-men-ses e-mailekből álló korpuszt. Elemezze minden korpuszt, és állapítsa meg, hogy mely jellemzők lennének alkalmasok az osztályozásra (unigram, brigram, üzenethossz, feladó, érkezési idő). Ezután tanítson egy osztályozó algoritmust

(döntési fa, naiv Bayes vagy más, Ön által választott algoritmus) egy tanító halmazon, majd értékelje pontosságát egy teszthalmazon.

- 23.4.** Készítsen öt lekérdezésből álló teszthalmazt, majd küldje el három nagy webkeresőnek. Értékelje mindegyik pontosságát az első 1, 3 és 10 dokumentumra és az átlagos reciprok rangra. Próbálja megmagyarázni az eltéréseket.
- 23.5.** Próbáljon megbizonyosodni arról, hogy az előbbi feladat keresőgépei közül melyek használnak betűkonverziót, szótövesítést, szinonimákat és helyesírási-hiba-javítást.
- 23.6.** Becsülje meg, hogy mekkora tárterület szükséges egy egymilliárd weboldalt tartalmazó korpusz indexeléséhez. Mutassa be a feltételezéseket, amelyekre alapozott.
- 23.7.** Írjon egy reguláris kifejezést vagy egy rövid programot vállalatnevek kinyerésére. Tesztelje üzleti híreket tartalmazó cikkeken. Adja meg az elért felidézést és pontosságát.
- 23.8.** Válasszon ki öt mondatot és küldje el egy internetes fordítószolgáltatásnak. Fordítsa le angolról egy idegen nyelvre, majd vissza angolra. Értékelje a kapott mondatokat nyelvtani helyesség és a jelentés megtartásának szempontjából. Ismételje meg a folyamatot; az iteráció második lépése rosszabb vagy azonos eredményeket ad? Befolyásolja az eredmény minőségét a közbenső nyelv megválasztása?
- 23.9.** Gyűjtsön össze időre vonatkozó kifejezéseket, például „két órakor”, „éjfélkor”, „12:46-kor”. Találjon ki olyan nyelvtanilag helytelen példákat is, mint „huszonöt órakor” vagy „felnegyedháromkor”. Írjon nyelvtant az idő nyelvére.
- 23.10.** (Knight, 1999) alapján. Az IBM Model 3 gépi fordító feltételezi, hogy miután a szóválasztási modell egy szólistát ajánl, a nyelvi modell képes kiválasztani a legjobb permutációt. Ez a feladat megvizsgálja, hogy mennyire értelmes ez a feltételezés. Próbálja meghatározni a következő mondatok szavainak helyes sorrendjét:
- have programming a seen never I language better
  - loves john mary
  - is the communication exchange of intentional information brought by about the production perception of and signs from drawn a of system signs conventional shared
- Melyik mondat szavait tudta sorrendbe tenni? Milyen tudás alapján volt erre képes? Tanítson egy bigram modellt egy tanító korpusz alapján, és használja arra, hogy meghatározza egy tesztkorpusz mondatainak legnagyobb valószínűségű permutációit. Mutassa meg a modell pontosságát.

- 23.11.** Amennyiben megnéz egy angol-francia szótárt, a „hear” ige fordítása „entendre”.<sup>34</sup> Azonban ha az IBM Model 3-at a kanadai Hansard alapján tanítjuk, akkor a „hear” legvalószínűbb fordítása „Bravo”. Magyarázza meg, hogy mi ennek az oka, és becsülje meg, hogy mi lehet a „hear” termékenységi eloszlása. (Segítség: ha Hansard-szövegeket szeretne megnézni, írja be egy webkeresőbe a [Hansard hear] lekérdezést.)

<sup>34</sup> hall, hallgat (*A ford.*)

# 24. AZ ÉSZLELÉS

Ebben a fejezetben a számítógépet a zord, koszos világhoz csatlakoztatjuk.

Az észlelés információhoz juttatja az ágenst arról a viláról, amelyben létezik. Az észlelést az **érzékelők** (**sensors**) kezdeményezik. Egy érzékelő lehet bármí, ami a környezet valamelyik aspektusát rögzíti és továbbítja az ágensprogram számára. Ez olyan egyszerű is lehet, mint egy egybites érzékelő, amely egy kapcsoló be és ki állapotát jelzi, de lehet olyan bonyolult is, mint az emberi szem retinája, amely több mint százmillió fotoérzékeny elemet tartalmaz. Ebben a fejezetben a látásra összpontosítunk, mert a fizikai világ kezelésében ez az észlelésfajta messze a leghatásosabb.

## 24.1. BEVEZETÉS

Egy mesterséges ágens több észlelési modalitással is rendelkezhet. Amin az ágensek és az emberek is osztoznak, az többek között a látás, a hallás és a tapintás. A hallással a beszédfelismerés témán belül, a 15.6. alfejezetben foglalkozunk. A tapintás, másképpen a **taktilis érzékelés** (**tactile sensing**) a 25. fejezet témája, ahol a kézi műveleteket végző ágenseknél vizsgáljuk meg ennek a használatát, a fejezet hátralevő része pedig a látással foglalkozik. Egyes robotok **aktív érzékelést** (**active sensing**) végeznek, azaz kiküldenek egy jelet, mint például radar- vagy ultrahangjelet, és ennek a jelnek a környezetből érkező visszaverődését vizsgálják.

Egy ágens kétféle módon használhatja az érzékeléseit. A **tulajdonság kinyerési** (**feature extraction**) megközelítés szerint az ágens néhány kevés számú tulajdonságot észlel az érzékelő bemenetein, és ezeket továbbítja az ágensprogramnak, amely a tulajdonságoknak megfelelően reagál, vagy kombinálja őket más információkkal. A wumpus ágens íly módon működött, öt érzékelővel, amelyek mindegyike egybites információt nyert ki. Ma már ismert, hogy egy légy az optikai bemenetről tulajdonságokat nyer ki, és ezeket direkt módon a kanyarodásban segítő izmokhoz továbbítja, lehetővé téve, hogy 30 milliszekundumon belül reagáljon, és irányt váltszon.

Az alternatív lehetőség a **modellalapú** (**model-based**) megközelítés, ahol a szenzoros bemenet alapján a világ egy modellje épül fel. E megközelítés szerint egy  $f$  függvényből indulunk ki, amely a világ  $W$  állapotát az általa keltett  $S$  ingerületre képezi le:

$$S = f(W)$$

Az  $f$  függvényt a fizika és az optika határozza meg, és viszonylag jól ismerjük.  $S$  és egy valós vagy elköpelt  $W$  világ előállítása  $f$ -ból a **számítógépes grafika** (**computer**

**graphics**) által kezelt probléma. A számítógépes látás bizonyos értelemben a számítógépes grafika fordítottja, ha adott  $f$  és  $S$ , akkor  $W$ -t próbáljuk kiszámítani:

$$W = f^{-1}(S)$$

Sajnos az  $f$ -nek nincs jól definiált inverze. Nem láthatunk a sarkon túlra, így a világ aktuális állapotának minden aspektusát az ingerületből visszaállítani nem tudjuk. Sőt az a része, amit *láthatunk*, nagyban bizonytalan: kiegészítő információ nélkül nem tudjuk megmondani, hogy  $S$  egy játék Godzilla-képe, amint egy hatvan centiméteres modell-épületet tarol le, vagy egy valóságos szörnyeteg, amely egy hatvanméteres épületet pusztít el. Ezen problémák egy részét kezelhetjük úgy, hogy egy világ megtalálása helyett világok valószínűségi eloszlását határozzuk meg:

$$P(W) = P(W|S)P(S)$$

Az ilyen modellezés legfontosabb hátránya az, hogy túl nehéz problémát kísérel meg megoldani. Gondoljunk bele, hogy a számítógépes grafikában többórányi számításra lehet szükség egy film egyetlen kockájának az előállításához, amiből 24 kocka kell másodpercenként, és  $f^{-1}$  kiszámítása sokkal bonyolultabb, mint  $f$ -é. Világos, hogy ez túl sok számítás egy szuperszámítógép számára, hogy valós időben reagáljon, nem is beszélve egy légyről. Szerencsére az ágensnek nem kell a fényképhű számítógépes grafikához hasonló szintű modellel rendelkeznie a vilagról. Az ágensnek elegendő azt tudnia, hogy rejtozködik-e egy tigris a bozótban, és nem kell tudnia a tigris hátán levő minden egyes szörszál pontos elhelyezkedését és irányát.

A fejezet nagy részében azt fogjuk látni, hogy hogyan kell objektumokat – például tigriseket – felismerni, továbbá olyan módszereket, amelyek ezt a tigris minden egyes részletének reprezentálása nélkül teszik. A 24.2. alfejezetben a képalakítás folyamatát tanulmányozzuk majd, definíálva az  $f(W)$  függvény egyes aspektusait. Először a folyamat geometriájára tekintünk. Látni fogjuk, hogy a fény a világ objektumairól az ágens érzékelőjének képskiján lévő pontokra verődik vissza. Ez a geometria magyarázza meg, hogy egy nagy Godzilla a távolban miért néz ki úgy, mint egy kis Godzilla a közelben. Ezután a folyamat fotometriáját vizsgáljuk, amely leírja, hogy a táj fénye hogyan határozza meg a kép pontjainak fényességét. A geometria és a fotometria együtt egy olyan modellt nyújtanak, amely leírja, hogy a világ objektumai hogyan képződnek le a képpontok kétdimenziós tömbjére.

Annak megértése után, hogy hogyan jönnek létre a képek, megvizsgáljuk, hogy hogyan dolgozzuk fel őket. A vizuális feldolgozás folyamata az emberekben és a számítógépekben is három részre bontható. Először, avagy az alacsony szintű látás során (24.3. alfejezet), a feldolgozatlan képet simítjuk, hogy kiszűrjük a zajt, majd a kétdimenziós kép tulajdonságait kinyerjük, elsősorban a kép régiói közötti éleket. A középső szintű látásban ezeket összekapcsolva régiókat hozunk létre. A magas szintű látásban (24.4. alfejezet) a kétdimenziós régiókat mint a világ valóságos objektumait ismerjük fel (24.5. alfejezet). A képekben megtalálható olyan speciális információkat tanulmányozunk, amelyeket erre a célra felhasználhatunk, mint például a mozgást, a térbeli információt, a textúrát, az árnyalást és a kontúrokat. Az objektumfelismerés fontos a vadonban levő ágens számára, hogy észlelj a tigriseket, és fontos az ipari robotok számára, hogy megkülönböztesse az anyákat a csavaroktól. Végezetül a 24.6. alfejezet megmutatja, hogy az objektumok felismerése hogyan segíthet nekünk olyan hasznos feladatokban, mint

például a manipulálás vagy a navigáció. A manipulálás azt jelenti, hogy megragadhatunk és használhatunk szerszámokat, illetve más tárgyakat, a navigáció pedig azt, hogy egyik helyről átmehetünk egy másikra anélkül, hogy nekimennénk valaminek. Ezeket a feladatokat észben tartva biztosíthatjuk, hogy egy ágens csak akkora modellt építse, amire célja eléréséhez szüksége van.

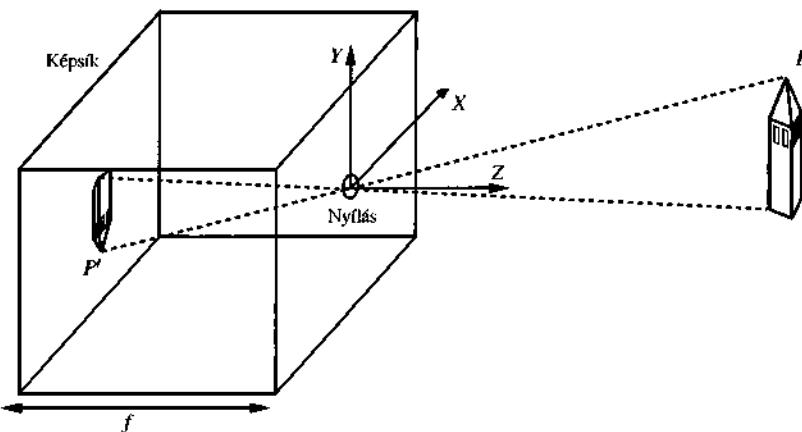
## 24.2. KÉPALKOTÁS

A látás egy **jelenet** (scene) objektumairól szóródó fényt összegyűji, majd egy két-dimenziós képet (image) alkot egy képsíkon. A képsíkot fényérzékeny anyag borítja: a retinában rodopszinmolekulák, a negatív filmen ezüst-halogenidek, a digitális fényképezőgépben pedig egy töltéscsatolt eszköz (CCD) lapka. A CCD minden cellája összegyűjti a fényelnyelés által adott idő alatt keltett elektronokat. A digitális fényképezőgépben a képsík egy néhány millió **képpontból** (pixel) álló négyzetláncra van felosztva. A szem hasonló képontráccsal rendelkezik, amely körülbelül 100 millió pálcikát és 5 millió csapot tartalmaz egy hexagonális mozaikba rendezve.

A jelenet nagyon nagy, és a képsík egészén kicsi, tehát lennie kell valamilyen módszernek a fény képsíkra fókuszálásához. Ezt megtehetjük lencsével vagy anélkül. Bárhogy is tesszük, a lényeg a geometria meghatározása, hogy megmondhassuk, a jelenet melyik pontja hova kerül a képsíkon.

### Lencsék nélküli képek: a sötétkamra

A képalkotás legegyszerűbb eszköze a sötétkamra, amely egy doboz elején található kis lyukból ( $O$ ) és a doboz hátsó részén levő képsíkból áll (lásd 24.1. ábra). Egy három-dimenziós koordináta-rendszert fogunk használni, amelynek az origója az  $O$  pont, a jelenet egy  $P$  pontját pedig az  $(X, Y, Z)$  koordinátákkal reprezentáljuk. A  $P$  pontot egy



24.1. ábra. A képalkotás geometriája a sötétkamrában

$P'$  pontra képezzük le a képsíkon, amelynek koordinátái  $(X, Y, Z)$ . Ha  $f$  az  $O$  pontszerű lyuknak a képsíktól vett távolsága, akkor a hasonló háromszögek alapján az alábbi egyenleteket tudjuk levezetni:

$$\frac{-x}{f} = \frac{X}{Z}, \frac{-y}{f} = \frac{Y}{Z} \Rightarrow x = \frac{-fX}{Z}, y = \frac{-fY}{Z}$$

Ezek az egyenletek egy **perspektivikus vetítés** (**perspective projection**) néven ismert képalkotási eljárást definiálnak. Vegyük észre, hogy a  $Z$  az osztóban azt jelenti, hogy minél távolabb van az objektum, annál kisebb lesz a képe. Valamint azt is figyeljük meg, hogy a kép, mind a bal–jobb, mind a fel–le irányban *fordított* a jelenethez képest, amit az egyenletekben a negatív előjel jelez.

A perspektivikus vetítésben a párhuzamos vonalak a horizontvonalon lévő pontban futnak össze. Nézzük meg, miért szükségszerű ez! Az  $(X_0, Y_0, Z_0)$  ponton keresztül  $(U, V, W)$  irányban áthaladó egyenes leírható, mint az  $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$  pontok halmaza, ahol  $\lambda$  a  $+\infty$  és  $-\infty$  között változik. Ezen egyenes egy  $P_\lambda$  pontjának képsíkra vett vetületét az:

$$\left( f \frac{X_0 + \lambda U}{Z_0 + \lambda W}, f \frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right)$$

adj. Amint  $\lambda \rightarrow \infty$  vagy  $\lambda \rightarrow -\infty$  ez a pont a  $p_\infty = (fU/W, fV/W)$  lesz, ha  $W \neq 0$ . A  $p_\infty$  pontot az  $(U, V, W)$  irányú egyenesek családjával kapcsolatos **távlatpontnak** (**vanishing point**) nevezzük. Az azonos irányú egyenesek távlatpontja azonos.

Ha a tárgy a kamrától vett távolságához képest viszonylag lapos, a perspektivikus vetést a **skálázott ortografikus vetítéssel** (**scaled orthographic projection**) közelíthetjük. Ennek elve a következő. Ha a tárgy pontjainak  $Z$  mélysége egy bizonyos  $Z_0 \pm \Delta Z$  tartományban változik, és  $\Delta Z = Z_0$ , akkor a perspektívá  $f/Z$  skála tényezőjét az  $s = f/Z_0$  állandóval lehet közelíteni. Az  $(X, Y, Z)$  jelenet koordinátákból a képsíkra történő vetítés egyenletei az  $x = sX$  és az  $y = sY$  lesznek. Jegyezzük meg, hogy a skálázott függőleges síkú vetítés egy közelítés, amely a jelenet azon részeire érvényes, amelyeknek a saját mélységváltozásuk elenyésző. A globális tulajdonságok tanulmányozására ezt a módszert nem szabad használni. Hogy az óvatosság nem árt, erre egy példa: függőleges síkú vetítésben a párhuzamos vonalak párhuzamosak maradnak ahelyett, hogy egy távlatpontban összefutnának.

## Lencserendszerek

A gerincesek szeme és a modern kamerák is lencséket használnak. A lencse a sötétkamra nyilásánál nagyobb, így több fényt enged át. Ennek az az ára, hogy egyidejűleg a jelenet minden részének az élességét nem lehet biztosítani. A jelenet egy  $Z$  távolságban lévő pontjának a képe a lencsétől egy rögzített  $Z'$  távolságban keletkezik, és a  $Z$  és a  $Z'$  között az:

$$\frac{1}{Z} + \frac{1}{Z'} = \frac{1}{f}$$

reláció áll fenn, ahol  $f$  a lencse fókusztávolsága. Ha a lencse optikai középpontja és a képsík közötti távolságot valamilyen  $Z'_0$ -nak választjuk, a jelenet azon tárgyai, ame-

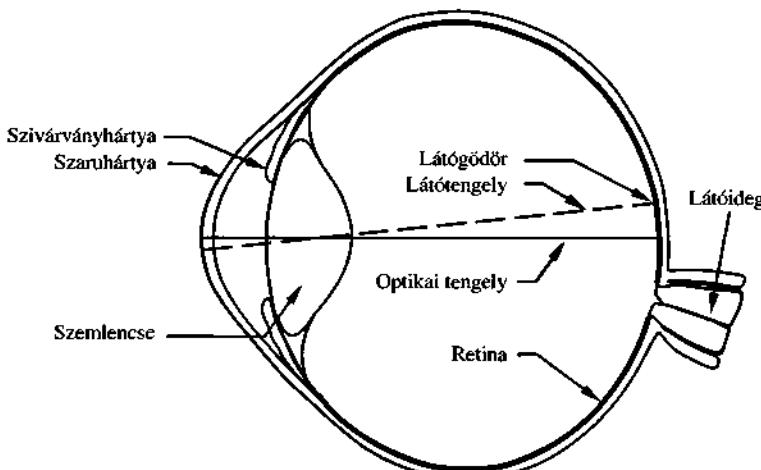
lyek a  $Z_0$  körül egy adott mélységtartományban fekszenek, ahol  $Z_0$  a megfelelő tárgytávolság, nagyjából élesen fognak látszani. A jelenet ezen mélységi tartományát **mélységelességnak (depth of field)** nevezzük.

Jegyezzük meg, hogy mivel a Z tárgytávolság általában sokkal nagyobb, mint a  $Z'$  képtávolság vagy mint az  $f$ , sokszor jogos az alábbi közelítés:

$$\frac{1}{Z} + \frac{1}{Z'} \approx \frac{1}{Z'} \Rightarrow \frac{1}{Z'} \approx \frac{1}{f}$$

Azaz a képtávolság  $Z' \approx f$ . A sötétkamra vetítésegényleteit tehát egy lencse által létrehozott kép geometriájának leírásához is használhatjuk.

Ahhoz, hogy a különböző Z távolságban lévő tárgyakat élesen lássuk, az emberi szem lencséje (lásd 24.2. ábra) változtatja az alakját, a kamera lencséje pedig Z irányban elmozdul.

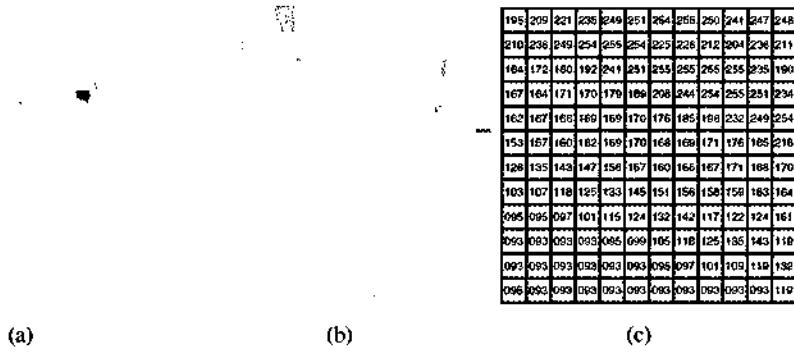


24.2. ábra. Az emberi szem vízszintes keresztmetszete

## A fény: a képalkotás fotometriája

A fény nélkülözhetetlen a látáshoz: nélküle minden kép egyformán sötét lenne, függetlenül attól, mennyire érdekes a jelenet. A **fotometria (photometry)** a fény tanulmányozása. Saját céljainkra azt fogjuk modellezni, hogy a jelenet fénye hogyan képződik le az időben a képsík fényintenzitására, amit  $I(x, y)$ -nal<sup>1</sup> jelölünk. A látás rendszere ezt a modellt visszafelé alkalmazza, a képek intenzitásából kiindulva a világ tulajdonságai felé. A 24.3. ábra egy asztalon lévő tűzögép digitalizált képét és a tűzögép egy  $12 \times 12$  képpontból álló részletét mutatja. Egy számítógépes program, amely értelmezni próbálja a képet, egy ilyen intenzitásmátrixból indulna ki.

<sup>1</sup> Ha az időbeli változást is vizsgáljuk, akkor az  $I(x, y, t)$  jelölést alkalmazzuk.



24.3. ábra. (a) Egy asztalon lévő tűzörgép fényképe. (b) Az (a) egy  $12 \times 12$  képpontból álló, nagyított részlete. (c) A részletnek megfelelő képfényességértékek egy 0-tól 255-ig tartó skálán.

A kép egy képpontjának a fényessége arányos a jelenet képpontba vetített felületeleme által a kamera felé irányított fény mennyiségével. Ez viszont függ a felületelem fényvisszaverő tulajdonságaitól és a fényforrások elhelyezésétől, valamint eloszlásától a jelenetben. Egy képpont fényességébe a jelenet egyéb részeinek fényvisszaverő tulajdonságai is beleszólnak, hiszen a jelenet más felületei indirekt fényforrásként szolgálnak, mivel a rájuk eső fényt az adott felületelem felé verik vissza.

Kétféle visszaverődést modellezhetünk. A **tüköröző visszaverődés** (specular reflection) azt jelenti, hogy a fény a tárgy külső felületéről verődik vissza, és teljesíti azt a kényszert, miszerint a beesés és a visszaverődés szöge megegyezik. Ilyen a tökéletes tükröződés. A **diffúz visszaverődés** (diffuse reflection) azt jelenti, hogy a fény a tárgy felszínén belülre hatol, a tárgy a fényt elnyeli és ismételten kisugározza. A tökéletesen diffúz (avagy **Lambert-féle**) felület minden irányban azonos intenzitással szórja a fényt. Az intenzitás egyedül a fényforrástól érkező fény beesési szögétől függ: a felületre pontosan merőleges fényforrás esetén lesz a legnagyobb intenzitású a refleksió, míg majdnem párhuzamos fényforrás esetén a legkisebb. E két véglet között a visszaverődést Lambert koszinusztörvénye írja le:

$$I = kI_0 \cos\theta$$

ahol  $s$  a fényforrás intenzitása, a felület normálisa és a fény beesési iránya közötti szög,  $k$  a visszaverődési tényező értéke, amely a felület visszaverési képességeitől függ, és 0-tól (tökéletesen fekete felület) 1-ig (tökéletesen fehér felület) változik.

A valódi életben a felületek kevert, diffúz és tükrös tulajdonságúak. Számítógépes modellezésük a számítógépes grafika fő alkalmazási területe. A valósághű képek előállítása általában fénysugár-követési eljárással történik, amely szimulálja azt a fizikai folyamatot, ahogyan a fénysugár a forrásnál keletkezik, és a tárgyakról többszörösen visszaverődik.

## Színek: a képalkotás spektrális fotometriája

A 24.3. ábrán egy fekete-fehér képet mutattunk, nagyban figyelmen kívül hagyva azt a tényt, hogy a látható fény a hullámhosszak egész tartományát öleli át – a 400 nm-től kezdve a spektrum ibolyaszínű végénél, egészen a 700 nm-ig a vörös színű végénél. Bizonyos fény csak egyetlen hullámhosszból áll, amely a szírvány egy színének felel meg. De más fények különböző hullámhosszok keverékei. Azt jelenti ez, hogy az  $I(x, y)$  mértékére egyetlen szám helyett az értékek egy keveréke kell? Ha a fény fizikáját pontosan akarnánk reprezentálni, akkor bizony igen. De ha csak utánozni akarjuk az emberek (és más gerincesek) fényérzékelését, akkor köthetünk kompromisszumokat. Kísérletek mutatják (egészen 1801-től, Thomas Youngtól), hogy hullámhosszak bármilyen komplex keverékét is előállíthatjuk minden össze három szín keverésével. Azaz, ha van egy fénygenerátor, amely képes lineárisan kombinálni három hullámhosszt (tipikusan a vörös [700 nm], a zöld [546 nm] és a kék [436 nm] színeket választjuk), akkor az egyes színeket erősítő, másokat gyengítő gombok csavargatásával *bármilyen* hullámhossz-kombináció beállítható, legalábbis az emberi érzékelés szempontjából. Ez a kísérleti tény azt jelenti, hogy a képek egy olyan vektorral reprezentálhatók, amely képpontokként csak három intenzitásértéket tárol: minden elsőleges hullámhosszra egyet. A gyakorlatban mindegyiket egy bájtal ábrázolva a kép igen jó minőségű reprodukcióját kapjuk. A színek ezen háromszínű észlelése azzal a tényel van összefüggésben, hogy háromféle csap található a retinában, amelyek érzékelési maximuma 650, 530 és 430 nm, de az összefüggés pontos részletei az egy az egyben történő leképezésnél bonyolultabbak.

## 24.3. ELŐZETES KÉPFELDOLGOZÁSI MŰVELETEK

Láttuk eddig, hogy a fény hogyan tükröződik a jelenet tárgyairól egy mondjuk ötmillió hárombájtos képpontból álló képet formálva. Mint minden érzékelő esetében, most is lesz zaj a képben, és minden esetben nagyon sok adattal kell dolgozni. Elsőként a finomítás műveleteit nézzük meg, amelyekkel a zaj csökkenthető, majd az eldetektálás műveleteit vizsgáljuk. Ezeket „előzetes” vagy „alacsony szintű” műveleteknek hívjuk, mivel ezek az elsők egy műveletsorozatban. Az előzetes látási műveletekre egyrészt a lokalitás jellemző (a kép egy részletére végrehajthatók anélkül, hogy néhány képpontnál messzebb bármit is figyelembe vennénk), másrészt az ismeretek hiánya: anélkül finomíthatjuk a képeket vagy detektálhatunk éleket, hogy bármilyen fogalmunk lenne arról, milyen objektumok vannak a képeken. Ez az alacsony szintű műveleteket esetleggyessé teszi a párhuzamos hardveren történő megvalósításra, akár élőlényekben, akár mesterséges eszközökben. Ezek után a középső szinten levő műveletek nézzük majd meg, amelyekkel a kép régióra bontható. A műveletek ezen fázisa még mindig a képen dolgozik, nem a jeleneten, de nem helyi feldolgozást igényel.

A 15.2. alfejezetben a simítás (smoothing) azt jelentette, hogy egy állapotváltozó értékét játszottuk meg valamelyen múltbeli  $t$  időpontra, bizonyos, a  $t$  időpontra és további, egészen a jelenig terjedő időpontokra vonatkozó tények ismeretében. Most ugyanezt az ötletet alkalmazzuk a térbeli kiterjedésre az időbeli helyett: a simítás egy képpont értékének játslását jelenti a környező képpontok alapján. Vegyük észre, hogy világosan megkülönböztetjük a képpontban mért megfigyelt értéket és azt a valódi értéket, amit

ott mérnünk kellett volna. Ezek véletlenszerű és szisztematikus mérési hibák (például ha a CCD érzékelője elromlott) miatt különbözhetnek.

A simítás egyik módszere az, ha minden képponthoz az öt környező szomszédok értékeinek átlagát rendeljük. Ez az extrém értékek kihagyása felé mutató megoldás. De hány szomszédot kell figyelembe vennünk – az egy képpontnyi távolságra vagy a kettő esetleg több pont távolságra lévőket? Erre egy válasz, amely jól működik Gauss-zajok kiszűrésére, a **Gauss-szűrő** (**Gaussian filter**) használó súlyozott átlag. Emlékezzünk a standard  $\delta$  szórással rendelkező Gauss-függvényre:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-x^2/2\sigma^2} \quad \text{egy dimenzióban, illetve}$$

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad \text{két dimenzióban.}$$

Egy Gauss-szűrő alkalmazása azt jelenti, hogy az  $I(x_0, y_0)$  intenzitást lecseréljük a minden  $(x, y)$ -ra kiszámolt  $I(x, y)G_\sigma(d)$  intenzitások összegével, ahol  $d$  az  $(x_0, y_0)$  és az  $(x, y)$  pontok távolsága. Az ilyen fajta súlyozott összeg olyan gyakori, hogy van számról egy speciális elnevezés és jelölés. Azt mondjuk, hogy a  $h$  függvény az  $f$  és a  $g$  függvények **konvolúciója (convolution)** (amit  $h = f * g$  alakban írunk), ha

$$h(x) = \sum_{u=-\infty}^{+\infty} f(u)g(x-u) \quad \text{egy dimenzióban, illetve}$$

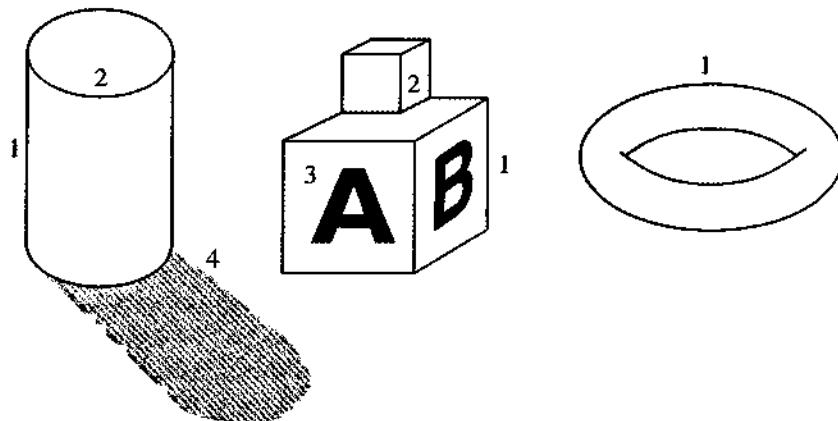
$$h(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)g(x-u, y-v) \quad \text{két dimenzióban.}$$

Azaz a simított függvényt úgy kapjuk, hogy a képet a Gauss-függénnyel konvolváljuk:  $I * G_\sigma$ . Egy képpontnyi  $\sigma$  kevés zaj kisimitásához elegendő, míg 2 képpontnyi több zajt fog elsimítani, de valamilyen mértékű részletvesztés mellett. Mivel a Gauss-hatás a távolsággal eltűnik, a gyakorlatban a  $\pm \infty$  a szummnákban  $\pm 3\sigma$ -val helyettesíthető.

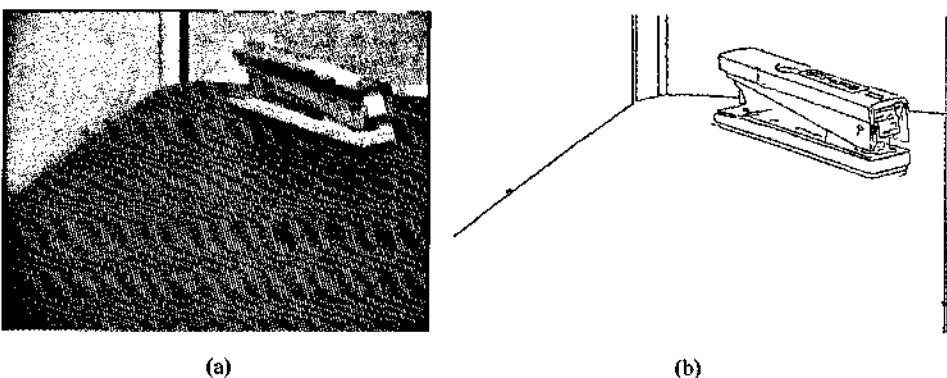
## Éldetektálás

Az előzetes látás következő lépése az **élek** (**edges**) detektálása a képsíkon. Az élek olyan egyenes vagy görbe vonalak a képsíkon, amelyekre merőlegesen a kép fényességében „lényegesen” változás van. Az éldetektálás végső célja, hogy eltávolodjunk a bonyolult, több megabájtos képtől egy tömörebb, absztrakt reprezentáció felé, mint amilyen a 24.4. ábrán látható. Az indíték az, hogy a kép élkontúrai a jelenet fontos kontúrainak felelnek meg. A képen három mélységi diszkontinuitás példa szerepel 1-gyel címkézve; a 2-vel címkézett felületi orientáció diszkontinuitásból két példa látható, míg a 3-mal címkézett visszaverődési képesség diszkontinuitásokról és a 4-gyel címkézett fényességi diszkontinuitásokról (árnyék) egy-egy. Az éldetektálás csak a képpel foglalkozik, így nem tesz különbséget a jelenet ezen diszkontinuitásfajtáit között, de a későbbi feldolgozás igen.

A 24.5. (a) ábra egy asztalon nyugvó tűzögépet tartalmazó jelenet mutatja, a 24.5. (b) pedig a képre lefuttatott éldetektáló algoritmus eredményét mutatja. Láthatjuk, hogy az éldetektáló kimenete és az ideális vonalas ábra között van különbség. A kis részek élei nem mind illeszkednek egymáshoz, vannak hézagok, ahol nincsenek vonalak, valamint



24.4. ábra. Különböző élek: (1) mélységi diszkontinuitás, (2) felületi orientáció diszkontinuitás, (3) viszszaverődési képesség diszkontinuitás, (4) fényességi diszkontinuitás (árnyékok)



24.5. ábra. (a) Egy asztalon nyugvó tűzögép fényképe. (b) Az (a) alapján meghatározott élek.

„zajos” kontúrok, amelyek a jelenetben semmilyen lényeges vonásnak nem felelnek meg. A későbbi feldolgozási fázisokban ezeket a hibákat majd korrigálni kell. Hogyan detektáljuk a képen az éleket? Figyeljük meg a kép fényintenzitásprofilját egy egymeneti részelmetszet mentén egy élere merőleges irányban, pl. az asztal bal élé és a fal között. Nagyjából olyan lesz ez, mint amit a 24.6. (a) ábra mutat. Az él helye az  $x = 50$ -nek felel meg.

Tekintettel, hogy az élek azoknak a képhelyeknek felelnek meg, ahol a fényességen hirtelen változás tapasztalható, egy naiv ötlet az lehetne, hogy a képet differenciáljuk, és azokat a helyeket keressük meg, ahol az  $I'(x)$  derivált értéke nagy. Nos, az ötlet majdnem működőképes. A 24.6. (b) ábrán láthatjuk, hogy az  $x = 50$  csúccsal együtt más helyeken, másodlagos csúcsok is megjelentek (például  $x = 75$ ), amiket tévesen valódi éleknek fogadhatunk el. A jelenség oka a kép zajossága. Ha a képet először simítjuk, akkor a felesleges csúcsok eltűnnek, ahogy azt a (c) mutatja.

Van arra esélyünk, hogy optimalizáljuk a munkát ezen a ponton: egyetlen műveletben kombinálhatjuk a simítást és az éldetektálást. Egy téTEL szerint tetszőleges  $f$  és  $g$

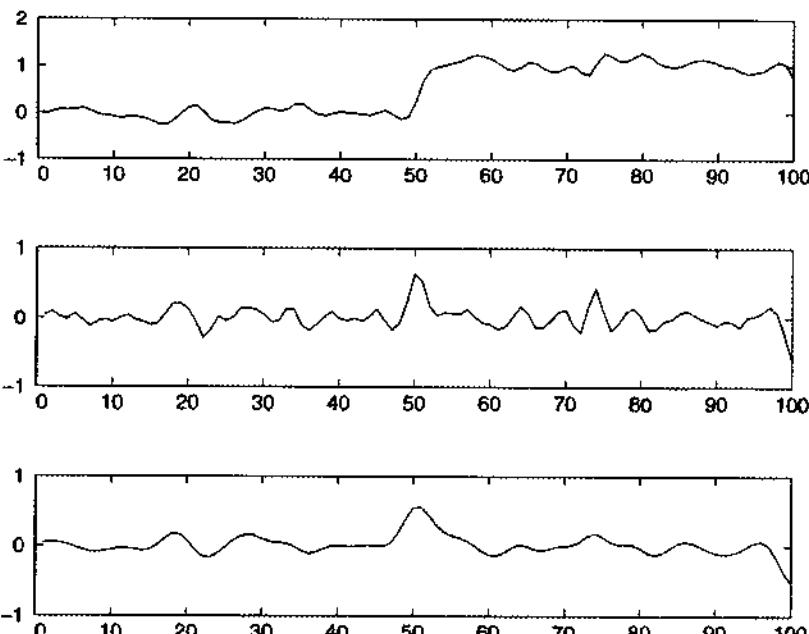
függvények esetén meghatározható, hogy a konvolúció deriváltja  $(f * g)'$  megegyezik a derivált konvolúciójával:  $f * (g)'$ . Tehát ahelyett, hogy először simítanánk, majd differenciálnánk, egyszerűen konvolváljuk a képet a Gauss simító függvény deriváltjával,  $G'_\sigma$ -val. Így az egydimenziós éldetektálás algoritmusára a következő:

1. Az  $I$  képet a  $G'_\sigma$ -val kell konvolválnunk, hogy az  $R$ -et megkapjuk.
2. Jelöljük be az azokat a csúcsokat  $\|R(x)\|$ -ben, amelyek egy előre definiált  $T$  küszöbnél magasabbak. A küszöb értékét úgy határozzuk meg, hogy a zaj miatti másodlagos csúcsokat elimináljuk.

Két dimenzióban egy él irányítottsága tetszőleges  $\theta$  szögű lehet. A függőleges élek detektálásának módja nyilvánvaló:  $G'_\sigma(x) G_\sigma(y)$ -nal kell konvolválnunk a képet. A hatás y irányban simító (a Gauss-konvolúció miatt), x irányban pedig egy simítással kísért differenciálás. A függőleges élek detektálási algoritmusára így az alábbi:

1. Konvolváljuk az  $I(x, y)$  képet az  $f_V(x, y) = G'_\sigma(x) G_\sigma(y)$ -nal, hogy az  $R_V(x, y)$ -t megkapjuk.
2. Jelöljük be azokat a csúcsokat az  $\|R_V(x, y)\|$ -ben, amelyek egy előre definiált  $T_n$  küszöbnél magasabbak.

A tetszőleges orientációjú él detektálásához a képet két szűrővel: az  $f_V = G'_\sigma(x) G_\sigma(y)$  és az  $f_H = G'_\sigma(y) G_\sigma(x)$  szűrővel kell konvolválni (az  $f_H$  az  $f_V$  90°-kal elforgatott változata). A tetszőleges orientációjú éldetektáló algoritmus az alábbi:



**24.6. ábra.** Felül: Az  $I(x)$  intenzitásprofil egy egydimenziós részlet mentén egy lépcsőfok előn át. Középen: Az intenzitás deriváltja,  $I'(x)$ . A függvény nagy értékei éleknek felelnek meg, de a függvény zajos. Lent: Az intenzitás simított változatának deriváltja,  $(I * G'_\sigma)'$ , amely  $I * G'_\sigma$ -hoz hasonlóan egy lépéshoz számolható. Az  $x = 75$  pontban levő zajos él jelölt eltűnt.

1. Konvolváljuk az  $I(x, y)$  képet az  $f_V(x, y)$ -nal, valamint az  $f_H(x, y)$ -nal, hogy az  $R_V(x, y)$ -t és az  $R_H(x, y)$ -t megkaphassuk. Definiáljuk az  $R(x, y) = R_V^2(x, y) + R_H^2(x, y)$ -t.
2. Jelöljük be azokat a csúcsokat az  $||R(x, y)||$ -ban, amelyek egy előre definiált  $T$  küköznél magasabbak.

Miután ezzel az algoritmussal megjelöltük az élek képpontjait, a következő lépés az azonos élekhez tartozó képpontok összekötése. Ez megtehető, ha feltételezzük, hogy bármely két szomszédos képpont, amely élpont és konzisztens irányítottságú, szükségszerűen ugyanahhoz az élhez tartozik. Ezt a folyamatot **Canny-féle éldetektornak** (**Canny edge detection**) nevezzük, kitalálója, John Canny után.

Miután detektáltuk őket, az élek alkotják több következő feldolgozási lépés alapját: felhasználhatjuk őket a térbeli képalkotásban, a mozgásdetektálásban és az objektumok felismerésében.

## A kép szegmentálása

Az emberek *rendszerzik* észleléseiket; az egyes fotóérzékelőkhöz tartozó fényességértékek halmaza helyett vizuális csoportokat észlelünk, amelyek általában objektumokkal vagy azok részeivel asszociáltak. Ez a képesség a számítógépes látás számára is éppúgy fontos.

A **szegmentálás (segmentation)** a kép felbontása részcsoportokra a képpontok hasonlósága alapján. Az alapötlet a következő: minden képponthoz rendelhetünk bizonyos vizuális tulajdonságokat, mint például fényesség, szín és mintázat.<sup>2</sup> Ezek az attribútumok egy objektumon vagy annak egyetlen részén belül, csak kismértékben változnak, míg az objektumok közötti határok mentén tipikusan egyik vagy másik attribútum jelenlegesen változik. Úgy kell a képet képpontok halmazaira felosztanunk, hogy ezeket a kényezeteket amennyire csak lehet, kielégítsük.

Többféle módszer létezik ezen intuitív elképzélés formalizálására. Például Shi és Malik (Shi és Malik, 2000) ezt gráfparticionálási problémaként határozzák meg. A gráf csomópontjai a képpontoknak felelnek meg, az élek pedig a köztük levő kapcsolatoknak. Egy  $W_{ij}$  súly értéke az  $i$  és  $j$  képpontokat összekötő élen attól függ, hogy a két képpont mennyire hasonló fényességen, színben, mintázatban és más tulajdonságokban. Ezek után olyan particiókat keresnek, amelyek minimalizálnak egy normalizált vágási kritériumot. Durván fogalmazva a gráf particionálásának kritériuma, hogy a csoportok közötti élek súlyainak összegét minimalizáljuk, a csoportokon belüli élek súlyainak összegét pedig maximalizáljuk.

A pusztán alacsony szintű, helyi tulajdonságokon – például a fényességen és színén – alapuló szegmentáció hibázásra hajlamos eljárás. Ahhoz, hogy az objektumokhoz rendelt határokat megbízhatóan megtalálhassuk, a jelenetben várható objektumfajtákról magas szintű tudást is be kell építenünk. A beszédfelismerésre ezt a rejtegett Markov-modell formalizmusa teszi lehetővé, míg a képek kontextusában egy ilyen egységes keretrendszer még mindig az aktív kutatás tárgya. Mindenesetre az objektumokról szóló magas szintű tudás a következő alfejezet tárgya.

<sup>2</sup> A mintázati jellemzők a képpontra középpontozott kis felületdarabka statisztikai elemzésén alapulnak.

## 24.4. 3D INFORMÁCIÓ KINYERÉSE

Ebben az alfejezetben azt mutatjuk meg, hogyan kell a kétdimenziós képből kiindulva a jelenet háromdimenziós reprezentációjáig eljutni. Fontos, hogy a jelenettel foglalkozunk, hiszen végső soron az ágens a világban létezik, nem a képsíkon, a látás célja pedig annak elérése, hogy az ágens a világ objektumaival együttműködjön. Mindazonáltal az ágensek többségének csak a jelenet egyes aspektusainak korlátozott absztrakt reprezentációjára van szüksége, nem minden részletre. A könyv hátralevő részében látható, a világgal foglalkozó algoritmusok az objektumok tömör leírására építének, nem minden háromdimenziós felületdarabka kimerítően teljes számbavételére.

Elsőként az **objektumfelismerést** (*object recognition*) tárgyaljuk, azt a folyamatot, amely során a képek jellemzői (mint például az élek) átalakulnak ismert objektumok (mint például a tűzögépek) modelljeivé. Az objektumfelismerés három lépésből áll: a jelenet szegmentálása különálló objektumokra, mindenkoruk objektum pozíciójának és térfelület orientációjának a megfigyelőhöz képesti meghatározása, valamint mindenkoruk objektum alakjának a meghatározása.

A manipuláció és a navigálás feladatait szempontjából legfontosabb az objektum pozíciójának és térfelület orientációjának (az objektum ún. **helyzetének – pose**) a meghatározása a megfigyelőhöz képest. Ahhoz, hogy egy nyüzsgő gyárban eligazodjunk, tudunk kell az akadályok elhelyezkedését, mert akkor egy olyan pályát tudunk tervezni, amely az akadályokat kikerüli. Ha egy tárgyat szeretnénk felvenni és kézben tartani, a kézhez képest helyzetet kell ismernünk, mert csak így tudjuk generálni a megfelelő pályájú cselekvésszekvenciát. A manipulálást és a navigálást tipikusan szabályozási hurokban oldjuk meg – az érzékelők információja visszacsatolást jelent, amivel a robot vagy a robotkar mozgását befolyásolni tudjuk.

Használunk most matematikai jelölést a pozíció és az orientáció leírására. A jelenet egy  $P$  pontjának a pozíóját három szám jellemzi, a  $P$  pont ( $X, Y, Z$ ) koordinátái a sötét-kamera nyílásában elhelyezett origójú és az optikai tengellyel parallel  $Z$  tengelyű koordináta-rendszerben (lásd 24.1. ábra). Amivel rendelkezünk, az a kép egy pontjának ( $x, y$ ) perspektivikus vetülete. Ez egyben a fénysugarat is definiálja a nyílás felől, amely minden valahol elhelyezkedik a  $P$  pont. A  $P$  pont távolságát azonban nem ismerjük. Az „orientáció” fogalmát kétféle értelemben lehet használni:

- Az objektum egészének az orientációja.** Ezt egy háromdimenziós forgatási függvényben specifikálhatjuk, az objektum koordináta-rendszerét a kamerához viszonyítva.
- Az objektum felületének az orientációja a  $P$  pontban.** Ez egy  $n$  normál vektorral adható meg, amely a felületre merőleges irányt határozza meg. A felület orientációját sokszor a lejtéssel (*slant*) és a dölléssel (*tilt*) fejezzük ki. A döllés az  $n$  és a  $Z$  tengely által bezárt szög. A lejtés az  $X$  tengely és az  $n$  képsíkra vett vetülete által bezárt szög.

Amikor a kamera a tárgyhoz képest elmozdul, az objektumnak mind a távolsága, mind az orientációja megváltozik. Ami viszont nem fog változni, az a tárgy **alakja (shape)**. Ha a tárgy pl. egy kocka, akkor a kocka jellege nem változik, ha a tárgy mozog. A geometria kutatói századok óta próbálkoznak az alak fogalmának formalizálásával – abból az alapötletből kiindulva, hogy az alak az a valami, ami bizonyos transzformációcsoporthoz – pl. a forgatások és a transzlációk kombinációi – hatására nem változik. A nehézség abban rejlik, hogy hogyan keressük meg a globális alak egy olyan reprezentációját, amely

elegendően általános ahhoz, hogy a valódi világ alakbeli változékonysságát – és nem csupán oly egyszerű formákat, mint a hengerek, a kúpok és a gömbök – lefedje, és mégis könnyen kinyerhető legyen a vizuális bemenetből. A felület *lokális* alakjának jellemzése sokkal jobban feltárt. Ez lényegében a görbülettel lehetséges – azaz azzal, hogy hogyan változik a felület normálisa, ha a felületen különböző irányokba elmozdulunk. A sík esetén egyáltalán nincs változás. A henger esetén sincs, ha a tengely mentén haladunk, a rá merőleges irányban azonban a normális, a henger sugarával fordítottan arányos ütemben forog. Ezekkel a kérdésekkel a differenciálgeometria foglalkozik.

A tárgy alakja bizonyos manipulációs feladatok szempontjából fontos, pl. ahhoz, hogy eldöntsük, hogy a tárgyat hol fogjuk meg. Legfontosabb szerepe azonban a tárgyak felismerésében van, ahol a geometriai alak a színnel és a textúrával egyetemben a legjellemzőbb vonást jelenti, hogy a tárgyat azonosíthassuk, a kép tartalmát a korábban látott osztályokba besoroljuk és így tovább.

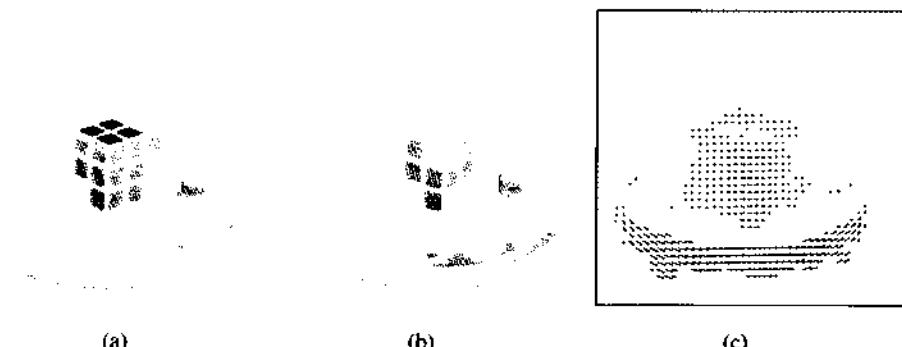
Az alapvető probléma a következő: ha egyszer a perspektivikus vetítés alatt a háromdimenziós világ egy pontból kiinduló sugár mentén található minden pontja a képnek ugyanabba a pontjába vetítődött, hogyan nyerjük vissza a háromdimenziós információt? A vizuális érzékelésben ehhez számos segítő dolog áll rendelkezésre, a mozgást, a sztereoláztatást, a textúrát, az árnyalást és a kontúrokat beleértve. Ezek mindegyike, ahhoz hogy a kép értelmezését (közel) egyértelmű módon megoldhassa, a jelenetre vonatkozó háttér-feltételezések alapul. Az alábbiakban ezekkel a módszerekkel foglalkozunk.

## A mozgás

Eddig egy időpontban csak egy képpel foglalkoztunk. A videokamerák azonban másodpercenként 30 képkockát rögzítenek, és a kockák közötti különbségek fontos információk forrásai lehetnek. Abban az esetben, ha a kamera a háromdimenziós jelenethez képest mozog, a képen keletkező látszólagos mozgást **optikai folyamnak (optical flow)** nevezzük. A folyam a képbeli jellegzetességek mozgásirányát és mozgási sebességét írja le, amit a jelenet és a megfigyelő egymáshoz viszonyított relatív mozgásának következtében kapunk. A 24.7. (a) és (b) ábrán egy forgó Rubik-kockát ábrázoló videofelvételből látunk két filmkockát. A (c) képen az előbbi képekből számított optikai folyam vektorai láthatók. Az optikai folyam a jelenet struktúrájára nézve tartalmaz hasznos információt. Például egy forgó kocsiból kinézve a távoli tárgyaknak sokkal lassúbb a látszólagos mozgása, mint a közelieknek, így a látszólagos mozgás üteme a távolságról is hordoz valamiféle információt.

Az optikai folyam vektormezije az  $x$  irányú  $v_x(x, y)$  és az  $y$  irányú  $v_y(x, y)$  komponensekkel jellemezhető. Ahhoz, hogy az optikai folyamot megmérhessük, meg kell találni az egymásnak megfelelő pontokat a szomszédos képkockákon. Ehhez azt a tényt használjuk fel, hogy az egymásnak megfelelő pontok körül képrészüknek hasonló a fényességmintázata. Tekintsük a  $p$ ,  $(x_0, y_0)$  képpont körül  $t_0$  időpillanatban csoportosuló képponttömböt. Ezt a tömböt a  $t_0 + D_t$  időpillanatban a különböző  $(x_0 + D_x, y_0 + D_y)$  pozíciójú  $q_i$  képpontjelöltek körül csoportosuló képponttömbökkel össze kell hasonlítni. A hasonlóság egyik szóba jöhető mértéke a **különbségek négyzetösszege (sum of squared differences, SSD)**:

$$SSD(D_x, D_y) = \sum_{(x,y)} (I(x, y, t) - I(x + D_x, y + D_y, t + D_t))^2$$



**24.7.** (a) Rubik-kocka egy forgó lemezkorongon. (b) Ugyanaz a kocka 19/30 másodperccel később (Richard Szeliski nyomán). (c) Az (a) és (b) képek összehasonlítása alapján számolt optikai folyam vektorok (Joe Weber és Jitendra Malik hozzájárulásával).

Az  $(x, y)$  az  $(x_0, y_0)$  képpont körül tömb képpontjainak halmozán fut végig. Megkeresük most azt a  $(D_x, D_y)$ -t, amely az SSD-t minimizálja. Az  $(x_0, y_0)$ -beli optikai folyam ilyenkor:  $(v_x, v_y) = (D_x/D_t, D_y/D_t)$ . Alternatívaként maximalizálhatjuk a keresztkorrelációt (cross-correlation):

$$\text{Korreláció}(D_x, D_y) = \sum_{(x,y)} I(x, y, t)I(x + D_x, y + D_y, t + D_t)$$

A keresztkorreláció akkor használható a legjobban, ha a jelenet képében textúra található, mert ekkor a vizsgált ablakokban nagy a fényességváltozás a képpontok mentén. Ha egy egyenletesen fehér falra nézünk, a különböző  $q$  jelöltekre számított keresztkorreláció majdnem azonos lesz, és az algoritmus a gyakorlatban véletlenszerűen fog választani.

Tegyük fel, hogy a megfigyelő  $T$  transzlációs és szögsebességgel rendelkezik [amelyek így a sajátmozgást (egomotion) írják le]. Származtatni lehet olyan összefüggéseket, amelyek a megfigyelő sebességét, az optikai folyamot és a jelenet tárgyainak a pozícióját kapcsolják össze. Tegyük fel, hogy  $f = 1$ , akkor:

$$v_x(x, y) = \left[ -\frac{T_x}{Z(x, y)} - \omega_y + \omega_z y \right] - x \left[ -\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x \right]$$

$$v_y(x, y) = \left[ -\frac{T_y}{Z(x, y)} - \omega_z x + \omega_x \right] - y \left[ -\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x \right]$$

ahol  $Z(x, y)$  az  $(x, y)$  képponthoz tartozó jelenetbeli pontnak a  $z$  koordinátája.

A képlet jobb megértését szolgálja, ha az egyszerű transzláció esetét nézzük. Ebben az esetben a folyammező:

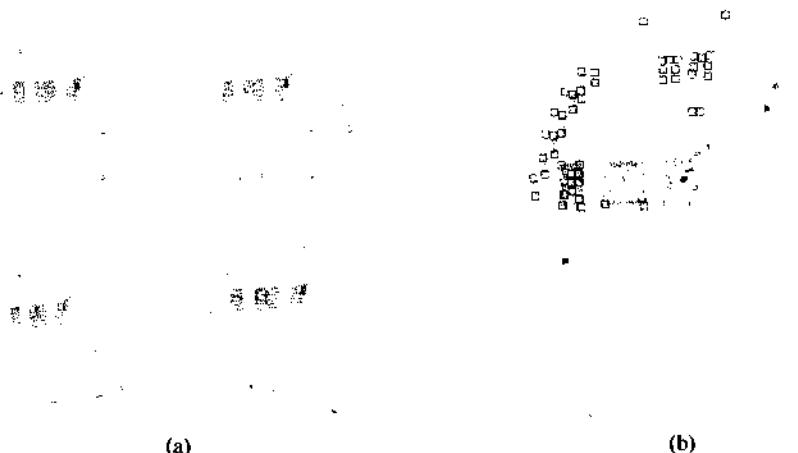
$$v_x(x, y) = \frac{-T_x + xT_z}{Z(x, y)} \quad v_y(x, y) = \frac{-T_y + yT_z}{Z(x, y)}$$

lesz. Néhány érdekes tulajdonságot figyelhetünk meg. Az optikai folyam minden komponense,  $v_x(x, y)$  és  $v_y(x, y)$  is zérus az  $x = T_x/T_z$ ,  $y = T_y/T_z$  pontban. Ezt a pontot a

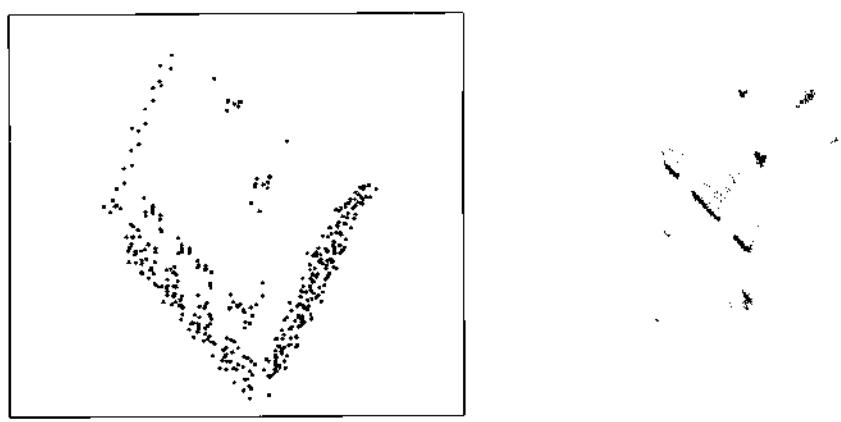
folyammező **expanziófókusznak** (**focus of expansion**) nevezik. Tegyük fel, hogy az  $x - y$  sík origóját az expanziófókuszból helyezzük át. Ilyenkor az optikai folyam kifejezései igen egyszerű formát öltenek. Jelölje  $(x', y')$  az  $x' = x - T_x / T_z$ , és az  $y' = y - T_y / T_z$  által definiált új koordinátákat. Ekkor:

$$v_x(x', y') = \frac{x' T_z}{Z(x', y')} \quad v_y(x', y') = \frac{y' T_z}{Z(x', y')}$$

Ennek az egyenletnek van néhány érdekes alkalmazása. Tegyük fel, hogy ön egy légy, amely egy falra próbál leszállni, és azt szeretné tudni, hogy adott aktuális sebesség mellett mennyi idő múlva ér a falhoz. Ezt az időt a  $Z / T_z$  adja meg. Figyeljük meg, hogy annak ellenére, hogy a pillanatnyi optikai folyam nem képes sem a  $Z$  távolságot, sem



**24.8. ábra.** (a) Egy olyan videokép-sorozat négy kockája, amelyben a kamera a tárgyhoz képest mozgott és forgott. (b) A sorozat első képkockája, amelyen apró négyzetek jelzik a tulajdonságfelismerő által megtalált tulajdonságokat (Carlo Tomasi hozzájárulásával).



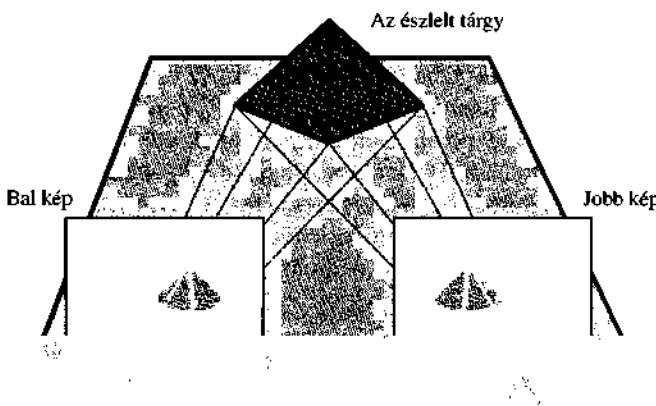
**24.9. ábra.** (a) A 24.8. ábrán látható kép tulajdonságai elhelyezkedésének háromdimenziós rekonstrukciója. (b) A valódi ház ugyanabból a perspektívából.

a  $T_z$  sebességi komponenst megadni, a kettő arányát meg tudja adni, és így a leszállás irányításában felhasználható. Az élő legyekkel való kísérletek alátámasztják, hogy a legyek pontosan ezt a mechanizmust használják. A legyek a legmerészebb repülők az állatok és gépek között, és érdekes, hogy ezt egy olyan látórendszerrel teszik, amelynek borzasztóan gyenge a térbeli felbontása (körülbelül 600 érzékelője van, összehasonlítva az emberek 100 milliójával), de kiemelkedő az időbeli felbontása.

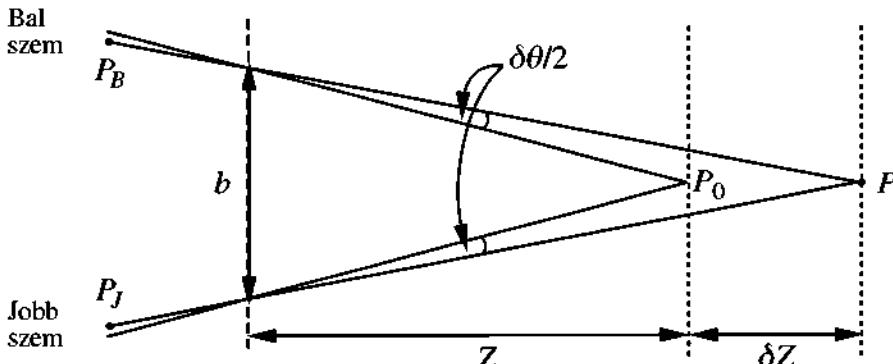
A mélység kiszámításához több képkocka használata szükséges. Ha a filmre egy merev testet veszünk fel, akkor ennek alakja képkockáról képkockára nem fog változni, és így az optikai folyam természeténél fogva zajos méréseit jobban kezelhetjük. Az egyik ilyen megközelítés eredményét a 24.8. és a 24.9. ábrán láthatjuk (Tomasi és Kanade, 1992).

## Kétkamerás (binokuláris) térbeli látás

A legtöbb gerincesnek két szeme van. Ez hasznos redundancia arra az esetre, ha elvezítené az egyiket, de más módon is segít. A legtöbb zsákmányállatnak a feje oldalán vannak a szemei, hogy nagyobb területet beláthasson. A ragadozóknak elől, hogy **kétkamerás térbeli látást (binocular stereopsis)** valósíthassanak meg. Maga a gondolat a mozgási parallaxisra igen hasonlít, azonban az eltérő időpontokhoz tartozó képek helyett két (vagy több), térben szeparált képet használunk, hasonlóan ahhoz, mint amilyeneket pl. az előrenéző emberi szemek szolgáltatnak. Tekintettel arra, hogy a jelenet egy adott jellemzője az egyes képsíkok z koordinátájához képest más helyen lesz a képen, ha a két képet egymásra fektetjük, a képjellemző elhelyezkedésében a két kép különbséget – **diszparitást (disparity)** – fog mutatni. Ez látszik a 24.10. ábrán, ahol a piramis legközelebbi pontja balra mozdult el a jobb oldali képen, és jobbra a bal oldalon.



**24.10. ábra.** A térbeli látás alapgondolata: különböző kamerapozíciók ugyanannak a háromdimenziós jelenetnek kissé eltérő kétdimenziós nézeteit eredményezik



24.11. A diszparitás és a mélység összefüggése a sztereolátásban

Dolgozzuk ki a diszparitás és a mélység geometriai kapcsolatát. Először azt az esetet vizsgáljuk meg, amikor a két szem (illetve a két kamera) előrenéz, és optikai tengelyük párhuzamos. A jobb és a bal kamera kapcsolata így egy  $b$  nagyságú (bázisvonal) transzláció az  $x$  koordináta-tengely mentén. A  $H = v_x \Delta t$  vízsintes és  $V = v_y \Delta t$  függőleges képeltérés számításához az optikai folyam előbb megismert kifejezéseit használhatjuk,  $T_x = b / \Delta t$ ,  $T_y = T_z = 0$  mellett. Az  $\omega_x$ ,  $\omega_y$ , és  $\omega_z$  rotációs paraméterek zérusok. Azt kapjuk eredményül, hogy  $H = b / Z$  és  $V = 0$ . Szavakkal megfogalmazva ez azt jelenti, hogy a vízsintes képeltérés a bázisvonal és a mélység aránya, a függőleges eltérés pedig zérus.

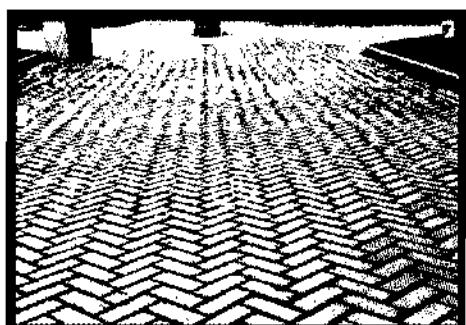
Normális látási körülmények között az emberek fixálnak, azaz **összetartóan néznek előre (fixate)**, vagyis a jelenetben létezik egy olyan pont, amelyben a két optikai tengely metszi egymást. A 24.11. ábra egy  $P_0$  pontra fixáló szemeket mutat, amely pont a két szem középvonalától  $Z$  távolságban helyezkedik el. Az egyszerűség kedvéért szögeltérést fogunk számítani radiánban. Az összetartás  $P_0$  pontjában az eltérés zérus. A jelenet egy  $\delta Z$ -vel távolabbi  $P$  pontja esetén számíthatjuk a  $P$  pont  $P_L$  bal és a  $P_R$  jobb oldali képének a szögeltolódását. Ha azok mindegyike a  $P_0$ -hoz képest  $\delta\theta/2$  szöggel eltolt, akkor a  $P_L$  és a  $P_R$  eltolódása, ami a  $P$  képeltérése, pontosan  $\delta\theta$ . Egyszerű geometriai átalakításokból azt fogjuk kapni, hogy:

$$\frac{\delta\theta}{\delta Z} = \frac{-b}{Z^2}$$

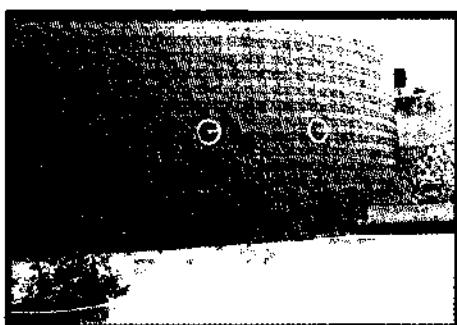
Embereknek a  $b$  **bázisvonal (baseline)** kb. 6 cm. Tegyük fel, hogy  $Z$  kb. 100 cm. A legkisebb detektálható  $\delta\theta$  (ami a képpont nagyságával függ össze) kb. 5 szögmásodperc, vagy  $2,42 \times 10^{-5}$  radián, amiből a  $\delta Z$  0,4 mm-re adódik.  $Z = 30$  cm-re  $\delta Z = 0,036$  mm-es lenyűgöző értéket kapunk. Ez azt jelenti, hogy 30 cm-es távolságban az emberek akár a 0,036 mm-es képmélység különbség megkülönböztetésére is képesek. Ez teszi lehetővé, hogy be tudjuk fűzni a cérnát egy tübe, és hasonló műveleteket tudunk elvégezni.

## Textúragradiensek

A minden nap nyelvben a **textúra** vagy **felületmintázat** (*texture*) a felület kitapintható tulajdonságaira vonatkozik (az angol nyelvben ugyanazzal a szótővel rendelkezik, mint maga a szöveg<sup>3</sup>). A számítógépes látásban a textúra egy ehhez közel fogalomra, egy felületnek a síkban ismétlődő, vizuálisan érzékelhető mintázatára vonatkozik. Példaként említhetjük egy épület ablakainak a mintázatát, egy szetter kötési mintáját, a leopárd foltos bundáját, a pázsit fűszálainak a mintázatát, kavicsokat a strandon vagy embertőmeget egy stadionban. Az elrendezés néha közel periodikus, mint például a kötési szemek a szetteren, más esetekben, mint például a strandon lévő kavicsok esetén, a szabályosság csak statisztikai értelemben létezik – a kavicsok sűrűsége a strand különböző részein nagyjából azonos.



(a)



(b)

**24.12. ábra.** (a) A textúragradienseket illusztráló jelenet. Feltéve, hogy a valós mintázat egyforma, lehetővé teszi a felület irányágának meghatározását. A számított irányt a képen oly módon elhelyezett fehér kör és nyíl jelzi, mintha a kör a felületre festették volna abban a pontban. (b) Egy ívelt felület alakjának meghatározása a mintázat alapján. (A képeket Jitendra Malik és Ruth Rosenholtz hozzájárulásával közöljük [Malik és Rosenholtz, 1994].)

Amit most a *jelenetre* vonatkozóan állítottunk, igaz. A képen a textúraelemek vagy **texelek** (*texels*) látszólagos nagysága, alakja, távolsága stb. igencsak változik, ahogy ezt a 24.12. ábra mutatja. A csempelapok a jeleneten azonosak. A lapok kivételttel nagyságának és alakjának megváltozásában két fő ok játszik szerepet:

1. Az egyes texeleknek a kamerától vett változó távolsága. Emlékezzünk arra, hogy a perspektivikus vetítésben a távoli tárgyak kisebbnek tűnnek. A skálatényező  $1/Z$ .
2. Az egyes texelek változó rövidülése. Ez a texeleknek a kamerából kiinduló rálátási irányhoz viszonyított orientációján múlik. Rövidülés nincs, ha a texel a rálátási irányra merőleges. A rövidülés mértéke  $\cos \sigma$ -val arányos, ahol a  $\sigma$  a texel síkjának a lejtése.

Egy kevés matematikai elemzést követően meghatározhatjuk a különböző texeltulajdonságok, mint pl. a terület, a rövidülés és a sűrűség változási mértékének kifejezéseit.

<sup>3</sup> avagy „textile” (A ford.)

Ezek az ún. **textúragradiensek (texture gradients)** a felület alakjának és a megfigye-lőhöz képesti dőlésének és lejtésének a függvényei.

Ahhoz, hogy az alakot a textúrából kinyerhessük, kétlépcsős eljárásra kell folyamod-nunk: (a) először mérjük a textúragradienseket; (b) majd becsüljük a mért gradienseket okozni látszó felületalakot, azok dőlését és lejtését. A 24.12. ábra mutatja ezen eljárás eredményeit.

## Árnyalás

**Az árnyalást (shading)** – a jelenethez tartozó felület különböző részeiről kapott meg-világítás intenzitásváltozását – a jelenet geometriája és a felület-visszaverési tulajdonságai határozzák meg. A számítógépes grafikában a feladat a kép  $I(x, y)$  fényességfügg-vényének a megadása, ha a jelenet geometriája és a visszaverődési tulajdonságok adottak. A számítógépes látásban azt reméljük, hogy ez a folyamat invertálható, vagyis visszaál-líthatjuk a jelenet geometriáját és visszaverődési tulajdonságait, ha az  $I(x, y)$  képfényesség adott. A probléma olyan nehéznek bizonyult, hogy a legegyszerűbb esetektől eltekintve e téren nem sikerült eredményeket elérni.

Indulunk ki egy olyan példából, ahol az alakot tényleg visszanyerhetjük az árnyalás-ból. Tekintsünk egy Lambert-féle felületet, amit egy távoli, pontszerű fényforrás világít meg. Tételezzük fel azt is, hogy a felület messze van a kamerától, így függőleges síkú vetítést fogunk használni a perspektivikus vetítés megközelítésére. A kép fényessége:

$$I(x, y) = k\mathbf{n}(x, y) \cdot \mathbf{s}$$

ahol  $k$  egy skálaegyüttható,  $\mathbf{n}$  a felület normál egységvektora és  $\mathbf{s}$  a fényforrás irányába mutató egységvektor. Mivel  $\mathbf{n}$  és  $\mathbf{s}$  egységvektorok, skalárszorzatuk a kettő közötti szög koszinusra. A felület alakját az  $\mathbf{n}$  felület menti változása tartalmazza. Tételezzük fel, hogy  $k$  és  $\mathbf{s}$  ismertek. A problémánk most a felület  $\mathbf{n}$  normálisának a kiszámítása, ha az  $I(x, y)$  képfényesség adott.

Az első megjegyzés, amit meg kell tenni, az az, hogy az  $\mathbf{n}$  meghatározása az adott  $(x, y)$  képpontfényesség ismeretében lokálisan alulhatározott. Ki tudjuk ugyan számítani az  $\mathbf{n}$  és a fényforrás közötti szöget, ez azonban az  $\mathbf{n}$ -et csak annyiban korlátozza, hogy egy bizonyos,  $\mathbf{s}$  irányú és  $\theta = \cos^{-1}(I/k)$  palástszögű kúp felületén helyezkedik el. Hogy továbbléphessünk, jegyezzük meg, hogy képpontról képpontra az  $\mathbf{n}$  változása nem lehet tetszőleges. Egy sima felület normálisa csak sima módon változhat – amely korlátozást az **integrálhatóság (integrability)** szakkifejezéssel jelöljük. Ezt a felismerést számos módszerben fel is használják. Egy lehetséges eljárás az  $\mathbf{n}$  átfrása a  $Z(x, y)$  mélység  $Z_x$  és  $Z_y$  parciális deriváltjainak felhasználásával. Ennek eredménye  $Z$  egy parciális differenciálegyenlete, amit megfelelő határfeltételek figyelembevételével  $Z(x, y)$ -ra nézve meg lehet oldani.

Az eljárás általánosítható. Nem szükséges, hogy a felület Lambert-féle legyen, és az sem, hogy a fényforrás pontszerű legyen. Lényegében ugyanez a módszer használható, ha sikerül a felület  $R(\mathbf{n})$  **reflektanciatérképét (reflectance map)** meghatározni, amely a felületelem fényességét annak  $\mathbf{n}$  normálisa függvényében fejezi ki.

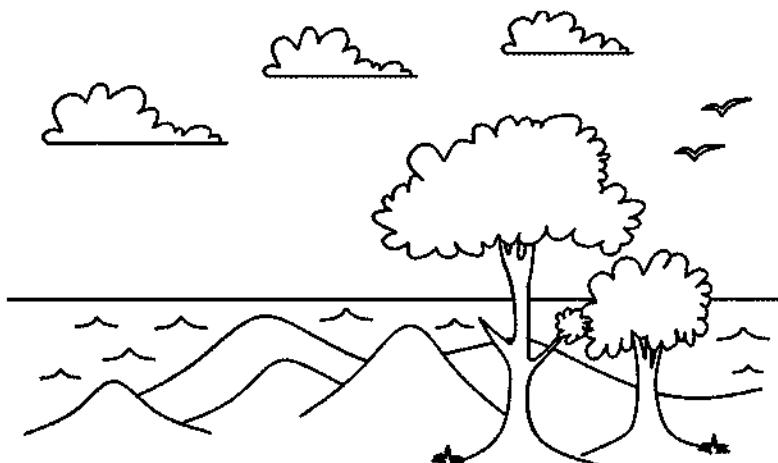
Az igazi nehézség a kölcsönös visszaverődések kezelése. Egy tipikus belsőterjelenet esetén, mint például egy irodában lévő tárgyak esetén a felületeket nem csupán a fény-

források világítják meg, hanem a jelenet más felületeitől visszavert fény is, amelyek tényleges másodlagos fényforrásokként szolgálnak. Ezek a kölcsönös megvilágítási hatások igen jelentősek lehetnek. Az ilyen helyzetben a reflektanciatérkép megközelítés teljes mértékben használhatatlan – a kép fényessége nemcsak a normálisan műlik, hanem a jelenet különböző felületeinek bonyolult térbeli viszonyain is.

Az ember az árnyalásból képes valamilyen mértékű alakinformáció kinyerésére, így ez a probléma – minden nehézsége ellenére – továbbá is érdekes marad.

## Kontúrok

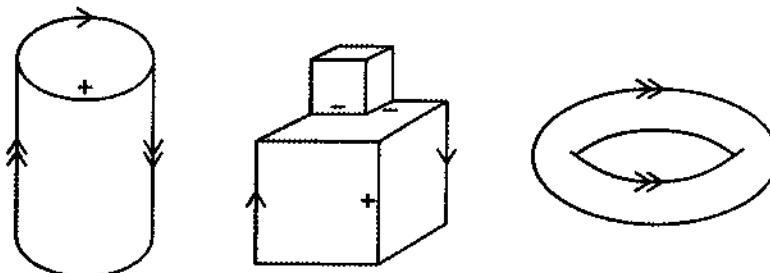
Amikor egy olyan vonalas rajzot nézünk, mint amilyen a 24.13. ábra, a kép egy háromdimenziós elrendezés életszerű benyomását kelti bennünk. Hogyan lehetséges ez? Hiszen láttuk korábban, hogy egy és ugyanaz a vonalas kép végtelen sok jelenet konfigurációból származhat. Figyeljük meg, hogy még a felszín lejtésének és emelkedésének érzetét is megkapjuk. Ez minden bizonnal (a tipikus alakzatokra vonatkozó) magas szintű tudás és valamilyen alacsony szintű kényszerek összekapcsolásának következtében lehetséges.



24.13. ábra. Egy felidéző vonalas rajz (Isha Malik hozzájárulásával)

A vonalas ábrában rejлő kvalitatív tudással fogunk foglalkozni. Korábban láttuk, hogy a rajz vonalainak különféle lehet a fontossága (lásd 24.4. ábra és kísérőszövege). Az a folyamat, ahogy a kép minden vonalának a tényleges fontosságát megállapítjuk, a vonalcímkezés (line labeling), és ez volt a számítógépes látás által elsőként tanulmányozott problémák egyike. Tegyük fel egyelőre, hogy a világ olyan leegyszerűsített modelljével dolgozunk, ahol a tárgyaknak nincsenek felületei, és ahol az olyan vonalakat, amelyek megvilágítási diszkontinuitásokból adódnak, mint amilyenek az árnyékélek és a tükrözések, valamilyen előfeldolgozó eljárással a képből kiemeltük. A figyelmünket így olyan vonalas ábrára összpontosíthatjuk, ahol minden vonal vagy mélységi, vagy orientáció diszkontinuitáshoz tartozik.

Mindegyik vonalat ezek után vagy egy a felület határán megjelenő **pszeudoél** (**limb**) vettületének (ami a felület azon pontjainak összessége, ahol a rálátás iránya a felület érintője), vagy egy **élnek** (**edge**) lehet tekinteni (ami a felület normálisának egy diszkontinuitása). Az éleket továbbá konvex, konkáv vagy határoló élosztályokba sorolhatjuk. A pszeudoélek és a határoló élek esetén érdekes annak az eldöntése, hogy a vonalas ábrán a görbület határoló két felületből melyik van hozzánk közelebb. Ezeket a következetésekkel úgy képzelhetjük el, hogy minden vonalhoz a 6 lehetőséges **vonalcímkének** (**line label**) az egyiket rendeljük hozzá, ahogy az a 24.14. ábrán látható.



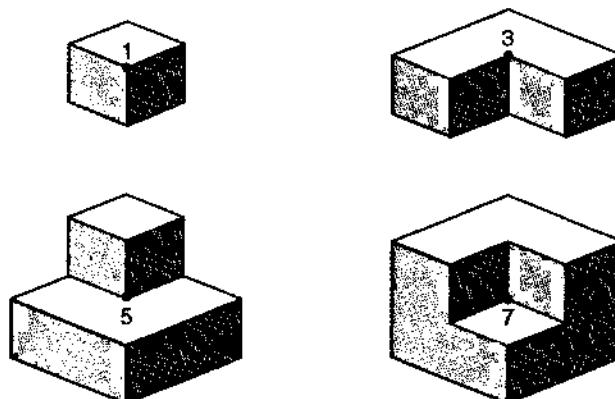
24.14. ábra. Különféle vonalcímek

1. A „+” és a „-” címkék rendre a konvex és a konkáv éleket jelentik. Ezeket az olyan normális diszkontinuitásokhoz rendeljük, ahol az él mentén találkozó minden felület látható.
2. A „←” és a „→” határoló konvex éleket jelentenek. A kamerával felülről nézve az él mentén találkozó minden felületelem azonos oldalon fekszik úgy, hogy az egyik eltarthatja a másikat. Ha a nyíl irányába mozgunk, a felületek hozzánk képest jobbra esnek.
3. A „←←” és a „→→” címkék a pszeudoéleket jelölik. Az ilyen helyeken a felület sima módon görbülni, és saját magát takarja. Ha a dupla nyíl irányába lépünk, a felület tőlünk jobbra helyezkedik. A rálátás iránya a pszeudoél minden pontjában a felület érintője. Ahogy a nézőpont változik, a pszeudoél a felület mentén mozog.

Egy ábrán található  $n$  számú vonal kombinatorikusan elvileg lehetséges címke-hozzárendelésből csupán csekély számú hozzárendelés lehetséges fizikailag. Ezeknek a címkezéseknek a megállapítása az ún. vonalcímkezési probléma. Jegyezzük meg, hogy a probléma csak akkor értelmes, ha a címke a vonal mentén nem változik. Ez nem minden igaz, hiszen a görbült objektumok képein a címke egy vonal mentén igenis változhat. Ebben a részben ez nem jelent problémát, mert csak poliéderes objektumokkal foglalkozunk.

A poliéderes jelenetelemzéssel először Huffman és tőle függetlenül Clowes próbálkoztak (Huffman, 1971; Clowes, 1971). Huffman és Clowes elemzésüket a nem átlátszó **triéderes** (**trihedral**) testekből – azaz olyan testekből, amelyeknél minden sarokpontjánál pontosan három felület fut össze – álló jelenetek esetére korlátozták. A több objektumot tartalmazó jelenetek esetén kizárták a triéderes feltételt megsértő objektumkonfigurációkat is, mint amilyen például az egyik élük mentén érintkező két kocka.

Nem engedték a **repedéseket (cracks)** sem, azaz olyan „éleket”, amelyek mentén a tangenciális síkok folytonosan változnak. A triéderes világban Huffman és Clowes kimerítő listát adtak az összes különböző sarokponttípusról meg arról is, hogy azok hogyan látszhatnak általános nézőpontból szemlélve. Az általános nézőpont feltétel lényegében azt tételezi fel, hogy az összes csatlakozás megtartja a jellegét, ha a szem csak kissé mozog el. Ha például a képen három vonal metszi egymást, akkor a feltételből következik, hogy a jelenet hozzájuk tartozó éleinek szintén metszeniük kell egymást.

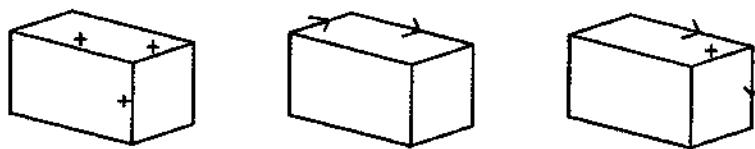


24.15. ábra. A triéderes csúcsok négy fajtája

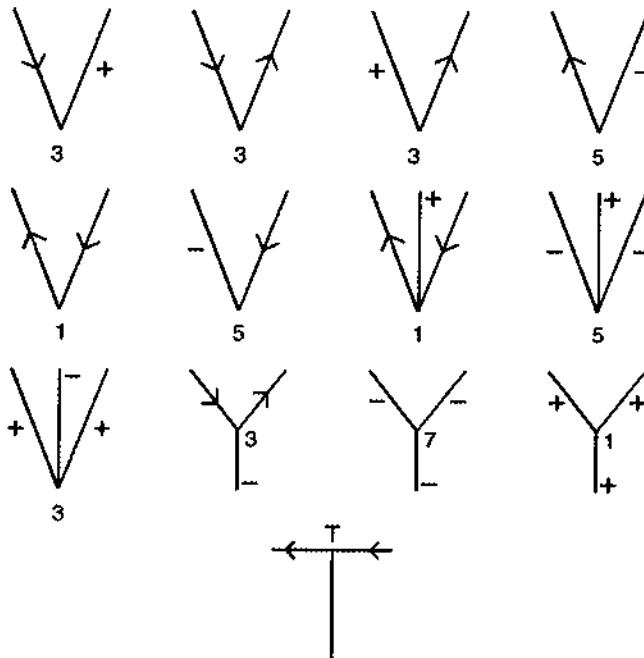
A 24.15. ábra mutatja azt a négyféle helyzetet, ahogy a három síkfelület egy sarokpontban találkozhat. Ezeket az eseteket úgy szerkesztettük, hogy egy kockát nyolc oktánsra (octant) bontottunk. A triéderes sarokpontok egész választékát szeretnénk generálni a kocka középpontjában úgy, hogy a különböző oktásokat kitöltsük. Az 1-es címkéjű sarokpont az egyetlenegy kitöltött oktánshoz tartozik, a 3-as címkéjű a három kitöltött oktánshoz és így tovább. Az olvasónak kellene belátnia, hogy ezek tényleg kimerítik az összes lehetséges helyzetet. Ha valaki például a kockában két oktánst kitölt, akkor nem lesz képes a középpontban egy érvényes triéderes sarokpontot meg-szerkeszteni. Jegyezzük meg azt is, hogy ezen négy eset a sarokpontban találkozó konvex és konkav élek különböző kombinációinak felel meg.

A sarokpontban találkozó három él a körülöttük lévő teret nyolc oktánsra osztja. A sarokpontot bármely, anyaggal nem kitöltött oktánsból lehet szemlélni. A nézőpont oktánson belüli elmozdítása a képen látható csatlakozások jellegét megváltoztatni nem fogja. A 24.15. ábrán látható 1-es címkéjű sarokpontot a maradó hét oktánrból szemlélve a 24.16. ábrán látható csatlakozáscímkéket kapjuk.

Annak kimerítő vizsgálata, hogy egy sarokpont milyen módon látható, a 24.17. ábrán felsorolt lehetőségekhez vezet. A képen négy különböző csatlakozástípust azonosíthatunk: L, Y, nyíl- és T csatlakozást. Az L csatlakozás a két látható él esete. Az Y és a nyílcsatlakozás három élnek felel meg. Az Y csatlakozásban a szögek egyike sem lépi túl a  $180^\circ$ -ot. A T csatlakozás a takarással kapcsolatos. Ha egy közelí, nem átlátszó felület egy távolabbi élét eltakar, egy fél éllel találkozó folytonos él kapunk. A T csatlakozás négy címkéje a négyféle él takarásának felel meg.



24.16. ábra. A 24.15. ábrán 1-gyel jelölt csúcs különböző előfordulásai



24.17. ábra. A Huffman-Clowes-címkekészlet

Ha egy vonalas rajzot ennek a csatlakozásszótárnak a segítségével címkézünk meg, a probléma annak a megállapítása, hogy mely csatlakozásinterpretációk lesznek globálisan konzisztensek. A konziszenciát az a szabály kényszeríti ki, miszerint a rajzon egy vonalnak egy és csakis egy címkéje lehet a vonal egész hosszában. Ezen probléma algoritmikus megoldását Waltz adta meg (Pontosabban egy olyan kibővített probléma megoldását adta meg, amely az árnyékot, a görbületi diszkontinuitásokat [repedéseket] és a szétválaszthatóan konkáv éleket is figyelembe veszi [Waltz, 1975]). Az algoritmus a kényszerielégítés egyik legelső alkalmazása volt az MI területén (lásd 5. fejezet). A kényszerielégítés nyelvén a változók a csatlakozások, a változók értékei a csatlakozások címkézése, a kényszer pedig az, hogy minden vonalnak csak egy címkéje van. Bár a triéderes vonalcímkézési probléma NP-teljes, a közönséges kényszerielégítési algoritmusok a gyakorlatban jól működnek.

## 24.5. OBJEKTUMOK FELISMERÉSE

A látás lehetővé teszi számunkra, hogy megbízhatóan felismerjünk embereket, állatokat és élettesen dolgokat. Az MI-ben, illetve a számítógépes látásban az *objektum-felismerés* fogalmát szokás szerint minden dolgokra alkalmazzák. Ez magában foglalja a képen rögzített objektumok osztályának felismerését – például: ez egy arc, illetve adott objektumok felismerését is – például: ez Bill Clinton arca. A motivációt ilyen alkalmazási területek jelentik:

- **Biometrikus azonosítás (biometric identification):** a bűnügyi vizsgálatok és a védett létesítmények belépésszabályozása megköveteli az egyének azonosítását. Az ujjlenyomatok, az íriszlenyomat és az arcképek olyan képeket eredményeznek, amelyeket adott egyénekhez kell rendelni.
- **Tartalommalapú képkeresés (content-based image retrieval):** könnyű egy dokumentumban megtalálni azt a helyet – ha létezik –, ahol a „macska” szó szerepel. Bár melyik szövegszerkesztő képes erre. Most gondoljon arra a feladatra, hogy egy képen megtalálja a képpontok azon részhalmazát, amely egy macska képének felel meg. Ha valaki rendelkezne ezzel a képességgel, akkor megválaszolhatna olyan kereséseket, mint például „Bill Clinton és Nelson Mandela együtt”, egy „görkoris a levegőben”, „az Eiffel-torony éjszaka” stb., anélkül hogy egy gyűjtemény minden egyes fényképhez feliratkulcsszavakat kellett volna rendelnünk. Ahogy a kép- és videogyűjtemény növekszik, a kézi feliratozás lehetetlenné válik.
- **Kézírás-felismerés (handwriting recognition):** például az aláírások, borítékok címzözői, csekkeken szereplő összegek és a kéziszámítók tollalapú beviteli megoldása.

A látást nemcsak objektumok, hanem cselekvések felismerésére is használjuk. Felismerhetünk járásmódokat (egy barát járását), kifejezéseket (egy mosolyt, egy grimaszt), gesztikulációt (egy intégető embert), cselekvéseket (ugrást, táncot) és így tovább. A cselekvésfelismerést célzó kutatás még gyerekcipőben jár, így ebben az alfejezetben az objektumfelismerésre koncentrálunk.

A vizuális objektumok felismerésének feladata általában könnyű az emberek számára, de a számítógépek számára igen nehéznek bizonyult. Fel akarjuk ismerni egy ember arcát függetlenül a megvilágítás, a kamerához képesti helyzet és az arckifejezés különböző variációitól. Ezen variációk bármelyike széles körű változásokat eredményez a képpontok fényességértekben, így a képpontok közvetlen összehasonlítása valószínűleg nem fog működni. Ha valaki egy kategória (például az „autó”) példányait szeretné felismerni, akkor a kategórián belüli variációkat is figyelembe kell vennie. Még a postai irányítószámoknál a kézzel írt számjegyek felismerésének igen korlátozott feladata is komoly kihívásnak bizonyult.

A felügyelt tanulás vagy a mintaosztályozás természetes keretet biztosít az objektum-felismerés tanulmányozásához. Pozitív („arcok”) és negatív példaként („nem arcok”) adott képek esetében a cél egy olyan függvény megtanulása, amely az új képeket felcímkelzi az *arc* és a *nem arc* jelölésekkel. A 18. és 20. fejezet összes technikája lehetséges jelölt a feladatra: többrétegű perceptronokat, döntési fákat, legközelebbi-szomszéd osztályozókat és kernelgépeket alkalmaztak már objektumfelismerési feladatokra. Meg kell azonban jegyeznünk, hogy ezen technikák alkalmazása objektumok felismerésére távolról sem egyértelmű.

Az első kihívás a kép szegmentálása. minden kép tipikusan több objektumot fog tartalmazni, így először olyan képpontrészhalmazkra kell osztanunk, amelyek egy-egy objektumnak felelnek meg. Miután a képet régiókra osztottuk, utána a régiókat vagy az azokból összeállított egységeket átadjuk egy osztályozónak, hogy meghatározza az objektumok címkéit. Sajnálatos módon a lentről felfelé szegmentáció hibákra érzékeny eljárás, ezért alternatív megoldásként kínálkozik a fentről lefelé objektumcsapat meghatározása. Eszerint keresni kell egy képpontrészhalmazt, amit arcként lehet osztályozni, és ha ez sikerült, már van egy csoportunk! A tisztán fentről lefelé módszerek nagyon számításigényesek, mivel különböző méretű képablakokat kell megvizsgálni különböző helyszíneken, és az összes különböző objektumhipotézissel össze kell vetni őket. Jelenleg a leginkább gyakorlati objektumfelismerő rendszerek ilyen fentről lefelé stratégiát alkalmaznak, bár ez megváltozhat, ahogy a lentről felfelé technikák fejlődnek.

A második kihívás annak biztosítása, hogy a felismerés folyamata kellően robusztus a megvilágítás és az elhelyezkedés változásaira. Az emberek fel tudnak ismerni objektumokat a képpontok fényességéről, mint pontos megjelenés jelentős változásai ellenére is. Például egy barát arcát különböző fényviszonyok mellett vagy különböző látószögekből is felismerjük. Egy ennél is egyszerűbb példaként megvizsgálhatjuk a kézzel írt 6-os számjegy felismerését. Képesnek kell lenni a 6-os felismerésére különböző méretekben, a kép különböző pozícióiban és a kép kisebb elforgatásai ellenére is.<sup>4</sup>

A kulcsmomentum azt észrevenni, hogy a geometriai transzformációk, mint például az elfordítás, a méretváltás és a forgatás, vagy a kép fényességének megváltozásai a fényforrások fizikai mozgatása által, más jellemzőkkel rendelkeznek, mint a kategórián belüli variációk, mint például az emberek arcainak különbözőségei. Nyilvánvaló, hogy a tanulás az egyetlen mód az emberi arcok különbözőségeinek megtanulására vagy a 4-es számjegy különböző írásmódjainak felismerésére. Másrészről a geometriai és fizikai transzformációk hatásai szisztematikusak és kiküszöbölhetők a tanuló minták paramétereit leíró tulajdonságok megfelelő tervezésével.

Annak érdekében, hogy a geometriai transzformációra vonatkozó invarienciát biztosítsuk, egy egészen hatékonyan bizonyult eljárás a kép régiójának előfeldolgozása úgy, hogy standard pozícióba, méretre és orientációra alakítjuk. Egy alternatív megoldás az, ha egyszerűen figyelmen kívül hagyjuk a geometriai és fizikai transzformációk alkalmi természetét, és úgy gondolunk rájuk, mint az osztályozó variálhatóságának egy újabb forrására. A tanító minták halmazában az összes ilyen variációra példákat kell elhelyeznünk, és abban reménykedünk, hogy az osztályozó kikövetkeztet egy megfelelő transzformációhalmazt a bemenetre, és így kiszűri a variációkat.

Vizsgálunk meg most az objektumfelismerésre szolgáló speciális algoritmusokat! Az egyszerűség érdekében kétdimenziós összeállításban fogjuk a problémát vizsgálni, ahol mind a tanuló, mind a tesztminták kétdimenziós fényességábrák formájában adottak. Olyan területeken, mint például a kézirás-felismerés, ez nyilvánvalóan elegendő. Még háromdimenziós objektumok esetében is hatékony stratégia az objektumokat több kétdimenziós nézettel reprezentálni (lásd 24.18. ábra), és az új objektumokat a (valami-ilyen reprezentációjú) tárolt nézetekkel összehasonlítva osztályozni.

<sup>4</sup> A teljes elforgatásinvariancia nem szükséges és nem is kívánatos – ekkor a 9-es és a 6-os összekoverhető lenne!



**24.18. ábra.** Két háromdimenziós tárgy többféle nézete

Az előző alfejezet megmutatta, hogy több tényező vesz részt a háromdimenziós információ kinyerésében egy jelenetből. Az objektumfelismerés is több tényezőn alapszik – egy tigrist a narancs és fekete színek keveredéséről, csíkos mintázatáról és testének alakjáról ismerünk fel.

A szín és a mintázat hiszogramok vagy empirikus frekvenciaeloszlások segítségével reprezentálható. Ha adott egy tigris képe mintaként, akkor megmérhetjük a különböző színű képpontok arányát. Ezután, ha egy ismeretlen példát látunk, összevethetjük a szín hiszogramját a korábbi tigrispéldánknál látottakkal. A mintázatok elemzéséhez a kép különböző irányú és skálázású szűrőkkel történt konvolúciója után kapott eredményeinek hiszogramjait vizsgáljuk, egyezést keresve.

Az alak felhasználása az objektumfelismerésben sokkal bonyolultabbnak bizonyult. Nagyjából két irányzatot különbözthetünk meg: **fényességalapú felismerést** (**brightness-based recognition**), ahol a képpontok fényességértékeit direkt módon használjuk, és **jellemzőalapú felismerést** (**feature-based recognition**), ahol kiemelt jellemzők, mint például az élek vagy a kulcspontok részleges elrendezését használjuk. Miután részletesebben megvizsgáltuk e két megközelítést, tárgyalni fogjuk a **pozícióbecslést** (**pose estimation**), azaz az objektum helyzetét és irányát a jelenetben.

## Fényességalapú felismerés

Ha adott egy képponthalmaz, amely egy lehetséges objektumnak felel meg, akkor válaszszuk a jellemzőknek magukat a képpontok fényességeit. Egy másik variáció szerint először konvolválhatjuk a képet többféle lineáris szűrővel, és az eredménytől kapott kép pontjainak értékeit tekinthetjük a jellemzőknek. Ez a megközelítés különösen sikeres olyan feladatokban, mint a kézzel írt számjegyek felismerése, ahogy azt a 20.7. alfejezetben láthattuk.

Sokféle statisztikai módszert használtak arra, hogy képeket tartalmazó adatbázisok segítségével arcfelismerőket fejleszzenek ki, mint például feldolgozatlan képpontbemenetben működő neurális hálókat képpontbemenettel, vonal- és élszűrők által előállított



**24.19. ábra.** Egy arcmeztaláló algoritmus eredménye (Henry Schneiderman és Takeo Kanade hozzájárulásával)

jellemzőket használó döntési fákat és wavelet-jellemzőket használó, naiv Bayes-modellket. Az utóbbi módszer néhány eredménye a 24.19. ábrán látható.

A feldolgozatlan képpontok jellemző vektorként történő használatának egy negatív aspektusa az ebben a reprezentációban rejlik hatalmas redundancia. Vegyük például két egymás mellettí képpontot az arc pofa részén; valószínűleg igen erős közöttük a korreláció, hiszen hasonló a geometriájuk, megvilágításuk stb. Az adatmennyiséget csökkentő technikák, mint például a főkomponens-analízis, sikeresen használhatók a jellemzővektor dimenziószámának csökkentésére, lehetővé téve az olyan dolgok, mint például arcok, nagydimenziós terekben való felismerésénél nagyobb sebességű felismerését.

## Jellemzőalapú felismerés

A feldolgozatlan képpontok fényességértékei jellemzőként történő felhasználása helyett felismerhetünk adott helyen levő jellemzőket, mint például régiókat vagy éleket (lásd 24.3. alfejezet). Két motivációt van az élek használatára. Az egyik az adatmennyiség csökkentése – sokkal kevesebb él van, mint képpont. A másik a megvilágításfüggelenség – a kontraszt egy megfelelő tartományában az éleket nagyjából azonos helyen ismerjük fel, függetlenül a tényleges fényforrás-konfiguraciótól. Az élek egydimenziós jellemzők; kétdimenziós és nulladimenziós tulajdonságokat (régiókat, illetve pontokat) szintén használunk. Vegyük észre a térbeli elhelyezkedés kezelésében jelentkező különbséget a fényességalapú és a jellemzőalapú módszerek között! A fényességalapú módszerekben az  $(x, y)$  hely a jellemző.

Az élek elrendezése egy objektum tulajdonsága – ez az egyik oka, ami miatt a rajzokat könnyedén értelmezni tudjuk (lásd 24.13. ábra), még akkor is, ha ilyen képek nem

fordulnak elő a természetben! Ezen tudás legegyszerűbb felhasználásának módja egy legközelebbi-szomszéd osztályozóval történhet. Előre kiszámítjuk és eltároljuk az ismert objektumok nézeteihez tartozó élkonfigurációkat. Ha adott egy ismeretlen objektumnak megfelelő élelrendezés a bemeneti képen, akkor meghatározhatjuk az eltárolt nézetekhez mért „távolságát”. A legközelebbi-szomszéd osztályozó a legközelebbi illeszkedő választja.

Sokféle definíciót javasoltak a képek közötti távolságra. Az egyik legérdekesebb megközelítés a **deformációs illeszkedés** (**deformable matching**) ötletén alapul. D'Arcy Thompson klasszikus munkájában (*On Growth and Form* [D'Arcy Thompson, 1917]) megfigyelte, hogy a hasonló, de nem azonos alakzatok gyakran egybeesésbe hozhatók egyszerű koordináta-transzformációkkal.<sup>5</sup> E módszer szerint az alakzatok hasonlóságát egy három lépésből álló folyamattal határozzuk meg: (1) megoldjuk a két alakzat megfeleltetési problémáját, (2) a megfeleltetést felhasználjuk egy egymásra illesztési transzformáció megbecslésére, és (3) kiszámítjuk a két alakzat közötti távolságot az egymásnak megfelelő pontok illeszkedési hibáinak összegeként, kiegészítve a fedésbe hozó transzformáció nagyságával.

Az alakzatot a belső vagy külső körvinonalát alkotó pontok diszkrét halmazaként reprezentáljuk. Ezeket előállíthatjuk egy éldetektáló által megtalált élképpontok helye alapján, így kapunk egy  $N$  pontból álló  $\{p_1, \dots, p_N\}$  halmazt. A 24.20. (a) és (b) ábra két alakzat mintapontjait mutatja.

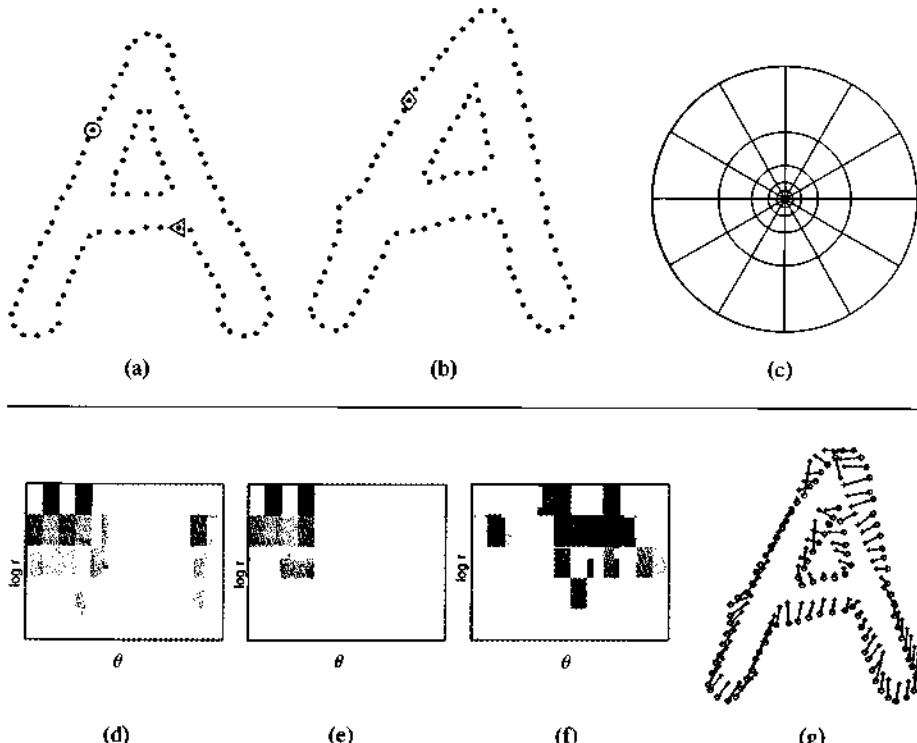
Most tekintsünk egy  $p_i$  mintapontra és az onnan induló összes többi mintapontra mutató vektorra egy alakzaton. Ezek a vektorok kifejezik a teljes alakzat konfigurációját a referenciaponthoz viszonyítva. Ez a következő ötlethez vezet: rendeljünk minden mintaponthoz egy lefrót, az **alakzatkontextust** (**shape context**), amely megadja az alakzat további részének durva elrendezését az adott ponthoz viszonyítva. Pontosabban fogalmazva a  $p_i$  alakzatkontextusa a maradék  $N - 1$   $p_k$  pontra a  $p_k - p_i$  relatív koordináták egy közelítő  $h_i$  térbeli hiszogramja. Egy logaritmikus poláris koordináta-rendszert használunk a tartók meghatározására annak biztosítása érdekében, hogy a lefró érzékenyebb legyen a közelíti képpontok esetében. A 24.20. (c) ábra mutat erre egy példát.

Vegyük észre, hogy az alakzat kontextusának definíciója magában hordozza az elforgatásinvarianciát, hiszen minden mérést az objektum pontjaihoz képest végezünk. A méret-invariancia eléréséhez minden sugárirányú távolságot normalizálunk a pontpárok közötti átlagos távolsággal.

Az alakzatkontextus lehetővé teszi, hogy a megfeleltetési problémát megoldjuk két hasonló, de nem azonos objektumra, mint ahogy az a 24.20. (a) és (b) ábrán látszik. Az alakzatkontextus különböző lesz egy  $S$  alakzat különböző pontjaira, de egy  $S$  és egy  $S'$  alakzat megfelelő (homológ) pontjainak alakzatkontextusa hajlamos lesz a hasonlóságra. Ezek után a két alakzat megfeleltethető pontjai megtalálásának problémáját akként kezelhetjük, mint hasonló alakzatkontextussal rendelkező párok megtalálását.

Precízebben fogalmazva: tekintsük a  $p_i$  pontot az első, a  $q_j$  pontot a második alakzaton. Jelölje a  $C_{ij} = C(p_i, q_j)$  a két pont illesztésének költségét. Minthogy az alakzatkontextusok hiszogramokkal reprezentált eloszlások, így természetes a  $\chi^2$  távolság használata:

<sup>5</sup> A modern számítógépes grafikában erre az ötletre az áttűnessel, **morfolással** (**morphing**) hivatkoznak.



**24.20. ábra.** Alakzatkontextus-számítás és -illesztés. (a, b) Két alakzat élének mintapontjai. (c) Az alakzatkontextus számítására használt log-polár hisztogram rekeszek ábrája. 5 rekesz használunk a log  $r$  és 12 rekesz  $\theta$  számára. (d-f) Az (a) és (b) ábrán jelölt referenciaminták alakzatkontextusai:  $\circ$ ,  $\diamond$ ,  $\triangleleft$ . minden alakzatkontextus a maradék pontalmaz koordinátáinak log-polár hisztogramja a referenciaPontot mint origót használva. (A sötét cellák több pontot jelentenek a rekeszben.) Vegye észre a  $\circ$  és  $\diamond$  pontok alakzatkontextusainak vizuális hasonlóságát, amelyeket a két alakzat relatíve hasonló pontjaira számoltunk! Ezzel ellentétben a  $\triangleleft$  alakzatkontextusa egészben eltérő. (g) Az (a) és (b) között kétoldali illesztéssel megtalált megfeleltetések, a hisztogramok  $\chi^2$  távolsága által definiált költséget használva.

$$C_{ij} = \frac{1}{2} \sum_{k=1}^K \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

ahol  $h_i(k)$  és  $h_j(k)$  a normalizált hisztogram  $k$ -adik értékét jelöli a  $p_i$ , illetve a  $q_j$  pontokban. Ha adott az első objektum  $i$  és a második objektum  $j$  pontjainak összes lehetséges párosítására a  $C_{ij}$  költségek halmaza, akkor az illesztés teljes költségét akarjuk minimalizálni azzal a kényszerfeltételel, hogy az illeszkedésnek kölcsönösen megfelelőnek kell lennie. Ez egy példa a **súlyozott páros illesztés (weighted bipartite matching)** problémájára, amely  $O(N^3)$  időben megoldható a magyar algoritmussal.

A mintapontok megfeleltetése alapján a teljes alakzatra kiterjeszhető a megfeleltetés annak a fedési transzformációnak a megbecsülésével, amely az egyik alakzatot a másikra képezi le. A regularizált vékonylemez spline-ok (thin plate spline) különösen hatékonyak. Ha az alakzatokat fedésbe hoztuk, a hasonlósági mértékek kiszámítása viszonylag

kézenfekvő. A két alakzat közötti távolság definiálható az egymásnak megfelelő pontok alakzatkontextus-távolságainak és a vékonylemez spline-hoz rendelt kötési energiának a súlyozott összegeként. Ezt a távolságmértéket meghatározva egy egyszerű legközelebbi-szomszéd osztályozó megoldja a felismerési problémát. A 20. fejezetben írtuk le ezen módszer kitűnő teljesítményét a kézzel írt számjegyek osztályozásában.

## Az elhelyezkedés becslése

Amellett hogy meghatározzuk, mi is az objektum, a helyzetét is szeretnénk megállapítani, azaz a nézőhöz képesti pozícióját és az irányát. Például egy ipari alkalmazásban a robot karja nem tud felvenni egy objektumot, amíg nem tudja a helyzetét. Merev két-, illetve háromdimenziós objektumok esetében a problémának egy egyszerű és jól definiált megoldása van, amely az **illesztési módszeren (alignment method)** alapszik, amit a következőkben fejtünk ki.

Az objektumot  $M$  jellemzővel, avagy  $m_1, m_2, \dots, m_M$  háromdimenziós ponttal reprezentáljuk – például egy soklapú objektum sarokpontjaival. Ezeket valamilyen, az objektumnak természetes koordináta-rendszerben határozzuk meg. Ezek után a pontokat egy ismeretlen háromdimenziós forgatásnak,  $R$ -nek vetjük alá, amit egy ismeretlen t mértékű eltolás követ, végezetül egy vetítés következik, amivel előállnak a képsík  $\{p_1, p_2, \dots, p_N\}$  jellemzői. Általában  $N \neq M$ , mivel bizonyos modellpontokat elnyelhetek, és a jellemződetektor kihagyhat egyes jellemzőket (vagy hamisakat határozhat meg a zaj miatt). Ezt egy háromdimenziós  $m_i$  modellpontra és a neki megfelelő  $p_i$  képpontra a következőképpen fejezhetjük ki:

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) == Q(m_i)$$

Itt az  $\mathbf{R}$  a rotációs mátrix,  $\mathbf{t}$  az eltolás, és  $\Pi$  jelöli a perspektivikus vetítést vagy annak egy közelítését, mint például a skálázott ortografikus vetítést. A végső eredmény a  $Q$  transzformáció, ami az  $m_i$  modellpontot megfelelteti a  $p_i$  képpontnak. Bár kezdetben nem ismerjük a  $Q$  transzformációt, (merev objektumokra) tudjuk, hogy  $Q$ -nak minden modellpontra *azonosnak* kell lennie.

Meghatározzuk  $Q$ -t a modellpontjainak és a pontoknak megfelelő kétdimenziós projekciók adott háromdimenziós koordinátái alapján. Intuíciónk a következőt súgja: felírhatunk olyan egyenleteket, amelyek az  $m_i$  modellpontokat és a  $p_i$  képpontokat összekapcsolják. Ezekben az egyenletekben az ismeretlen mennyiségek az  $\mathbf{R}$  forgatási mátrix és a  $\mathbf{t}$  eltolási vektor paramétereinek felelnek meg. Ha van elég egyenletünk, ki kell tudnunk számolni  $Q$ -t. Nem fogjuk bizonyítani itt, hanem csupán állítjuk a következő eredményt:

Ha adott három, nem egy egyenesre eső  $m_1, m_2$  és  $m_3$  modellpont, és ezek skálázott ortogonális projekciója,  $p_1, p_2$  és  $p_3$  a képsíkon, akkor létezik pontosan két transzformáció a háromdimenziós modell koordinátakeretből a kétdimenziós képköordinátakeretbe.

Ezek a transzformációk a képsíkra vonatkozó tükrözésen keresztül kapcsolatban állnak egymással, és egy egyszerű zártalakú megoldással meghatározhatók. Ha meg tudnánk határozni a kép három jellemzőjének megfelelő három modelljellemzőt, akkor kiszá-

```

function ILLESZTÉS(kép, modell) returns egy megoldás vagy kudarc
  inputs: kép, képjellemző pontok listája
           modell, modelljellemző pontok listája

  for each p1, p2, p3 in TRIPLETS(kép) do
    for each m1, m2, m3 in TRIPLETS(modell) do
      Q  $\leftarrow$  TRANSZFORMÁCIÓT-KERES(p1, p2, p3, m1, m2, m3)
      if a Q szerinti projekció megmagyarázza a képen then
        return Q
  return kudarc

```

24.21. ábra. Az illesztésalgoritmus informális leírása

molhatnánk *Q*-t, az objektum helyzetét. Az előző alfejezetben tárgyaltunk egy alakzatkontextuson alapuló megfeleltetést. Ha az objektumnak vannak jól meghatározott sarkai vagy más jellegzetes pontjai, akkor egy még egyszerűbb technika alkalmazása válik lehetségesre. Az ötlet lényege a generálás és tesztelés. Egy kezdeti megfeleltetést kell találnunk egy képhármas és egy modellhármas között, majd a TRANSZFORMÁCIÓTKERES eljárást használjuk *Q* egy hipotézisének előállítására. Ha a tippelt megfelelés helyes volt, akkor *Q* helyes lesz, és ha a többi modellpontra alkalmazzuk, a képpontok predikcióját eredményezi. Ha a tippelt megfelelés helytelen volt, akkor *Q* helytelen lesz, és ha a maradék modellpontokra alkalmazzuk, nem állítja elő a képpontokat.

Ez a 24.21. ábrán látható ILLESZTÉS algoritmus alapja. Az algoritmus megtalálja egy adott modell elhelyezkedését, vagy hibával tér vissza. Az algoritmus időbeli komplexitása a legrosszabb esetben arányos a modell- és képponthármasok kombinációjának számával, vagyis  $\binom{N}{3} \binom{M}{3}$ -mal, megszorozva minden egyes kombináció ellenőrzésének költségével. Az ellenőrzés költsége  $M \log N$ , mivel az *M* modellpont mindenekre elő kell állítanunk a képbeli pozíciót, és meg kell találnunk a legközelebbi képponttól mért távolságot, ami egy  $\log N$  művelet, ha a kép pontjai egy megfelelő adatstruktúrában helyezkednek el. Így legrosszabb esetben az algoritmus időbeli komplexitása  $O(M^4 N^3 \log N)$ .



24.22. ábra. (a) A tűzgép fényképén talált sarkok. (b) Az eredeti képre ráhelyezett hipotetikus rekonstrukció (Clark Olson hozzájárulásával).

ahol  $M$  és  $N$  a modell és a kép pontjainak száma. Az elhelyezkedésklaszterezésen alapuló technikák randomizálással lecsökkentik a komplexitást  $O(MN^3)$ -re. Ezen algoritmus alkalmazásának eredményei a tűzögépre a 24.22. ábrán láthatók.

## 24.6. NAVIGÁLÁS ÉS MANIPULÁLÁS A LÁTÁS SEGÍTSÉGÉVEL

A látás egyik legfontosabb alkalmazása az, hogy információt szolgáltat az objektumok manipulálásához – hogy felszedhessük, megfoghassuk, elforgathassuk stb. azokat –, valamint hogy akadályok kikerülésével navigálhassunk. Az a képesség, hogy a látást ezen célokra használják, a legprimitívebb állati látórendszerben is jelen van. Sok esetben a látórendszer minimális abban az értelemben, hogy a rendelkezésre álló látómezőből csak annyi információt nyer ki, ami az állat viselkedésének a tájékoztatására szükséges. Igen valószínű, hogy a modern látórendszerek a korai, primitív organizmusokból fejlődtek ki, amelyek egy fényérzékeny foltot használtak az egyik végükön, és így tudták magukat a fény felé vagy azzal ellentétes irányba orientálni. A 24.4. alfejezetben láttuk, hogy a legek egy nagyon egyszerű optikai folyamra alapozó látást használnak, hogy le tudjanak szállni a falon. A *What the Frog's Eye Tells the Frog's Brain* c. klasszikus tanulmány (Lettvin és társai, 1959) megfigyeli, hogy egy béka „éhen fog halni, ha olyan étellel van körülvéve, amely nem mozog. Csak a méret és a mozgás alapján állapítja meg, hogy mi élelem.”

Az „organizmusokban” használt számítógépes látórendszereket robotoknak nevezzük. Tekintsünk egy speciális robotfajtát: egy automata járművezetőt egy autópályán (lásd 24.23. ábra). Először elemezzük a feladatot; azután meghatározzuk a látási algoritmust,

**24.23. ábra.** Az út képe egy autóban lévő kamerával lefényképezve. A vízszintes fehér csíkok jelzik azokat a keresési ablakokat, amiken belül a vezérlő az útvonal jeleket keresi. A gyenge képmínőség nem szokatlan az alacsony felbontású fekete-fehér videók esetében.

amely a feladatok helyes végrehajtásához szükséges. A vezető előtt álló feladatok között találjuk az alábbiakat:

1. Oldalsó irányítás – biztosítani, hogy a kocsi biztonságosan a közlekedési sávon belül maradjon, vagy szükség esetén zökkenőmentesen váltson sávot.
2. Hosszanti irányítás – biztosítani, hogy a kocsi előtt biztonságos távolság legyen.
3. Akadályelhárítás – a szomszédos közlekedési sávokban haladó kocsik figyelése és felkészülés az elkerülő manőverezésre, ha azok valamelyike sávot szeretne váltani.

A vezető feladata, hogy generálja ezen feladatok legjobb teljesítéshoz alkalmas kormányzási, gyorsítási vagy fékezési cselekvéseket.

Az oldalsó irányításhoz szükséges a gépkosci pozíciójának és irányítottságának a sávhoz képesti reprezentálása. A 24.23. ábrán látható kép esetén éldetektáló algoritmusokkal azonosítjuk a sávhatárokat jelző határoló vonalakkhoz tartozó éleket. Ezt követően ezekre az élekre sima görbüket illeszthetünk. Ezen görbék paraméterei információt hordoznak a gépkosci oldalsó elhelyezkedésére nézve, az autó haladási irányára a sávon belül és a sáv íveltségére. Ez az információ – a gépkosci dinamikájával együtt – képezi minden, amire a kormányzást irányító rendszernek szüksége van. Jegyezzük meg azt is, hogy mivel képkockáról képkockára a sáv képi vetületének pozíciója csupán csekély mértékben változik, tudjuk, hogy a képen a sávhatár jelző vonalakat *hol* keressük – csak azokon a területeken kell keresnünk, amelyek párhuzamos fehér vonalakkal vannak megjelölve.

A hosszanti irányításhoz az előtünk haladó gépkocsik távolságára van szükség. Erre a kétkamerás sztereolátás vagy optikai folyam segítségével tehetünk szert. Mindkét megközelítést lényegesen egyszerűsíthetjük, ha kihasználjuk a sík felületen történő haladásból adódó kényszereket. Ilyen technikák felhasználásával a vizuálisan irányított autók nagy sebességgel hosszú ideig tudnak közlekedni.

A gépkosci-vezetési példa egy szempontot igencsak világossá tesz: *egy konkrét feladat esetén a képen elvben található összes információ kinyerése nem szükséges*. Nem szükséges a gépkocsik pontos alakjának a meghatározása, az úttal szomszédos füves területekre vonatkozó az-alak-a-textúrából probléma megoldása stb. A feladat információs szükséglete csak egy bizonyos típusú információra vonatkozik, és lényeges számítási sebességet és robusztusságot lehet biztosítani, ha csak erre az információra összpontosítunk, és a problémából adódó kényszereket teljes egészében kihasználjuk. Az előbbi részekben bemutatott általános megközelítések célja az volt, hogy egy általános alapelméletet adhassunk, amit majd a konkrét feladat szükségleteihez lehet specializálni.

## 24.7. ÖSSZEFoglalás

Bár az ember számára az érzékelés könnyed cselekvésnek tűnik, igen nagy mennyiségi komplex számítást igényel. A látás célja információ kiemelése olyan feladatokhoz, mint a manipulálás, a navigálás és az objektumfelismerés.

- A képalkotás (**image formation**) folyamata, a geometriai és fizikai aspektusokat tekintve, jól feltárt terület. Ha adott egy háromdimenziós jelenet leírása, ennek valamelyen tetszőleges kamerapozíciónak megfelelő képét könnyűszerrel előállíthatjuk

(grafikai probléma). E folyamat megfordítása és a jelenet leírásának a kép alapján történő előállítása nehéz.

- Ahhoz, hogy a manipulálási, navigálási és objektumfelismerési feladatokhoz szükséges vizuális információt kiemelhessük, szükség van valamilyen közbülső reprezentáció megkonstruálására. A képekből az olyan primitív elemeket, mint az élek és a régiók, a **képfeldolgozási algoritmusok (image-processings)** emelik ki.
- Egy képből többféle segítség rejlik, amely lehetővé teszi, hogy a jelenetről háromdimenziós információt kapjunk. Idetartozik a mozgás, a sztereolátás, a textúra, az árnyalás és a kontúrelemzés. Ezek mindegyike a fizikai jelenetre vonatkozó háttér-feltételezéseken alapul, hogy az interpretációt közel egyértelművé tegye.
- A teljes körű objektumfelismerés nagyon nehéz probléma. A fényesség- és a jellemzőalapú megközelítéseket tárgyaltuk. Más lehetőségek is vannak.

## Irodalmi és történeti megjegyzések

Az emberi látás megértésére irányuló szisztematikus próbálkozások az ókorig nyúlnak vissza. Eukleidész (kb. i. e. 300) írt a természetes perspektíváról – az olyan leképezésről, amely a háromdimenziós világ minden  $P$  pontjához az  $OP$  sugár irányát rendeli hozzá, amely sugár a  $P$  pontot a vetítés  $O$  középpontjával köti össze. Eukleidész ismerte a mozgási parallaxis jelenségét is. A perspektivikus vetítés matematikai megértését, egy síkra való vetítés szempontjából, a 15. században az itáliai reneszánsz segítette elő. Az első, a háromdimenziós jelenet helyes geometriai vetítésén alapuló festményt általában Brunelleschinek tulajdonítja (1413). 1435-ben Alberti fogalmazta meg a szabályokat, és ő inspirálta a művészek generációját, akik művészeti teljesítményét máig csodáljuk. A perspektíva tudományának – ahogy akkor neveztek – kidolgozásában kimagaslott Leonardo da Vinci és Albrecht Dürer. Leonardo késő 15. századi leírásait a fény és az árnyék (*chiaroscuro*) játékáról, az árnyék *umbra* és *penumbra* régióról, valamint a légi perspektíváról még mindig érdemes elolvasni (Kemp, 1989).

Bár a perspektívát a görögök is ismerték, furcsamód a felfogásuk a szemnek a látásban betöltött szerepéiről zavaros volt. Arisztotelész azt gondolta, hogy a szem – a mai lézeres távolságmérők mintájára – fény sugarakat bocsát ki. Ezt a téves elképzelést a világnak az arab tudósok közvetítették, többek közt Alhazen a 10. században. Ezután különféle kamerák fejlesztése következett. Ezek egy szobából (latinul a *camera* szóból jelent) álltak, ahova a fényt a falon egy kis nyíláson át engedjük be, hogy az átellenes falon a kinti jelenet képet kivétele. Ezen kamerák mindegyikében a kép természetesen fordított volt, ami vég nélküli zavart okozott. Ha a szem ilyen elvű berendezés lenne, akkor hogyan lehetséges, hogy a képet helyesen láttuk? A probléma a korszak legnagyobb elmélet foglalkoztatta (Leonardót is beleértve). A kérdés megválaszolásához Kepler és Descartes munkássága kellett. Descartes az ablakzsalu nyílásába egy olyan szemet helyezett be, amelynek nem átlátszó felhámrétegét eltávolították. A retinára kiterített papíron így egy fordított képet kapott. Bár a retinás kép valóban fordított, ez nem okoz problémát, mert az agy helyesen interpretálja a képet. Mai szaknyelven azt mondhatnánk, hogy megfelelő módon kezelni kell az adatstruktúrákat.

A látás megértésében a következő lényeges előrelépés a 19. században történt. Helmholtz és Wundt 1. fejezetben említett munkássága a pszichofizikai kísérleteket szigorú

tudományos szintre emelte. Young, Maxwell és Helmholtz munkája megalapította a színes látás három színén alapuló elméletét. Azt a tényt, hogy az emberi szem a mélység érzékelésére is képes, ha a két szem kissé eltérő képet lát, Wheatstone a sztereoszkóp feltalálásával demonstrálta (Wheatstone, 1838). A készüléknek azonnal nagy sikere lett a szalonokban Európa-szerte. A kétkamerás sztereolátás lényegét, azaz hogy a jelenet két, kissé eltérő látószögből készített képe elegendő információt hordoz a jelenet három-dimenziós visszaállításához, a fotogrammetria területén aknázták ki. Megszülettek a kulcsfontosságú matematikai eredmények. Kruppa bebizonyította, hogy ha öt különböző pont két különböző képével rendelkezünk, vissza lehet állítani a két kamera pozíciója közötti elfordulást és eltolást, valamint a jelenet mélységét (egy skálafaktor erejéig) (Kruppa, 1913). Bár a sztereolátás geometriáját jó ideje ismerték, a fotogrammetria megfeleltetési problémáját általában emberek oldották meg, akik az egymásnak megfelelő pontokat igyekezték egymáshoz illeszteni. Az embereknek a megfeleltetési problema megoldásában megmutatkozó csodálatos képességét Julesz Béla demonstrálta a véletlen-pont sztereogram feltalálásával (Julesz, 1971). Ezen problema megoldására mind a számítógépes látás területén, mind a fotogrammetriában sok munkát fordítottak az 1970-es és az 1980-as években.

A 19. század második fele volt az a periódus, amikor az emberi látásra vonatkozó pszichofizikai kutatásokat megalapozták. A 20. század első felében a látás területén a legfontosabb eredményekhez a Max Wertheimer vezette Gestalt pszichológiai iskola jutott. „Az egész több, mint a részek összege” jelszóval azt a nézetet hangsúlyozták, hogy az észlelés elsődleges elemei a teljes formák legyenek, ne pedig az összetevők, mint például az élek.

A második világháború utáni szakaszt megújult aktivitás jellemzette. A legfontosabb J. J. Gibson munkája volt, aki rátámadott az optikai folyam és a textúragradiensek fontosságára olyan környezeti változók becslésénél, mint a dőlés és a lejtés (Gibson, 1950; 1979). Ő újból hangsúlyozta az ingerek fontosságát és gazdagságát. Gibson, Olum és Rosenblatt rátámadtak, hogy az optikai folyammező elegendő információt tartalmaz ahhoz, hogy a megfigyelő meghatározhassa saját mozgását a környezetben (Gibson és társai, 1955). A számítógépes látásban a kutatás ezen területen és a (matematikailag ekvivalens) struktúra-a-mozgásból területen főleg az 1980-as években ment végbe, Koenderink, Van Doorn, Ullman és Longuet-Higgins alapozó eredményeit követve (Koenderink és Van Doorn, 1975; Ullman, 1979; Longuet-Higgins, 1981) adott lendületet a tevékenységnek. A kezdeti aggodalmakat a mozgásból származtatott struktúra stabilitássával kapcsolatban Tomasi és Kanade munkája (Tomasi és Kanade, 1992) csillapította, akik megmutatták, hogy több képkocka felhasználásával és az ebből származó széles alapvonallal az alak egészen pontosan felismerhető.

Chan és társai frják le a légy meghökkentő vizuális apparátusát, amelynek tízszer nagyobb az időbeli vizuális aktivitása, mint az embereké (Chan és társai, 1998). Azaz egy légy egy maximum 300 képkocka/másodperc sebességű filmet is úgy tudna nézni, hogy felismeri az egyes képkockákat.

- Az 1990-es években megjelent koncepcionális újítás a mozgásból származtatott struktúra tanulmányozása volt. Egy ilyen beállításban a kamera kalibrációja nem szükséges, ahogy azt Faugeras megmutatta (Faugeras, 1992). Ez a felfedezés összefügg az objektumfelismerésben alkalmazott geometriai invariánsokkal, ahogy azt Mundy és Zisserman áttekintették (Mundy és Zisserman, 1992), és a mozgásból származtatott rokon struktúrá-

val (Koenderink és Van Doorn, 1991). Az 1990-es években a számítógépek sebességének és tárolási kapacitásának növekedésével a digitális videofelvéttelekből történő mozgási szekvenciaelemzést sok új területen alkalmazták. Valósvilág-beli jelenetek geometriai modelljeinek felépítése a számítógépes grafikai technikákkal történő előállítás céljából különösen népszerűnek bizonyult, amelyet olyan rekonstrukciós algoritmusok vezettek, mint amelyet például Debevec, Taylor és Malik fejlesztettek ki (Debevec, Taylor és Malik, 1996). Hartley és Zisserman, valamint Faugeras és társainak könyvei átfogóan tárgyalják a többszörös nézetek geometriáját (Hartley és Zisserman, 2000; Faugeras és társai, 2001).

A számítógépes látásban az alak textúrából való kinyerésére vonatkozó legfontosabb kezdeti eredményeket Bajcsynak, Liebermannnak és Stevensnek köszönhetjük (Bajcsy és Lieberman, 1976; Stevens, 1981). Míg ez a munka sík felületekre vonatkozott, a görbült felületekre vonatkozóan átfogó elemzéseket Garding, Malik és Rosenholtz végeztek (Garding, 1992; Malik és Rosenholtz, 1994).

A számítógépes látásban az alaknak az árnyékolásból való kinyerését Berthold Horn tanulmányozta (Horn, 1970). E terület fő cikkeit a (Horn és Brooks, 1989) tekinti át. Ez a megközelítés számos egyszerűsítő feltételelél által, amelyekből a legkényesebb a kölcsönös megvilágítás hatásának figyelmen kívüli hagyása volt. A kölcsönös megvilágítás fontosságát kellően értékelték a számítógépes grafika területén, ahol éppen ezen hatás figyelembevételére fejlesztették ki a fénysugárkövetést és a radiozitást. Egy elméleti és gyakorlati kritika a (Forsyth és Zisserman, 1991)-ben található.

Az alaknak a kontúrból való meghatározása területén Huffman és Clowes kezdeti, kulcsfontosságú hozzájárulásai után poliéderes objektumok esetére (Huffman, 1971; Clowes, 1971) Mackworth és Sugihara teljessé tették az elemzést (Mackworth, 1973; Sugihara, 1984). Malik kifejlesztette a tartományonként sima görbült objektumok címkezési sémáját (Malik, 1987). Kirousis és Papadimitriou megmutatták, hogy a triéderes jelenet vonalcímkézése NP-teljes (Kirousis és Papadimitriou, 1988).

Ahhoz, hogy a sima görbült objektumok vetületeiből a vizuális eseményeket megértsük, szükség van a differenciálgeometria és a szingularitáselmélet együttesére. Ezek tanulmányozásához a legjobb irodalom Koenderink *Solid Shape* c. műve (Koenderink, 1990).

A háromdimenziós objektumfelismerés területén Roberts az MIT-n benyújtott diszsertációja adott lényegi eredményeket (Roberts, 1963). Sokan úgy tartják, hogy ez a számítógépes látás területén az első PhD-disszertáció, amely sok kulcsfontosságú ötletet vezetett be, az éldetektálást és a modellalapú illesztést is beleértve. A Canny-féle éldetektálást (Canny, 1986) vezette be. A Roberts által bevezetett illesztés gondolata később Lowe, Huttenlocher és Ullman munkájában újra megjelent (Lowe, 1987; Huttenlocher és Ullman, 1990). A helyzet összerendelési módszer alapján történő becslésének hatékonyságát jelentős mértékben Olson növelte (Olson, 1994). A háromdimenziós objektumfelismerés kutatásának egy másik nagy vonulata, az alakzatok térfogati primitívekre, ún. általánosított hengerekre (**generalized cylinders**) alapuló leírása, amelyet Tom Binford vezetett be, különösen népszerűnek bizonyult (Binford, 1971).

Míg az objektumfelismeréssel foglalkozó számítógépes látás kutatása nagyrészt a háromdimenziós objektumok kétdimenziós képekre történő leképezésével foglalkozott, addig a mintafelismeréssel foglalkozó közösségen létezett egy párhuzamos irányzat, amely a problémát a mintaosztályozás problémájaként látta. Az ezt motiváló példák olyan területekről kerültek ki, mint az optikai karakterfelismerés és a kézzel írt irányító-

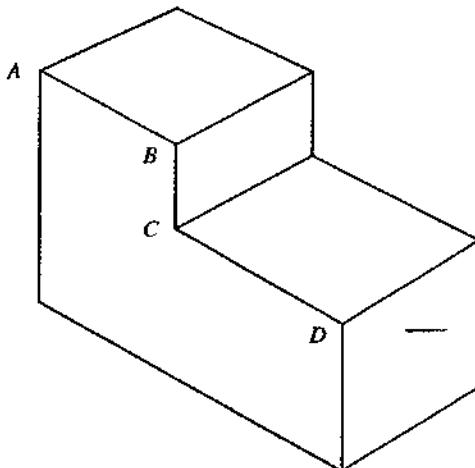
számok felismerése, ahol az elsődleges probléma egy objektumosztály tipikus variációinak jellemvonásait megtanulni, és ezek alapján különbözősztani őket más osztályoktól. A megközelítések összehasonlítását a (LeCun és társai, 1995) tartalmazza. Az objektum-felismerésről szóló további munkák között van (Sirovitch és Kirby, 1987; Viola és Jones, 2002) arcfelismerésről szóló munkája. Az alakkontextus módszerét (Belongie és társai, 2002) írja le. (Dickmanns és Zapp, 1987) demonstrált először nagy sebességgel az autópályán haladó, vizuálisan vezérelt járművel; Pomerleau neurális hálózatok segítségével ért el hasonló teljesítményt (Pomerleau, 1993).

Stephen Palmer munkája, a *Vision Science: Photons to Phenomenology* (Palmer, 1999) a legjobb összefoglaló munka az emberi látás leírására; David Hubel *Eye, Brain and Vision* (Hubel, 1988) és Irvin Rock *Perception* c. műve (Rock, 1984) rövid bevezetőt nyújtanak a neuropszichológia, illetve az észlelés területeibe.

A számítógépes látás területének legjobb összefoglaló jegyzete David Forsyth és Jean Ponce munkája, a *Computer Vision: A Modern Approach*. Sokkal rövidebben tár-gyalja a területet (Nalwa, 1993; Trucco és Verri, 1998). A *Robot Vision* (Horn, 1986) és a *Three-Dimensional Computer Vision* (Faugeras, 1993) régebbi, de még mindig hasznos jegyzetek, saját, speciális témaújkban. David Marr könyve, a *Vision* (Marr, 1982) fontos szerepet töltött be a számítógépes látás (pszichofizika) és a tradicionális biológiai látás (neurobiológia) összekapcsolásában. A számítógépes látás témajában a két fő folyóirat az *IEEE Transactions on Pattern Analysis and Machine Intelligence* és az *International Journal of Computer Vision*. A számítógépes látással foglalkozó konferenciák többek közt az ICCV (*International Conference on Computer Vision*), a CVPR (*Computer Vision and Pattern Recognition*) és az ECCV (*European Conference on Computer Vision*).

## Feladatok

- 24.1.** A sűrű, lombos koronával rendelkező fa árnyékában számos világos foltot lehet látni. Meglepésre minden körkörös. Miért? Hiszen ha úgy vesszük, a levelek közötti rések, amelyeken keresztül átvilágít a nap, nagy eséllyel nem ilyenek.
- 24.1.** Címkezz meg a 24.24. ábrán látható objektumot, azt feltételezve, hogy a kiulsó éleket takaró éleknek címkeztük, és az összes sarokpont triéderes. Tegye meg ezt a visszalépő algoritmussal, amely a sarokpontokat *A*, *B*, *C* és *D* sorrendben fogja megvizsgálni, mindegyik lépésnél úgy megválasztva a címkeket, hogy azok az előbb megcímkezett csatlakozásokkal és élekkel konzisztensek legyenek. Próbálja most a sarokpontokat *B*, *D*, *A* és *C* sorrendben megcímkezni.
- 24.3.** Tekintsen egy végtelen hosszú, *r* sugarú hengert, amelynek tengelye az *y* tengellyel párhuzamos. A henger Lambert-féle felülettel rendelkezik, és a pozitív *z* tengely felől irányítjuk rá a kamerát. Mit fog látni a képen, ha a hengert egy pontszerű fényforrás világítja meg, amely a pozitív *x* tengelyen a végtelenben helyezkedik el. Magyarázza meg a válaszát úgy, hogy az azonos fényességű kontúrok rajzolja fel a vetített képre. Az azonos fényességű kontúrok azonos távolságban helyezkednek-e el?

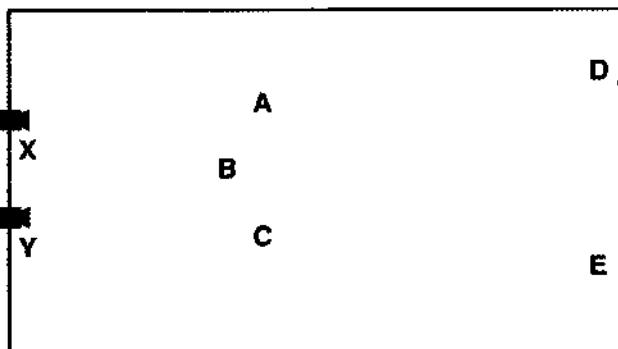


24.24. ábra. A megcímkezendő objektum, amelynek minden sarokpontja triéderes

- 24.4.** A képbeli élek a jelenetesemények sokaságának felelhetnek meg. Tekintsen egy tetszőleges fényképet, amely valódi háromdimenziós jelenetet ábrázol. Azonosítson a képen tíz élt, és kísérélje meg egyenként eldöntení, hogy az él (a) a mélység, (b) a felületi normális, (c) a reflektancia, illetve (d) a megvilágítás szakadásának felel-e meg.
- 24.5.** Mutassa meg, hogy a konvolúció egy adott függvénnyel felcserélhető a differenciálással, azaz, hogy:
- $$(f * g)' = f * g'$$
- 24.6.** Egy terület feltérképezéséhez egy sztereorendszer használatát fontolgatjuk. A rendszer két CCD kamerából fog állni, mindegyik  $512 \times 512$  pixel felbontású egy  $10 \times 10$  cm négyzet alakú érzékelőn. A felhasználandó lencsék fókusztávolsága 16 cm, és a lencsék a végtelenben fixálnak. A bal oldali kép  $(u_1, v_1)$  és a jobb oldali kép  $(u_2, v_2)$  egymáshoz tartozó pontjaira  $v_1 = v_2$ , hiszen a két képsíkon az  $x$  tengelyek az epipoláris vonalakkal párhuzamosak. A két kamera optikai tengelye párhuzamos. A két kamera közötti bázisvonal 1 m.
- Ha a legközelebbi mérendő táv 16 m, mi a tapasztalható legnagyobb diszparitás (képpontban)?
  - Mi a távmérés pixelfelbontásból adódó felbontása 16 m távolságban?
  - Milyen táv tartozik az egy képpontos képeltéréshez?
- 24.7.** Az illesztési algoritmust egy ipari alkalmazásnál szeretnénk használni, ahol lapos alkatrészeket egy szállítószalag visz, a szalag felett pedig egy függőlegesen elhelyezett kamera figyel. Az alkatrész elhelyezkedését három változó határozza meg, egy az elfordulást és kettő az alkatrész kétdimenziós pozícióját. Ez egyszerűsíti a problémát, és a TRANSZFORMÁCIÓT-KERES függvény az egymásnak

megfelelő kép- és modelltulajdonságokból két párt igényel ahhoz, hogy az elhelyezkedést azonosítsa. Határozza meg csak ebben a környezetben az illesztésnek a legrosszabb esetre vett komplexitását.

- 24.8.** (Pietro Perona nyomán.) A 24.25. ábra X és Y pontban két kamerát mutat, amelyek megfigyelnek egy jelenetet. Rajzolja le azt a képet, amit az egyes kamerák látnak (felteheti, hogy minden jelölt pont ugyanabban a vízszintes síkban helyezkedik el). Mi tudható meg a két kép alapján az A, B, C, D és E pontok a kamera alapvonalától mért relatív távolságáról, és milyen alapon?



**24.25. ábra.** Felülnézetű kép egy kétkamerás látórendszeről, amely egy üveget figyel meg, amely mögött egy fal van

- 24.9.** A következő állítások közül melyik igaz és melyik hamis?
- Sztereoképeken az egymásnak megfelelő pontok megtalálása a sztereomélység-megállapítás folyamatának legegyszerűbb feladata.
  - Az alak-mintázatból feladat megoldható egy fénycsíkokat tartalmazó háló jelenetre történő vetítésével.
  - A Huffman-Clowes címkéző rendszer mindenféle poliéderes objektumot tud kezelni.
  - Görbéket tartalmazó objektumok vonalas rajzaiban a vonal címkéje változhat az egyik végétől a másikig.
  - Ugyanazon jelenet sztereonézeteiben minél messzebb van a két kamera egymáshoz képest, annál pontosabban határozható meg a mélység.
  - Egy jelenetben található egyforma hosszúságú vonalak mindenkorábban egyforma hosszúságra vetítődnek a képen.
  - A képen egyenes vonalak szükségszerűen egyenes vonalaknak felelnek meg a jelenetben.

- 24.10.** A 24.23. ábra egy autó nézőpontjából készült egy autópálya kijáratánál. Két autó látható a közvetlenül balra levő sávban. Milyen okok miatt kell a nézőnek azt a következtetést levonnia, hogy az egyik közelebb van, mint a másik?

# 25. ROBOTIKA

Ebben a fejezetben az ágenseket fizikai beavatkozó szervekkel látják el, hogy rosszal-kodhassanak.

## 25.1. BEVEZETÉS

A **robotok** (**robots**) olyan fizikai ágensek, amelyek a fizikai világ megváltoztatásával oldanak meg feladatokat. E célból különböző **beavatkozó szervekkel** (**effectors**) szerelik fel őket, például lábakkal, kerekekkel, karokkal és megfogókkal. A beavatkozók kizártlagos célja, hogy fizikai hatást fejtsenek ki a környezetre.<sup>1</sup> A robotokat **érzékelőkkel** (**sensors**) is felszerelik, hogy érzékelhessék környezetüket. Manapság a robotikában számtalan különféle érzékelőt használnak: kamerákat és ultrahangradarakat a környezet mérésére, giroszkópokat és gyorsulásmérőket a robot saját mozgásának követésére.

A legtöbb mai robot három nagy kategória egyikébe sorolható. A **manipulátorok** (**manipulators**), vagy más néven robotkarok fizikailag a munka helyszínéhez rögzítettek, például egy ipari szelőssoron egy gyárban vagy a Nemzetközi Űrállomáson. A manipulátorok mozgását általában irányítható csuklók sora biztosítja, lehetővé téve, hogy a végbeavatkozó szerv a munkatér bármely pontjára eljuthasson. Az ipari robotok messze leggyakoribb fajtája a manipulátor, világszerte több mint egymillió működik belőlük. Bizonyos mobil robotkarokat kórházakban, műtéteknél használnak. Kevés autogyártó tudna ma már meglenni ipari robotok nélkül, és egyes robotkarok még műalkotások készítésére is képesek.

A második csoportot a **mobil robotok** (**mobile robots**) alkotják. A mobil robotok kerékek, lábak vagy hasonló szerkezetek segítségével mozognak a fizikai környezetben. Használják őket kórházakban ételhordásra, dokkokban árurakodásra és más, hasonló feladatokra. Korábban már említettünk egy példát, a NAVLAB **ember nélküli közúti járművét** (**unmanned land vehicle, ULV**), amely autópályán képes önállóan, sofőr nélkül navigálni. Másfajta mobil robotokat, például az **ember nélküli légi járműveket** (**unmanned air vehicles, UAV**) katonai felderítésre, mezőgazdasági permetezésre és megfigyelésre használnak. Az **autonóm víz alatti járművek** (**autonomous underwater vehicles, AUV**) nagy szerepet játszanak a mélytengeri felfedezésekben, míg a **bolygójárók** (**planetary rovers**), mint például a 25.1. (a) ábrán látható Sojourner, az ūrkutatásban segédkeznek.

<sup>1</sup> A 2. fejezetben még **beavatkozókról** (**actuators**) beszélünk, és nem beavatkozó szervekről (**effectors**). A beavatkozó egy vezérlőnél, amely a beavatkozó szervnek közvetíti az utasítást, míg a beavatkozó szerv maga a fizikai eszköz.



**25.1. ábra.** (a) A NASA Sojourner mobil robotja, amely a Mars felszínét derítette fel 1997 júliusában.  
 (b) A Honda P3 és Asimo elnevezésű humanoid robotjai.

A harmadik típusba tartoznak a hibridek: olyan mobil robotok, amelyekre karokat is szereltek. Ezek közé sorolhatjuk a **humanoid robotokat** (**humanoid robots**), amelyek fizikai felépítése hasonlít az emberéhez. A 25.1. (b) ábrán két ilyen humanoid robot látható, mindenki Japánban készült, a Hondánál. A hibridek a rögzített manipulátoroknál nagyobb távolságokban is képesek beavatkozó szerveiket használni, de általában nehezebben tudják végrehajtani a feladatukat, mert nem rendelkeznek kellő stabilitással és szilárdsággal, amit a lerögzítés biztosít.

A robotika téma köre magában foglalja a protéziseket (emberek részére készített mesterséges végtagok, mesterséges fül vagy szem), az intelligens környezeteket (mint például egy egész ház felszerelve szenzorokkal és beavatkozó szervekkel) és a több-elemű (multibody) rendszereket is, ahol a feladatokat **rengeteg apró együttműködő robot hajja** végre.

A valódi robotoknak általában olyan környezetben kell boldogulniuk, amely csak részlegesen megfigyelhető, sztochasztikus, dinamikus és folytonos. Néhány, de nem mindenik robotkörnyezet szekvenciális és multiágens jellegű is. A részleges megfigyelhetőség és a sztochasztikusság annak az eredménye, hogy egy meglehetősen nagy és komplex világgal kell foglalkozni. A robot nem lát a sarkok mögé, és bizonytalanság van a mozgásparancsok végrehajtásában a hajtások csúszása, a súrlódás stb. miatt. Mi több, a fizikai világ makacsul visszautasítja, hogy valós idejűnél gyorsabb legyen! Szimulációs környezetben egyszerűbb tanulási algoritmusok használatával (mint például a 21. fejezetben leírt Q-tanulás) lehetséges minden össze néhány CPU-munkaóra során több millió próbát végreghajtva tanulni, de valós környezetben éveket vehetnek igénybe ezek a kísérletek. Továbbá, az igazi ütközések valóban fájdalmat okozhatnak, ellentétben a szimuláltakkal. A valódi robotrendszerbe – ahhoz, hogy a robot gyorsan tanulhasson és biztonságosan üzemelhessen – előzetesen információt kell beépíteni magáról a robotról, fizikai környezetéről és az elvégzendő feladatról.

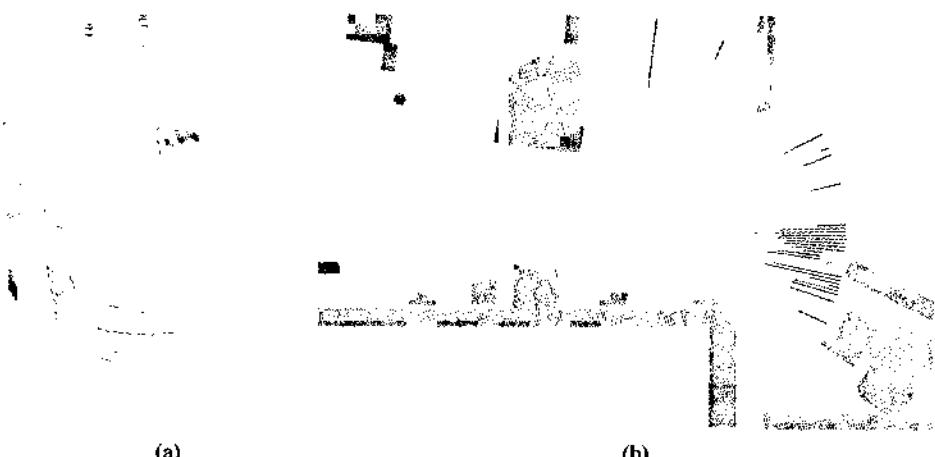
## 25.2. ROBOTHARDVER

Eddig a könyvben adottnak tekintettük az ágensek architektúráját (érzékelőket, beavatkozó szerveket és processzorokat), és csak az ágensek működtető programjára koncentráltunk. A valóságban a robotok sikere legalább annyira múlik a feladathoz illő, megfelelő szenzorok és beavatkozó szervek megválasztásán.

### Érzékelők

A szenzorok jelentik az érzékelési interfész a robotok és környezetük között. A **passzív érzékelők** (passive sensors), mint például a kamerák, ténylegesen pusztta megfigyelői a környezetüknek: olyan jeleket vesznek, amelyeket a robot környezetében lévő más tárgyak generálnak. Az **aktív szenzorok** (active sensors), mint például a hanglokátor, energiát sugároznak környezetükbe, és érzékelik, ha ez az energia visszaverődik. Az aktív szenzorok általában több információt szolgáltatnak, de ugyanakkor többet fogasztanak, és interferencia léphet fel, ha egyszerre többet is használunk. Akár aktív, akár passzív érzékelőkről van szó, három csoportba lehet őket osztani az alapján, hogy távolságot mérneken, teljes képet közvetítenek a környezetről vagy a robot egyes saját tulajdonságait figyelik.

Sok mobil robot egy adott térrészt lefedő **pásztázó távolságmérőt** (range finder) használ, olyan szenzort, amely a közelű tárgyak távolságát méri. Gyakori típus a szonárszenzor, más néven hanglokátor. A szonárszenzorok irányított hanghullámokat bocsátanak ki, amelyek egy része visszaverődik a tárgyakról. A visszaverődés ideje és a hullám intenzitása információval szolgál a közelű tárgyak helyzetéről. Az AUV-kre (autonóm víz alatti jármű) víz alatti hanglokátorokat szerelnek. Szárazföldön a hanglokátorokat – rossz irányfelbontásuk miatt – főleg kis távolságon belüli ütközések elkerülésére használják. A hanglokátorok alternatíváját jelenthetik (elsősorban légi járműveknél) a radaralapú és a lézeres távolságmérők. Egy lézeres pásztázó távolságmérő látható a 25.2. ábrán.



**25.2. ábra.** (a) A mobil robotknál népszerű SICK LMS lézeres távolságmérő. (b) Egy horizontálisan felszerelt távolságmérő által mért visszaverődési kép 2D-s térképre vetítve.

Egyes távolságmérők csak nagyon kicsi vagy nagyon nagy távolságokra használhatók. A kis távolságú szenzorok közé tartoznak a **taktilis**, vagyis tapintásalapú érzékelők (**tactile sensors**), mint például az érzékelőbajusz, lökhárító vagy az érintésérzékeny bőr. A skála másik végén a **globális helymeghatározó rendszer (Global Positioning System, GPS)** áll, amely a műholdakból érkező impulzusok alapján számítja a távolságot. Jelenleg két tucat műhold kering orbitális pályán, és minden egyik két külön frekvencián sugároz jeleket. A GPS-vevőkészülékek a fázistolásból tudják kiszámítani a műholdtól való távolságukat. Több műholdtól érkező jel alapján háromszögeléssel néhány méteres pontossággal meghatározható a Földön elfoglalt tényleges pozíció. A **differenciális GPS (differential GPS)** még egy pontosan ismert helyen lévő földi vevő jelét is használva – ideális esetben – milliméteres pontosságot ér el. Sajnálatos módon a GPS nem használható beltérben vagy víz alatt.

A szenzorok másik fontos osztályát képezik a **képerzékelők (imaging sensors)**. A kamerek képet adnak számunkra a környezetről, valamint – a 24. fejezetben tárgyalt, számítógépes gépi látási módszerek alkalmazásával – modellekkel, illetve tulajdonságokkal írják le a környezetet. A sztereolátás különösen fontos a robotikában, mivel mélységi információt is közvetít. Jövőjük mindenellett elég bizonytalan, mert sikeresen folyik új, aktív távolságleképezési technológiák kidolgozása.

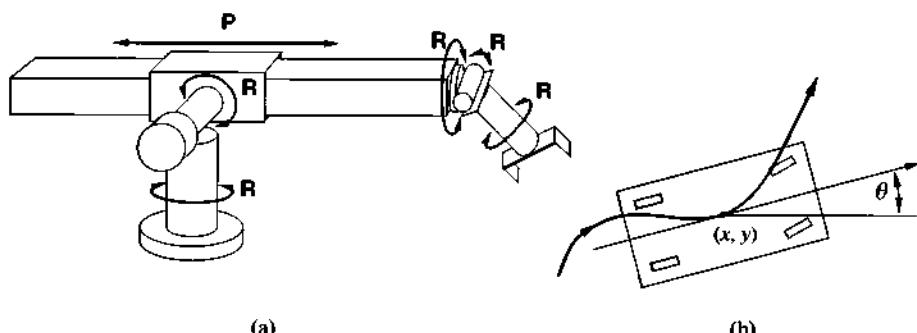
A harmadik fontos osztályt az **önérzékelők (proprioceptive sensors)** alkotják, amelyek a robot saját állapotáról adnak tájékoztatást. Annak érdekében, hogy a robot csuklóinak pontos állásáról információk legyenek, a motorok tengelyét sokszor **szöghelyzet-dekódolókkal (shaft decoders)** látják el, amelyek kis lépésekben követik a tengely elfordulását. A robotkarokon a tengelyre szerelt dekódolók bármikor pontos helyzetinformációt tudnak szolgáltatni. Mobil robotokon a szöghelyzet-dekódolókat a kerék mozgásának figyelésére használják, amely alapján számítható a megtett út. Ezt hívják **odometriának (odometry)**. Sajnálatos módon az odometria – a kerekek csúszása és sodródása miatt – csak kis távolságokban használható. A külső hatások, mint például az áramlások az AUV-knél (autonóm víz alatti jármű) vagy a szél az UAV-k (autonóm légi jármű) esetében tovább növeli a mérés bizonytalanságát. Az **inerciaszenzorok (inertial sensors)**, mint például a giroszkóp, javítanak a pontosságon, de önmagukban még nem küssöblik ki a pozíciómérés hibájának elkerülhetetlen halmozódását.

A robotállapot más fontos jellemzőinek mérésére **erő- és nyomatékérzékelőket (force, torque sensors)** használnak. Ezek nélkülözhetetlenek, ha a robotnak törékeny tárgyal kell dolgoznia, vagy olyannal, amelynek pontos mérete és helyzete nem ismert. Képzeliük csak el, amikor egy közel egytonnás manipulátornak egy villanykörtét kell becsavarlnia. Ha túl nagy erővel fogná meg a körtét, könnyen összeroppanthatná. Az erőérzékelők lehetővé teszik, hogy a robot tudja, milyen nehéz megfogni a körtét, mik a nyomatékérzékelők információt szolgáltatnak, hogy milyen nehéz becsavarni. A jó szenzorok képesek erőt mérimi minden irányban elmozdulási és minden irányban elfordulási irányban.

## Beavatkozó szervek

A robotok beavatkozó szerveik segítségével mozognak és változtatják alakjukat. Ahhoz, hogy megértsük a beavatkozó szervek felépítését, először is általános értelemben kell beszélnünk a mozgás és az alak elvont értelmezéséről, a **szabadságfokok (degree**

**of freedom, DOF)** koncepciójának felhasználásával. Szabadságfoknak számít minden olyan irány, amelyben a robot vagy egyik beavatkozó szerve mozogni képes. Például egy merev, szabadon mozgó robotnak (mint amilyenek az AUV-k) hat szabadságfoka van, három az ( $x$ ,  $y$ ,  $z$ ) térfelületi elhelyezkedés, három pedig a szögelfordulás (orientáció). Ezeket szokták *csavarásnak*, *billentésnek* és *forgatásnak* is hívni (RPY – *roll*, *pitch*, *yaw*). Ez a hat szabadságfok meghatározza a robot kinematikus állapotát,<sup>2</sup> más néven a pozícióját és az orientációját együttesen. A robot dinamikai állapota (*dynamic state*) minden egyes kinematikai paraméter változási sebességét is tartalmazza, így minden kinematikai dimenzióról egy-egy további dimenziót hoz be.



25.3. ábra. (a) A Stanford manipulátor: egy korai robotkar öt rotációs (R) és egy transzlációs (prizmatikus) csuklóval (P), összesen hat szabadságfokkal. (b) Egy elsőkerék-kormányzású nemholonomikus négykereki jármű mozgása.

Nem merev testek esetében magán a roboton belül további szabadságfokok vannak. Például az emberi kar esetében a könyök egy szabadságfokú, mert egy irányba képes elfordulni. A csukló három szabadságfokú, mert tud fel-le és jobbra-balra mozogni, továbbá forogni is. Általában minden egyes robotcsuklónak is egy, két vagy három szabadságfoka van. Hat szabadságfok szükséges ahhoz, hogy például egy kezet a térben tetszőleges pozícióba és irányba (orientációba) hozzunk. A 25.3. (a) ábrán látható robotkarnak pontosan hat szabadsági foka van: öt rotációs csukló (revolute joint), amelyek forgást tesznek lehetővé, és egy transzlációs csukló (prismatic joint), amely kinyúlásra képes. Egy egyszerű kísérlet segítségével könnyedén meggyőződhetünk arról, hogy az emberi kéznek valójában hatnál több szabadságfoka van: kezünket az asztalra rakva még mindig képesek vagyunk elforgatni a könyökünket, anélkül hogy a kezünk konfigurációja megváltozna. Azokat a manipulátorokat, amelyeknek több szabadságfoka van, mint ami minimálisan szükséges ahhoz, hogy végbeavatkozó szervükkel (end effector) a kívánt helyre vigyék, könnyebb vezérelni, mint azokat, amelyek csak a minimálisan szükséges számú szabadságfokkal rendelkeznek.

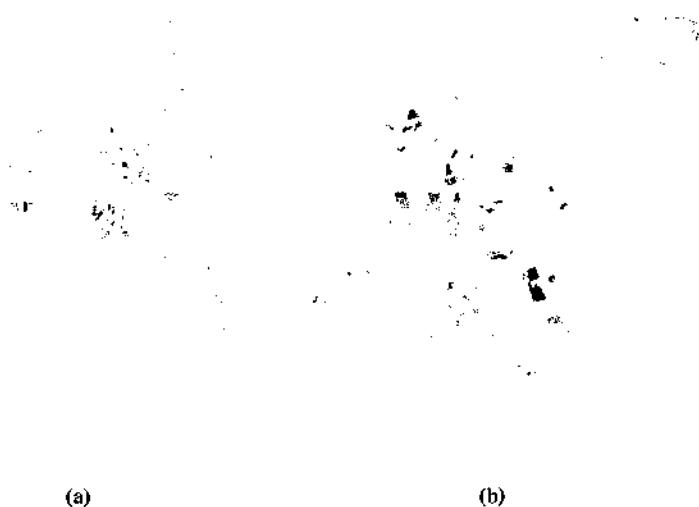
Mobil robotoknál a szabadságfokok száma nem feltétlenül egyezik meg a mozgatott részek számával. Vegyük például egy átlagos autót: tud mozogni előre-hátra és tud fordulni. Ez így két szabadságfok. Ennek ellenére a kocsi kinematikája háromdimenziós: egy szabad lapos felületen könnyedén el tud jutni bármielyen ( $x$ ,  $y$ ) pontba, tetszőleges

<sup>2</sup> A „kinematika” szó a görög „cinema”-ból ered, ami mozgást jelent.

orientációval (lásd 25.3. (b) ábra). Azaz a kocsinak 3 effektív szabadságfoka (**effective DOF**) van, de 2 irányítható szabadságfoka (**controllable DOF**). A robotra azt mondjuk, hogy **nemholonomikus** (**nonholonomic**), ha az effektív szabadságfoka nagyobb, mint az irányítható szabadságfoka, míg **holonomikus** (**holonomic**), ha ez a kettő meggyezik. A holonomikus robotokat könnyebb irányítani. Mennyivel egyszerűbb lenne olyan kocsival parkolni, amely képes oldal irányban is mozogni, nem csak előre-hátra! Viszont a holonomikus robotok jellemző módon mechanikailag sokkal bonyolultabbak. A legtöbb robotkar holonomikus, míg a mobil robotok általában nemholonomikusak.

A mobil robotok számára mozgató mechanizmusok széles tárháza áll rendelkezésre: kerekék, lánctalpak, lábak stb. A **differenciál hajtású** (**differential drive**) robotoknak kétoldalt külön mozgatható kerekeik (vagy lánctalpaik) vannak, ugyanúgy, mint a tankoknak. Ha mindegyik kerék azonos sebességgel mozog, akkor a jármű egyenesen halad előre. Ha ellenkező irányba mozognak, akkor a robot képes egy helyben megfordulni. A másik alternatívát a **szinkrón hajtás** (**synchro drive**) jelenti, amelynél minden kerék el tud fordulni a tengelye mentén. Ez könnyen vezethetne káoszhoz, ha nem lenne az a kényszer, hogy minden kerék mindenkor mindenkor azonos irányba álljon, és azonos sebességgel forogjon. Mind a differenciál, mind a szinkrón hajtás nemholonomikus. Egyes drága robotok holonomikus hajtást használnak, általában három vagy több önállóan irányítható kerékkel.

A kerekekkel ellentétben a lábak még a nagyon nehéz tereppel is megbirkóznak. Ugyanakkor a lábak meglehetősen lassúak sík terepen, és mechanikailag nehezebb megépíteni őket. Robotikai szakemberek próbáltak már különféle konstrukciókat, egy-től akár tucatnyi lábig. Készítettek lábbal rendelkező robotokat sétálásra, futásra, sőt még ugrálásra is. Egy ilyen példa látható a 25.4. (a) ábrán: ez a robot **dinamikusan stabil** (**dynamically stable**), ami azt jelenti, hogy képes talpon maradni, miközben körbe ugrál. **Statikusan stabilnak** (**statically stable**) hívják azokat a robotokat, amelyek úgy képesek állva maradni, hogy nem mozognak a lábaik. A robot statikusan akkor stabil, ha a súlypontja a lábai által kifeszített sokszög fölé esik.



**25.4. ábra.** (a) Marc Raibert egyik lépegető robotja mozgás közben. (b) A Sony AIBO robot focizás közben (copyright 2001, The RoboCup Federation).

Más típusú mobil robotok teljesen eltérő módszereket használnak a mozgáshoz. A légi járműveken általában propellereket vagy turbinákat alkalmaznak. Robot léghajók a meleg áramlatokat használják ki, hogy a levegőben maradjanak. Az autonóm víz alatti járművek gyakran alkalmaznak a tengeralattjárókon használtakhoz hasonló fúvókat.

Szenzorok és beavatkozó szervek önmagukban még nem tesznek ki egy robotot. Egy igazi robotnak energiaforrásra is szüksége van, hogy mozgathassa beavatkozó szerveit. A legnépszerűbb megoldás minden manipulátorok mozgatásához, minden helyváltoztatáshoz, a villanymotor. Ugyanakkor a pneumatikus (sűrített gázzal működő) és a hidraulikus (folyadékkel közvetített nyomással működő) hajtásoknak is megvannak a maguk alkalmazási területei. A legtöbb robot valamiféle digitális kommunikációs eszközzel is rendelkezik, például vezeték nélküli hálózati kapcsolattal. Végezetül szükség van valami vázra is, amelyre fel lehet szerelni az alkatrészeket és az eszközöket, valamint egy forrasztópákára, ha az alkatrészek és eszközök megszünnék működni.

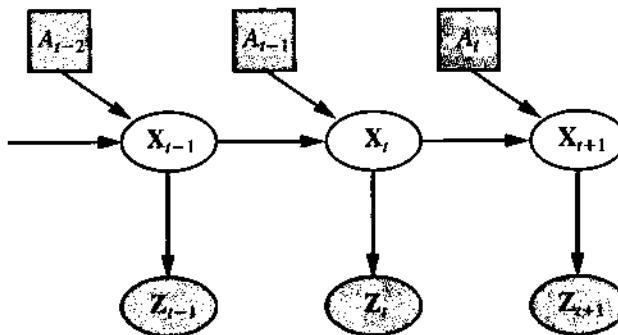
## 25.3. ÉRZÉKELÉS A ROBOTIKÁBAN

Az érzékelés az a folyamat, amelynek során a robot a szenzorairól érkező jeleket környezetének belső reprezentációjára képezi le. Az érzékelés nehéz, mivel általában a szenzorok adatai zajjal terheltek, a környezet csak részlegesen megfigyelhető, nem jósolható viselkedésű és gyakran dinamikus is. Ökolszabály, hogy egy jó belső reprezentáció három tulajdonsága van: elég információt hordoz ahhoz, hogy a robot a megfelelő döntést meghozhassa; strukturáltsága lehetővé teszi, hogy hatékonyan frissíthető legyen; valamint természetes olyan értelemben, hogy a belső állapotváltozók megfelelhetők egy-egy valós fizikai világbeli állapotváltozónak.

A 15. fejezetben megmutattuk, hogy a Kalman-szűrők, a rejtett Markov-modellek és a dinamikus Bayes-hálók alkalmásak egy részlegesen megfigyelhető környezet állapot-átmenet- és érzékelő modelljeinek a reprezentálására. Ismertünk egzakt és közelítő algoritmusokat a belső **hiedelmi állapot (belief state)** – a környezet állapotváltozói felett értelmezett a posteriori valószínűség-eloszlás – frissítésére. Erre számos dinamikus Bayes-hálós modellt mutattunk be a 15. fejezetben. Robotikai problémák esetében a modellhez általában megfigyelt változóként a robot saját korábbi cselekvéseit is hozzáveszünk, mint ahogy az a 17.9. ábra hálózatán látható. A 25.5. ábra ennek a fejezetnek a jelöléseit mutatja:  $\mathbf{X}_t$ , a környezet állapota (beleértve a robotot is) a  $t$  időpillanatban;  $\mathbf{Z}_t$ , a  $t$  időpontbeli megfigyelés (az érzékelésből származó adatok), az  $A_t$ , pedig az érzékelést követően végzett cselekvés.

A szűrési (**filtering**) feladat, vagyis a hiedelmi állapot frissítése alapvetően azonos a 15. fejezetben tárgyaltaikkal. A feladat az új hiedelmi állapot,  $P(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t})$  kiszámítása a  $P(\mathbf{X}_t|\mathbf{z}_{1:t}, a_{1:t-1})$  aktuális hiedelmi állapotból és az új  $\mathbf{z}_{t+1}$  megfigyelésből. Az alapvető különbség az, hogy (1) explicit feltételekkel élünk minden döntést, minden megfigyelést illetően, (2) *folytonos* változókkal kell dolgoznunk *diszkrétek* helyett. Ezért módosítanunk kell a (15.3)-as rekurzív szűrőalgoritmust, integrált használva az összegzés helyett:

$$P(\mathbf{X}_{t+1}|\mathbf{z}_{1:t+1}, a_{1:t}) = \alpha P(\mathbf{z}_{t+1}|\mathbf{X}_{t+1}) \int P(\mathbf{X}_{t+1}|\mathbf{x}_t, a_t) P(\mathbf{x}_t|\mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t \quad (25.1)$$



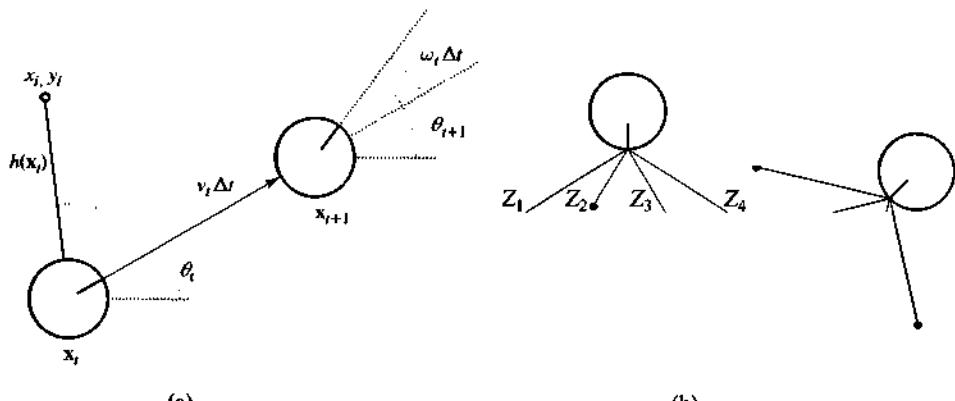
**25.5. ábra.** A robotérzékelés felfogható úgy, mint cselekvések és mérések sorozatának időbeli egymásra hatása, ahogy ezt a dinamikus Bayes-háló is illusztrálja

Az egyenlet azt fejezi ki, hogy az  $X$  állapotváltozó felettesi a posteriori eloszlást a  $t + 1$  időpillanatban rekurzívan számítjuk az egy időlépéssel korábbi megfelelő becslésből. A számításokhoz felhasználjuk a korábbi cselekvést,  $a_{t-t}$ , és az aktuális szenzoros megfigyelést,  $z_{t+1}$ -et. Például ha célunk egy futballozó robot fejlesztése, akkor  $X_{t+1}$  lehet a labda relatív helyzete a robothoz képest. A posteriori  $P(X_t | z_{1:t}, a_{1:t-1})$  minden állapotot felett értelmezett valószínűség-eloszlás, amelyek megörzik mindenkor, amit a korábbi érzékelő mérésekben és irányításokból tudunk. A (25.1) egyenlet megmondja, hogyan becsüljük rekurzívan ezt a pozíciót, folyamatosan felhasználva az újabb szenzoradatokat (például kameráképek) és a robot mozgásparamétereit. A  $P(X_{t+1} | x_t, a_t)$  valószínűséget állapotátmenet-modellnek (transition model) vagy más néven mozgásmodellnek (motion model) hívjuk, míg  $P(z_{t+1} | x_{t+1})$  az érzékelő modell (sensor model).

## Helymeghatározás

A **helymeghatározás** (localization) jellemző példa a robotérzékelésre. A probléma lényege a dolgok pontos helyzetének meghatározása. A helymeghatározás az egyik legfontosabb érzékelési feladat a robotikában, mivel a fizikai környezettel való sikeres kölcsönhatáshoz feltétlenül szükséges. Például a robotkaroknak tudniuk kell, hol van az a tárgy, amivel dolgozni akarnak. A navigáló robotoknak pedig pontosan ismerniük kell saját helyzetüket, hogy eljuthassanak a célpontjukhoz.

A helymeghatározási feladat három, egyre nehezebb problémaként jeletkezik. Ha a tárgy kezdeti pozíciója és iránya (orientációja) ismert, akkor a helymeghatározás tulajdonképpen követési feladatot (tracking) jelent, és ez korlátos bizonytalansággal jellemezhető. Ennél nehezebb a globális helymeghatározás (global localization), amikor is a kezdeti pozíció egyáltalán nem ismert. A globális helymeghatározás követési problémává egyszerűsödik, ha sikerül lokalizálni a kívánt tárgyat, de itt is előfordulhatnak olyan esetek, amikor a robotnak nagyon nagy bizonytalanságokkal kell megbirkóznia. Végezetül lehetünk gonoszak is a robotunkkal, ha elvesszük, „elraboljuk” előle az építési lokalizálni próbált tárgyat. Ezt elrablásos problémának (kidnapping problem)



**25.6. ábra.** (a) Egy mobil robot egyszerűsített kinematikai modellje. A robotot a kör jelképezi, és a bevágás mutatja a haladási irányt. Külön-külön láthatjuk a  $t$ -beli és a  $t + 1$ -beli pozíciót és orientációt a  $v_t \Delta t$  és  $\omega_t \Delta t$  frissítési értékekkel. Szintén fel van tüntetve egy  $t$  időpontban megfigyelt referencia-pont (tereptárgy) az  $(x_r, y_r)$  pontban. (b) A pázsítázó távolságmérés modellje. Egy adott távolságú méréshez  $(z_1, z_2, z_3, z_4)$  tartozó két lehetséges robothelyzet látható. Sokkal valószínűbb, hogy a bal oldali helyzetből származnak a távolságmérések.

hívják, és gyakran tesztelik vele a robot lokalizációs algoritmusának robusztusságát extrém körülmények között.

Azért, hogy egyszerűvé tegyük a dolgot, feltételezzük, hogy a robotunk lassan mozog sík terepen, és pontos térképe van a környezetéről. (Egy ilyen térkép látható a 25.8. ábrán.) Egy mobil robot helyzetét két derékszögű koordinátájával  $(x, y)$  és az irányát jellemző szöggel írjuk le, ahogy ez a 25.6. (a) ábrán is látható. (Mivel nem foglakoztunk a hozzá tartozó sebességekkel, ez inkább kinematikai, mint dinamikus modell.) Ha ezt a három értéket egy vektorba fogjuk össze, akkor bármely állapotot megadható  $\mathbf{X}_t = (x_t, y_t, \theta_t)^\top$  formában.

Kinematikai közelítésben minden cselekvés felbontható két „pillanatnyi” sebességre: egy  $v_t$  (transzlációs) sebességre és egy  $\omega_t$  (rotációs) szögsebességre. Kis  $\Delta t$  időre a robot mozgásának egy durva determinisztikus modellje az alábbi alakban adható meg:

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t}_{a_t}, \underbrace{\omega_t}_{\omega_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos \theta_t \\ v_t \Delta t \sin \theta_t \\ \omega_t \Delta t \end{pmatrix}$$

Az  $\hat{\mathbf{X}}$  jelölés a determinisztikus állapotbecslésre vonatkozik. Természetesen a fizikai robotok valamelyen módon mindenkor megjósolhatatlanok maradnak. Ezt gyakran modellezik egy  $f(\mathbf{X}_t, v_t, \omega_t)$  középértékű,  $\Sigma_x$  kovarianciájú Gauss-eloszlással. (Lásd még az A) függelék matematikai definícióit.)

$$P(\mathbf{X}_{t+1} | \mathbf{X}_t, v_t, \omega_t) = N(\hat{\mathbf{X}}_{t+1}, \Sigma_x)$$

A következő lépésben szükségünk van egy érzéklő modellre. Kétféle érzékelő modellről beszélhetünk. Az első azon alapul, hogy a szenzor **referenciapontoknak** (**landmarks**) nevezett állandó és felismerhető jellemvonásokat érzékel és azonosít a környezetben. minden referenciapontról meghatározza annak távolságát és szögét. Tegyük

fel, hogy a robot az  $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$  állapotban van, és érzékel egy ismert  $(x_i, y_i)^\top$  helyen lévő referenciaPontot. Ha nem vesszük bele az érzékelés zajosságát, egyszerű geometriával megkapható a távolság és a szög (lásd a 25.6. (a) ábra). Megfigyelés alapján a pontos becslés a távolságra és a szögre:

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

A zaj most is torzítja a méréseinket. Egyszerűsítésül Gauss-eloszlású,  $\Sigma_z$  kovarianciájú  $i$  zajt tételezhettünk fel:

$$P(\mathbf{z}_t | \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \Sigma_z)$$

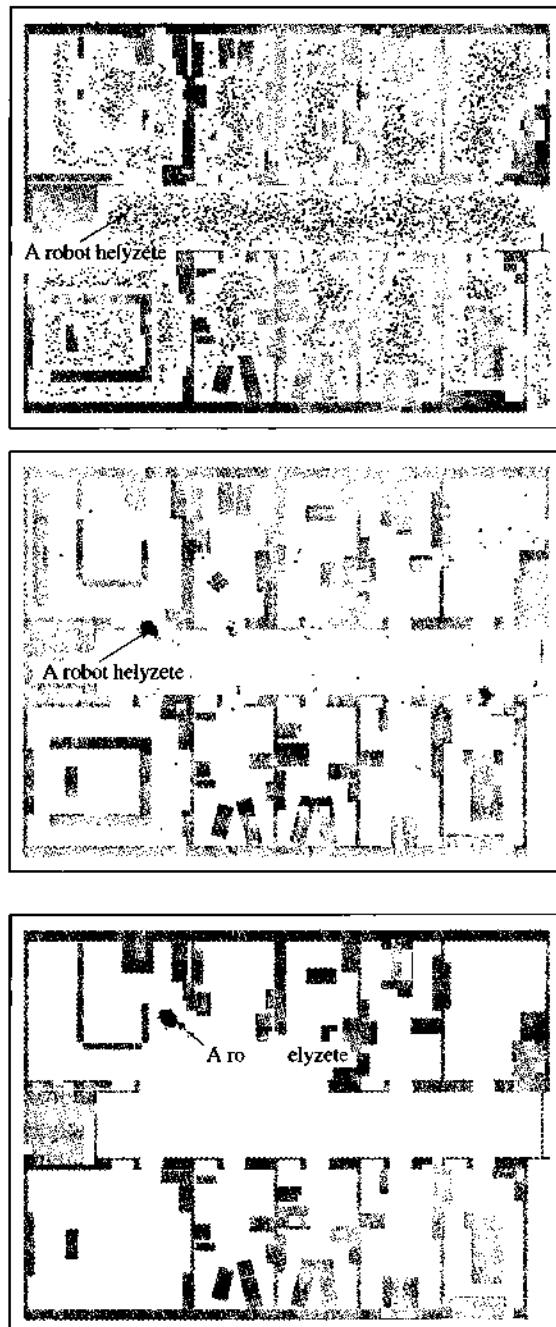
A 25.2. ábrán bemutatott távolságszenzorokhoz kicsit másféle érzékelő modell is megfelelő lehet. Az ilyen érzékelők egy  $\mathbf{z}_t = (z_1, \dots, z_M)^\top$  vektort adnak vissza távolságadatokkal, amelyek a robothoz képest rögzített irányokban lévő tárgyakról hordoznak információt. Adott  $\mathbf{x}_t$  pozícióban legyen  $\hat{z}_j$  a  $j$ -edik sugár irányába lévő legközelebbi akadály pontos távolsága. Csakúgy, mint korábban, most is Gauss-zajt tételezünk fel. Általánosan vehetjük úgy, hogy a különböző sugárirányokbeli hibák függetlenek és azonos eloszlásúak, így:

$$P(\mathbf{z}_t | \mathbf{x}_t) = \alpha \prod_{j=1}^M e^{-(z_j - \hat{z}_j)^2 / 2\sigma^2}$$

A 25.6. (b) ábra egy példát mutat be négyirányú távolságmérő szkennerre, két különböző pozíció esetén. A kettő közül az egyik sokkal nagyobb valószínűséggel adhatta a megfigyelés négy értékét. Összehasonlítva a pásztázó távolságméréses modellt a referenciaPontos modellel látható, hogy az előbbinek az az előnye az utóbbitival szemben, hogy nincs szükség egy referenciaPont azonosítására mielőtt egy tartománypásztázás értelmezhető lenne. Sőt az is előfordulhat – ahogy a 25.6. (b) ábra esetében is –, hogy a robot egy jellegtelén fallal találkozik. Ugyanakkor, ha van egy tisztán látható, azonosítható referenciaPont, akkor abból azonnal megkapható a pontos helyzet.

A 15. fejezetben már találkoztunk a Kalman-szűrővel, amely a hiedelmi állapotot egyetlen többváltozós Gauss-eloszlással reprezentálja, illetve a részecskeSzűrővel, amely a hiedelmi állapotot az állapotokhoz rendelt részecskeegyüttessel reprezentálja. A legtöbb modern lokalizáló algoritmus e két módszer egyikét használja a robot  $P(\mathbf{X}_t | \mathbf{z}_{1:t}, \sigma_{1:t-1})$  hiedelmi állapotának leképezésére.

A részecskeSzűrő lokalizációs algoritmust Monte Carlo lokalizáció (Monte Carlo localization, MCL) hívják. Az MCL alapvetően megegyezik a 15.15. ábrán bemutatott részecskeSzűrő algoritmust, minden összetevőtől eltérően, hogy szükségünk van a megfelelő mozgás és szenzor modellre. A 25.7. ábra bemutat egy változatot, pásztázó távolságméréses modellel. Az algoritmus működését, azt, hogy hogyan azonosítja a robot saját tartózkodási helyét az irodáepületben, a 25.8. ábra szemlélteti. Az első képen még a részecskék egyenletesen oszlanak el, az előzetes információ alapján, jelképezve a teljes bizonytalanságot a robot pozíóját illetően. A második képen már az első mérések alapján klasztereket alkotnak a részecskék, oda tömörülve, ahol valószínűleg tartózkodik a robot. A harmadik esetben pedig már elég információ áll rendelkezésre a mérésekkel, hogy az összes részecské ugyanoda jusson.



**25.8. ábra.** A Monte Carlo lokalizáció, egy részecskeszűrő algoritmus mobil robotok pozíciójának meghatározására. Felül: a kezdeti, teljes bizonytalanság. Középen: nagyjából kétpólusú bizonytalanság, miután a robot elindult a középső (szimmetrikus) folyosón. Alul: egyetlen pont köré szükült bizonytalanság, miután bement az egyik irodába.

```

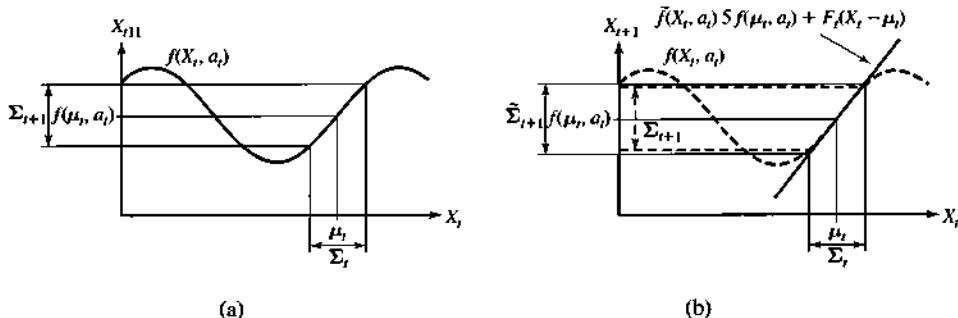
function MONTE-CARLO-HELYZETMEGHATÁROZÁS( $a, z, N, modell, térkép$ ) returns egy mintahalmaz
inputs:  $a$ , a megelőző robotmozgás-parancs
 $z$ , egy pásztázó távolságmérés  $M$  méréssel  $z_1, \dots, z_M$ 
 $N$ , a fentartandó minták száma
 $modell$ , egy valószínűségi környezet modell a helyzetre vonatkozó  $P(X_0)$  priorral
 $P(X_1|X_0, A_0)$  mozgásmodell és pásztázó távolságmérő  $P(Z|\hat{Z})$  zajmodell
 $térkép$ , a környezet 2D térképe
static:  $S$ , egy  $N$  mintából álló vektor, kezdetben a  $P(X_0)$  szerint generálva
local variables:  $W$ , egy  $N$  elemű súlyvektor

for  $i = 1$  to  $N$  do
     $S[i] \leftarrow$  minta a  $P(X_1|X_0 = S[i], A_0 = a)$ -ból
     $W[i] \leftarrow 1$ 
    for  $j = 1$  to  $M$  do
         $\hat{z} \leftarrow$  EGZAKT-TARTOMÁNY( $j, S[i], térkép$ )
         $W[i] \leftarrow W[i] \cdot P(Z = z_j | \hat{Z} = \hat{z})$ 
     $S \leftarrow$  SÚLYOZOTT-MINTA-HELYETTESÍTÉSSEL( $N, S, W$ )
return  $S$ 

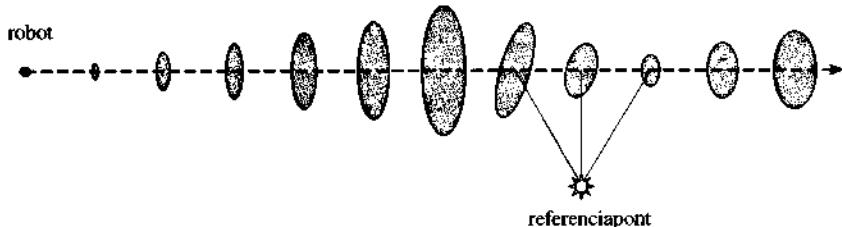
```

**25.7. ábra.** A Monte Carlo lokalizáló algoritmus, független zajjal terhelt pásztázó távolságméréses modellt használva

A Kalman-szűrő a másik széles körben használt lokalizáló algoritmus. A Kalman-szűrő a  $P(X_t|z_{1:t}, a_{1:t-1})$  posteriort egy Gauss-eloszlással reprezentálja. Az eloszlás középpéntékét  $\mu_t$ -vel, kovarienciáját pedig  $\Sigma_t$ -vel jelöljük. A fő probléma a Gauss-hiedelmekkel az, hogy csak lineáris mozgási modell,  $f$  és lineáris mérési modell,  $h$  mellett zártak. Nem-lineáris  $f$  vagy  $h$  esetén a szűrő frissítésének eredménye általában már nem lesz Gauss-eloszlású. Így Kalman-szűrőt használó lokalizációs algoritmusok linearizálják a mozgási és az érzékelő modelleket. A linearizálás egy nemlineáris függvény adott lokális lineáris közelítése. A 25.9. ábra a linearizálás elvét mutatja be egy (egydimenziós) robotmozgási modellen. A bal oldalon fut a mozgás nemlineáris  $f(x_t, a_t)$  modellje (az  $a_t$  vezérlés nem szerepel az ábrán, mert nem játszik szerepet a linearizálásban). A jobb oldalon ezt egy  $f(x_t, a_t)$  lineáris függvénnyel közelítjük. A lineáris függvény a  $\mu_t$  pontban érinti az  $f$ -et, ami a  $t$  időre vonatkozó becslésünk középpéntéke. Ezt a linearizálást (első fokú) Taylor-sorfejtésnek (Taylor expansion) hívják. Az olyan Kalman-szűrőt, amely  $f$ -et és  $h$ -t Taylor-sorfejtéssel linearizálja, kiterjesztett Kalman-szűrőnek (KKSZ; extended Kalman Filter – EKF) hívjuk. A 25.10. ábra egy KKSZ helymeghatározó algoritmust használó robot becsléssorozatát mutatja. Ahogy a robot halad, a helyzetre vonatkozó becslés bizonytalansága egyre nő, ahogy azt a hibaellipszis is mutatja. A hiba csökken, ahogy a robot egy ismert elhelyezkedésű referenciaPont távolsága és szöge alapján javítani tudja saját helyzetére vonatkozó becslését. A hiba végül újra nőni kezd, ahogy a referenciaPont kikerül a robot látóköréből. A KKSZ algoritmus jól működik, ha könnyen azonosítható referenciaPontok vannak. Ellenkező esetben az a posteriori eloszlás multimodális is lehet, mint a 25.8. (b) ábrán. A referenciaPontok azonosításának szükségességét jelentő probléma csak egy példa a 15. fejezet végén tárgyalt adatasszociációs (data association) problémának.



**25.9. ábra.** Egydimenziós, linearizált mozgási modell illusztrálása: (a) Az  $f$  függvény és  $t + 1$  időpontbeli  $\mu_t$  középtéréke és kovarienciája ( $\Sigma_t$  alapján). (b) A linearizált változat az  $f$  érintője a  $\mu_t$  pontban. A  $\mu_t$  középtérük ugyanaz marad, ugyanakkor az előrevetített  $\hat{\Sigma}_{t+1}$  kovariancia már különbözik a  $\Sigma_{t+1|t}$ -től.



**25.10. ábra.** Példa a kiterjesztett Kalman-szűrő használatára lokalizációs problémához. A robot egy egyenes mentén mozog. Ahogy halad előre, a saját helyzetének ismeretéhez tartozó bizonytalanság fokozatosan nő, ahogy azt a hibaellipszisek mutatják. Amikor észrevesz egy ismert pozíciójú referencia-pontot, a bizonytalanság csökken.

## Térképezés

Az előzőek során egyetlen tárgy helymeghatározásának problémáját tárgyalunk meg. A robotikában azonban általában egyszerre több tárgy helyzetét kell meghatározni. A klasszikus példa erre a feladatra a térképezés. Képzeljünk el egy robotot, amelyiknek nincs pontos térképe a környezetéről, magának kell azt elkészítenie. Az ember fantasztikus képességeket fejleszett ki, hogy feltérképezze környezetét, mára már az egész bolygót. Természetesen adódó feladat egy olyan algoritmus kitalálása, amely képessé teszi ugyanerre a robotot is.

Az irodalomban a robottérképezés problémáját gyakran mint **szimultán helymeghatározást és térképezést** (SZLT; simultaneous localization and mapping, SLAM) említik. A robotnak nem csupán egy térképet kell konstruálnia, hanem ezt anélkül kell tennie, hogy ismerné, Ő maga hol tartózkodik. Az SZLT az egyik alapvető robotikai probléma. Azt a változatot fogjuk tekinteni, amikor a környezet állandó. A feladat így is meglehetősen bonyolult; és még nehezebbé válik, ha megengedjük, hogy a környezet változzon, ahogyan a robot mozog benne.

Statisztikai oldalról nézve a térképezés egy Bayes következtetési probléma, csakúgy, mint a lokalizáció. Jelöljük  $M$ -mel a térképet és  $\mathbf{X}_t$ -vel a robot  $t$  pillanatbeli helyzetét, mint ahogy azt korábban is tettük. Ekkor átírhatjuk a (25.1) egyenletet, hogy a posterior tartalmazza a teljes térképet is:

$$P(\mathbf{X}_{t+1}, M | \mathbf{z}_{1:t+1}, a_{1:t}) = \alpha P(\mathbf{z}_{t+1} | \mathbf{X}_{t+1}, M) \int P(\mathbf{X}_{t+1} | \mathbf{x}_t, a_t) P(\mathbf{x}_t, M | \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t$$

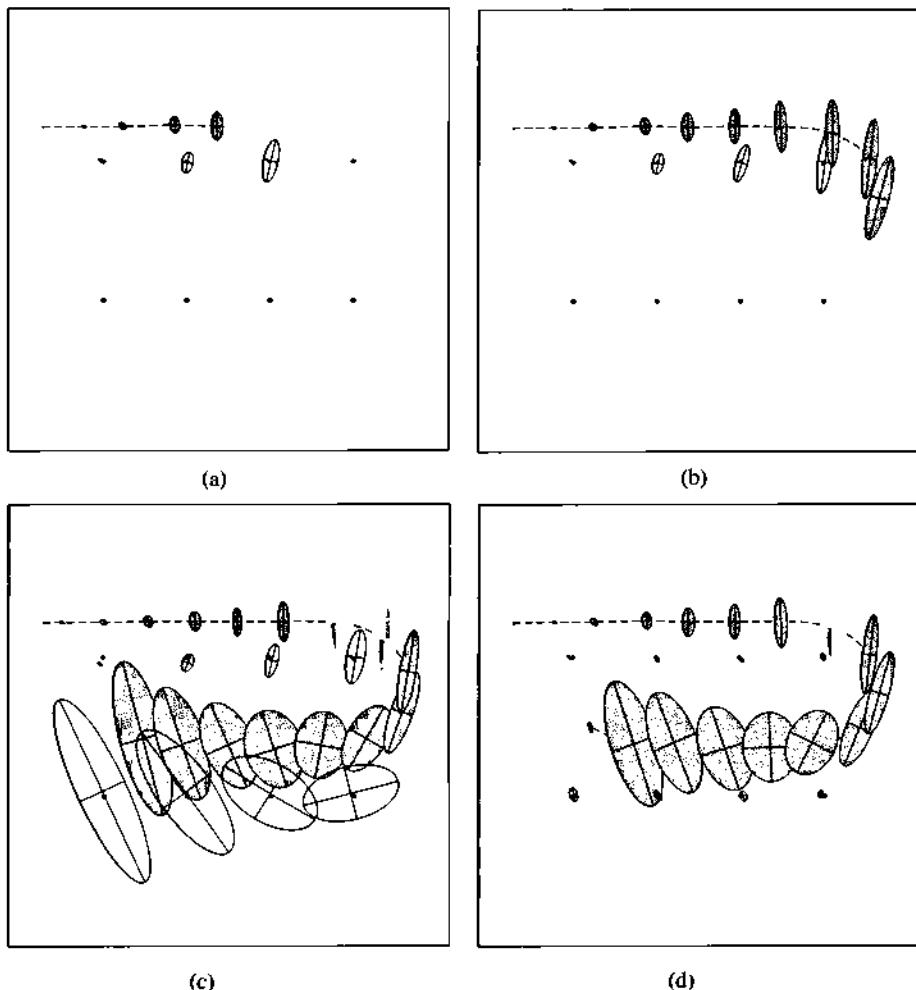
Szerencsére az egyenletből látszik, hogy a feltételes eloszlások, amelyek a cselekvések és mérések figylembenbételéhez kellenek, alapvetően megegyeznek a helymeghatározás problémájánál használtakkal. A fő különbség az, hogy az új állapottér már sokkal több dimenziós, mert tartalmazza az összes robothelyzeten kívül az összes térképet is. Képzeljük csak el, ha egy egész épületet akarunk leképezni fotorealistikusan! Ez esetleg több száz millió numerikus adatot igényelne. minden egyes numerikus adat egy véletlen változó lesz, és hozzájárul, hogy az állapottér elképesztően nagy dimenziós legyen. Ami még tovább bonyolítja a problémát, hogy a robot esetleg előre nem is tudja, hogy milyen nagy a környezete, ezért  $M$  dimenzióját a térképezés során dinamikusan kellene növelni.

Talán a legszélesebb körben használt módszer az SZLT-probléma megoldására a KKSZ. A szűrőhöz általában társítanak még egy referenciaPont-érzékelő modellt is, valamint kitételel, hogy az összes referenciaPont megkülönböztethető legyen. A korábbiakban a posterior becslését egy  $\mu$ , várható értékű,  $\Sigma_t$  kovariációjú Gauss-eloszlással jellemzti. Az SZLT/probléma KKSZ-megközelítésénél a posterior megint csak Gauss-eloszlású lesz, de most a középpérték egy sokkal nagyobb vektor. Nemcsak a robot helyzetét rögzíti, hanem tartalmazza a térkép összes jellemzőjét (vagy referenciaPontját) is. Amennyiben  $n$  ilyen jellemzők van, akkor a vektor  $2n + 3$  dimenziós lesz (két paraméterrel írható le a referenciaPontok helyzete és hárommal a robot póza). Következésképpen a  $\Sigma_t$  mátrix  $(2n + 3)$ -szor  $(2n + 3)$ -as lesz, az alábbi struktúrával:

$$\Sigma_t = \begin{pmatrix} \Sigma_{XX} & \Sigma_{XM} \\ \Sigma_{MX}^T & \Sigma_{MM} \end{pmatrix} \quad (25.2)$$

Ahol is a  $\Sigma_{XX}$  a robot helyzetének a kovariáciája, amit már korábban figyelembe vettünk a helymeghatározásnál.  $\Sigma_{XM}$  egy 3-szor  $2n$ -es mátrix, amely a referenciaPontok és a robotkoordináták közötti korrelációt fejezi ki. Végezetül a  $\Sigma_{MM}$  egy  $2n$ -szer  $2n$ -es mátrix, ami a térképjellemzők kovariációját írja le, beleértve az összes páronkénti korrelációt. Így a KKSZ memóriaigénye kvadratikusan nő  $n$ -nel, a térképen lévő jellemzők számával, és a frissítési idő szintén kvadratikus  $n$ -ben.

Mielőtt elmerülnénk a matematikai részletekben, tanulmányozzuk grafikusan a KKSZ-t. A 25.11. ábra egy robotot mutat, amelynek környezetében nyolc referenciaPont helyezkedik el a két sorban, mindegyikben négy referenciaPonttal. Kezdetben a robotnak nincs információja arról, hogy hol helyezkednek el a referenciaPontok. Tételezzük fel, hogy az összes pont más színű, és a robot ezeket egyértelműen meg tudja különböztetni egymástól! A robot egy ismert pozícióból balra indul el, de fokozatosan elveszti pontos helyének ismeretét. Ezt jelölik a hibaellipszisek a 25.11. (a) ábrán. Az ellipszisek szélessége növekszik, ahogy a robot halad előre. Ahogy a robot mozog, a közelíti referenciaPontok távolságának és szögének érzékelése révén a robot meg tudja becsülni azok helyzetét. Természetesen ezeknek a becsléseknek a bizonytalansága nagyban függ a robot saját helyzetére vonatkozó becslésének bizonytalanságától. A 25.11. (b) és (c) ábrák illusztrálják a robot hiedelmét, ahogy fokozatosan halad előre környezetében.



**25.11. ábra.** A KKSZ alkalmazása a robottérképezés problémájára. A robot pályája a szaggatott vonal, és a beszírozott ellipszisek jelentik helyzetére vonatkozó saját becsléseit. Nyolc pont jelenti az ismeretlen helyzetű referenciaPontokat, míg a helyzetükre vonatkozó becsléseket a fehér ellipszisek mutatják. Az (a), (b) és (c) ábrákon a robot helyzetére vonatkozó becslés bizonytalansága folyamatosan növekszik, ahogyan azoké a referenciaPontoké is, amelyek mellett elmegy. A (d) ábrán a robot újra elérkezik az első referenciaPonthoz, így az összes többi pont helyzetének bizonytalansága is csökken, köszönhetően annak, hogy a becslések korreláltak.

Ezen becslések fontos jellemzője – ami egyáltalán nem magától értetődő az ábrákon –, hogy a becslések egyetlen Gauss-eloszlás jellemzi. A 25. 11. ábrán látható hibaellipszisek csupán ennek az eloszlásnak a vetületei a robot- és referencia koordináták alterébe. Ez a többváltozós Gauss a posteriori eloszlás tartja fenn a korrelációt az összes becslés között. Fontos ez az észrevétel annak megértéséhez, hogy mi történik a 25.11. (d) ábrán. Itt a robot észrevetői már feltérképezett referenciaPontot. Ennek következtében saját pozíójának bizonytalansága drasztikusan csökken, mint ahogyan az öss-

szes többi referenciaPont bizonytalansága is. Ez annak a következménye, hogy a robot és a pontok helyzetének becslése erősen korreláltak a Gauss a posteriori eloszlásban. Bármilyen új információ egy változóról (ebben az esetben a robot helyzetéről) automatikusan csökkenti az összes többi változó bizonytalanságát.

A térképezéshez használatos KKSZ algoritmus hasonlít a korábban a lokalizációhoz használt KKSZ-re. A legfontosabb különbséget a referenciaPont-változók hozzáadása jelenti a posteriorban. A referenciaPontok mozgásmodellje triviális: nem mozognak. Így az  $f$  függvény azokra a változókra az identitásfüggvény. A mérési függvény alapvetően megegyezik a korábban használttal. Az egyetlen különbség a KKSZ frissítési egyenletben az, hogy a  $H$ , Jacobi-mátrixnak nemcsak a robot pozícióját kell figyelembe vennie, hanem a referenciaPontok  $t$ -ben megfigyelt helyzetét is. Az így kapott KKSZ-egyenletek még ijesztőbbek, mint a korábbiak, ezért itt nem is tárgyaljuk őket.

Ugyanakkor van még egy nehézség, amit szép csöndben figyelmen kívül hagytunk eddig: az  $M$  térkép méretét nem tudjuk előre. Ebből kifolyólag a végső becsléshez tartozó  $\mu$ , és  $\Sigma$ , elemeinek száma szintén ismeretlen. Ezeket dinamikusan kell meghatározni, ahogy a robot újabb referenciaPontokat talál. A probléma megoldása elég egyszerű: amikor a robot felfedez egy új referenciaPontot, egyszerűen hozzáad egy új elemet a posteriorhoz. Ha ennek az új elemnek a varianciáját nagyon nagy kezdeti értékűre választjuk, akkor az eredményül kapott posterior ugyanaz, mintha a robot már korábban is tudott volna annak a referenciaPontnak a létezéséről.

## További érzékelési típusok

Nem minden robotérzékelés helymeghatározás és térképezés. A robotok érzékelhetnek hőmérőket, szagokat, hangokat stb. Ezek közül sok valószínűségi alapon becsülhető, mint ahogyan a lokalizációt és a térképezést láttuk. Mindössze olyan feltételest valószínűség-eloszlásokat kell találni, amelyek leírják az állapotváltozók időbeli viselkedését, illetve olyan eloszlásokat, amelyek jellemzik a mérések és az állapotváltozók kapcsolatát.

Azonban nem minden működő robotérzékelés rendszer alapul valószínűségi modellezésen. Valójában nem szükséges, hogy minden belső állapotnak legyen valós fizikai interpretációja, ahogy az a mi példáink esetén volt. Például képzeljünk magunk elé egy járó robotot, amely éppen az egyik lábat próbálja egy akadály felett átemelni. Feltételezzük, hogy a robot olyan szabályt alkalmaz, amely szerint kiindulásként csak kicsit emeli fel a lábat, de egyre nagyobb és nagyobb magasságokkal próbálkozik, ha a lába folyton beleütöközik valamilyen akadályba. Mondhatjuk-e, hogy az emelt láb magassága valamilyen valós fizikai paraméter reprezentációja? Esetleg igen, és akkor az az akadály magasságára, illetve átléphetőségrére vonatkozik. Ugyanakkor mondhatjuk azt is, hogy a lábmagasság a robot vezérlésének egy segédváltozója, mindenfajta fizikai jelentés nélkül. Az ilyen reprezentációk nem ritkák a robotikában, és bizonyos feladatok megoldására kiválóan alkalmasak.

A robotika jelenlegi trendjei egyértelműen egy jól definiált szemantikával rendelkező leírás felé mutatnak. A valószínűség-alapú modellek egyre inkább kiszorítják a többi megoldást olyan komplex problémák esetén, mint amilyen a helymeghatározás és

a térképezés. Bizonyos esetekben azonban a statisztikai technikák túlságosan körülményesek, és néha a gyakorlatban az egyszerűbb megoldások is éppen olyan hatékonyak. A legmegfelelőbb alkalmazandó módszer kiválasztását leginkább a valódi robotokkal történő munka során szerzett tapasztalatok segítik.

## 25.4. MOZGÁSTERVEZÉS

A robotikában a döntések végül is a beavatkozó szervek mozgásában jutnak kifejezésre. A ponttól pontig mozdulás (**point-to-point motion**) esetén a cél a robotnak vagy végbeavatkozó szervnek egy megadott helyre való eljuttatása. Nagyobb kihívást jelent a követő mozdulás (**compliant motion**), amikor a robot úgy mozog, hogy közben végig fizikai kontaktusban marad egy akadálynal. Példa erre, amikor egy robotkar becsavar egy égőt, vagy amikor végig a sztalón egy dobozt.

Azzal kell kezdenünk, hogy keressük egy megfelelő reprezentációt, amely alkalmas a mozgástervezési problémák leírására és megoldására. Kiderül, hogy a konfigurációs tér (**configuration space**) – a robotállapotok tere, amelyet a pozíció, az orientáció és a csuklószögek definiálnak – alkalmasabb a feladatra, mint a 3D tér. A pályatervezés (**path planning**) azt jelenti, hogy meg kell találni az egyik konfigurációtól egy másikig vezető utat. A korábbi fejezetekben már számos úttervezési módszert vázoltunk fel. A robotikában az úttervezés legfőbb jellemzője az, hogy *folytonos* terekkel dolgozik. A szakirodalom számos különböző technikát tart számon, amelyek kifejezetten olyan esetekre alkalmasak, ahol a cél útvonalak keresése sokdimenziós folytonos terekben. Két fő megközelítés létezik: a **celladekompozíció** (**cell decomposition**) és a **szkeletonizáció** (**skeletonisation**). Mindkettő diszkrét gráf keresési feladatra vezeti vissza és egyszerűsíti le a folytonos útkeresési problémát azáltal, hogy kanonikus állapotokat és pályákat ír le a folytonos térben. Ebben a fejezetben végig feltételezzük, hogy a mozgás determinisztikus, és a robot helybeli azonosítása egzakt. A későbbi fejezetek gyengíténi fogják ezeket a feltételezéseket.

### Konfigurációs tér

A robotmozgási feladatok megoldásának első lépéseként a problémához ki kell találni egy megfelelő reprezentációt. Kezdjük egy egyszerű probléma egyszerű ábrázolásával. Vegyük a 25.12. (a) ábrán látható robotkart. Két csuklóval rendelkezik, amelyek egymástól függetlenül mozognak. Ha a csuklókat mozgatjuk, megváltozik a könyök és a megfogó ( $x, y$ ) koordinátája. (A kar z irányban nem képes elmozdulni.) Mindez azt sugallja, hogy a robot konfigurációi leírhatók egy négydimenziós koordináta-rendszerben, ahol:  $(x_{\text{könyök}}, y_{\text{könyök}})$  jelenti a könyök relatív helyzetét a környezethez képest, míg  $(x_{\text{megfogó}}, y_{\text{megfogó}})$  a megfogó helyzetét. Világos, hogy ez a négy koordináta leírja a robot összes lehetséges állapotát. Ezt a leírást **munkatér-reprezentáció** (**workspace representation**) is nevezik, mivel a robot helyzetét ugyanabban a koordináta-rendszerben írjuk le, mint a mozgatni (vagy éppen elkerülni) kívánt tárgyakét. A munkatér-reprezentáció jól használható ütközés elkerülésre, különösen akkor, ha a tárgyakat egyszerű poligonokként modellezzük.

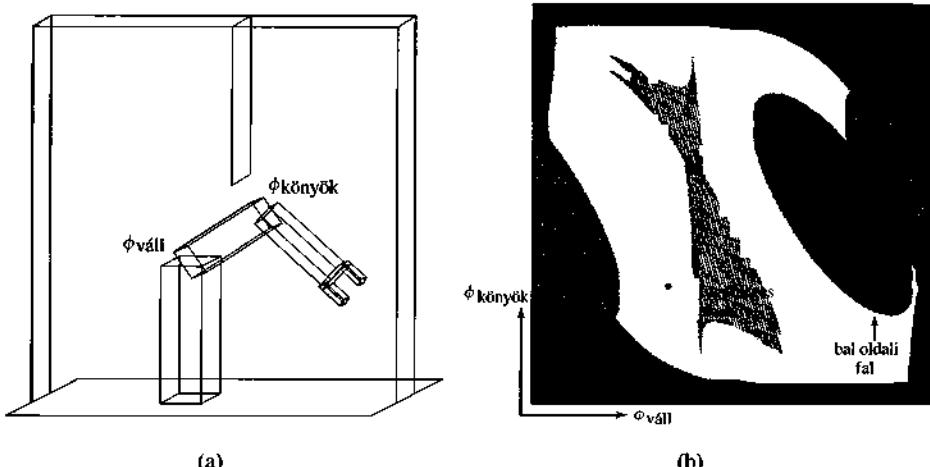
A munkatér-reprezentációval az a probléma, hogy a robot számára nem minden munkatér-koordináta érhető el, még akkor sem, ha nincsenek akadályok. Ez a munkatér elérhető koordinátáira vonatkozó összekapcsolási kényszerek (**linkage constraints**) miatt van. Például a könyök ( $x_{\text{könyök}}, y_{\text{könyök}}$ ) és a megfogó ( $x_{\text{megfogó}}, y_{\text{megfogó}}$ ) pozíciója egymástól mindenkorán adott távolságra van, mert egy merev alkar kapcsolja őket össze. Ha a robotmozgás-tervezés munkatér-koordinátákban történik, akkor a generált útvonalaknak mindenkorán meg kell felelniük ezeknek a kényszereknek. Ez azért különösen nehéz, mert az állapottér folytonos, a kényszerek pedig nemlineárisak.

Könnyebb a dolgunk, ha **konfigurációs térben** (**configuration space**) tervezünk. Ahelyett hogy Descartes-koordinátákkal írnánk le a robot helyzetét a téren, a csuklók konfigurációját vesszük figyelembe. A példában szereplő robotnak két csuklója van, így állapotát a két csukló állásával jellemzhetjük:  $\phi_{\text{váll}}$  jelenti a vállcsukló szögét, míg  $\phi_{\text{könyök}}$  a könyöksuklót. Ha semmilyen akadály sincs a munkatérben, akkor a robot csuklói bármilyen értéket szabadon felvehetnek a konfigurációs téren. Söt pályatervezés során a kiindulási és a célkonfiguráció egyszerűen összeköthető egy egyenessel. A robot ezt az egyenest követve konstans sebességgel mozgatja csuklóit, amíg el nem éri a kijelölt helyet.

Sajnálatos módon azonban a konfigurációs térnek is megvannak a maga problémái. A robot feladatai általában a munkatér koordinátáiban adottak, és nem a konfigurációs téren. Konkrétan például azt akarjuk, hogy a robot végberendezését a munkatérben egy adott pontra mozgassa, söt esetleg még az orientáció is adott. Ez felveti azt a kérdést, hogy hogyan képezzük le a munkatér-koordinátákat a konfigurációs tére. Általában ennek a problémának az inverze egyszerűbb, azaz könnyebben képezhetők le a konfigurációs tér koordinátái a munkatérbe, hiszen a probléma néhány egyértelmű koordináta-transzformációval megoldható. Ezek a transzformációk lineárisak transzlációs, és trigonometrikusak rotációs csuklók esetén. A transzformációk sorozatát **kinematikának** (**kinematics**) hívják. Ezzel a kifejezéssel már találkoztunk korábban, a mobil robotknál.

**Inverz kinematikának** (**inverse kinematics**) hívják egy olyan robot konfigurációjának kiszámítását, ahol a robot beavatkozó szerveinek helyzete a munkatér koordinátáiban adott. Az inverz kinematikai számítások általában nehezek, különösen sok szabadságfokú robotok esetében. Söt a megoldás ritkán egyértelmű. Példarobotunk esetében két különböző konfiguráció is létezik, amikor a megfogó ugyanazt a helyzetet veszi fel a munkatérben, mint ami a 25.12. ábrán látható.

Általában két szabadságfokú (2DOF) robotkar esetén az inverz kinematikai feladat megoldásainak száma nulla és kettő között van, bármilyen bemeneti munkatér-koordinátára. A legtöbb ipari robot esetén végtelen sok megoldás van. Ennek megértéséhez képzeljük el, hogy a példarobotunkhoz hozzáadunk még egy rotációs csuklót, aminek tengelye párhuzamos a meglévő csuklók egyikével. Így a legtöbb konfiguráció esetén szabadon tudjuk mozgatni a közbülső csuklókat, miközben a megfogó pozíciója nem változik (az orientációja azonban igen). Néhány további csuklóval (mennyivel?) megoldható, hogy a kart ugyanigy mozgatva az orientáció se változzon. Már láttunk erre példát, amikor letettük kezünket az asztalra, és mégis tudtuk mozgatni a könyökünket. A kéz pozíciójából adódó kinematikai korlátozások alapján még nem határozható meg egyértelműen az emberi könyök konfigurációja. Más szavakkal, a váll-kar együttes inverz kinematikai problémájára végtelen sok megoldás létezik.

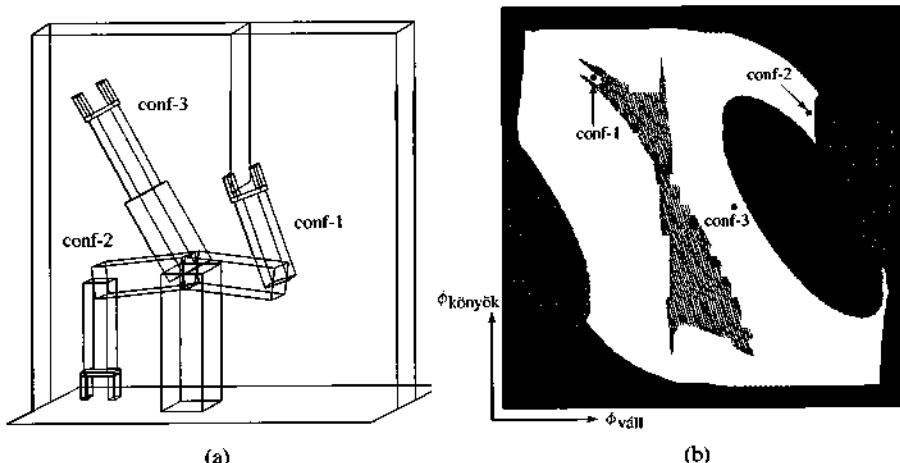


**25.12. ábra.** (a) Egy két szabadságfokú robotkar munkatere. A munkatér egy doboz, amiben egy lapos akadály lóg a mennyezetről. (b) Ugyanennek a robotnak a konfigurációs tere. A fehérrel jelzett terület jelenti azokat a konfigurációkat, ahol a robot nem ütközik össze semmivel. A pont az ábrán a robot bal oldali konfigurációját mutatja.

A másik probléma a konfigurációs térrrel kapcsolatosan akkor merül fel, amikor akadályok is vannak a munkatérben. A 25.12. (a) ábrán látható esetben például számos ilyen akadály van; az egyik pont a robot munkaterének közepébe nyúlik bele. A munkatérben ezek az akadályok egyszerű geometriai formákként jelennék meg, különösen a robotikai tankönyvekben, ahol a poligonális akadályokra fókusznak. De hogyan néznek ki ezek a konfigurációs térben?

Példarobotunk konfigurációs terét mutatja a 25.12. (b) ábra a 25.12. (a) ábrán látható akadályok esetére. A konfigurációs teret két altérre bonthatjuk: azoknak a konfigurációknak a terére, amelyek elérhetők – ez a szabad tér (*free space*), illetve az elérhetetlen konfigurációk által alkotott térrészre, amit foglalt térnek (*occupied space*) hívnak. A 25.12. (b) ábrán a fehér rész jelenti a szabad teret, míg az összes többi a foglalt tér. A különböző árnyalatok jelzik, hogy melyik akadályról is van szó: a szabad teret körbehatároló fekete részek jelentik azokat a konfigurációkat, amelyeknél a robot önmagával ütközik össze. Könnyű belátni, hogy a váll- és könyöksuklik szélén állásai ezt eredményezik. A robot két oldalán lévő ovális részek az asztalnak felelnek meg, amin a robot áll. A harmadik ovális terület a bal oldali fal. Végezetül, a konfigurációs térben a legkülönösebb alakzat a robot mozgásterébe belelőgő, egyszerű függőleges elem. Ennek igen különös alakja van, erősen nemlineáris, és helyenként még konkáv is. Egy kis képzőrővel felismerhetjük a megfogó alakját a bal felső szélén. Szakítsunk egy kis időt ennek a fontos ábrának a tanulmányozására. Az alakzat egyáltalán nem nyilvánvaló! A 25.12. (b) ábrán a pont a robot 25.12. (a) ábrán lévő állapotának megfelelő konfigurációt mutatja. A 25.13. ábrán további három konfigurációt is láthatunk, mind a munkaterben, mind a konfigurációs téren. A „conf-1” állásban a megfogó megközelíti a középső akadályt.

A legtöbb esetben a szabad tér alakja nagyon bonyolult a konfigurációs téren, még akkor is, ha a munkateret egyszerű poligonokkal reprezentáljuk. Éppen ezért a gyakorlatban



25.13. ábra. Három különböző robotkonfiguráció a munkatérben és a konfigurációs térben

inkább *feltérképezik* a konfigurációs teret, semmint pontosan kiszámítják. A térképezés során generálnak egy konfigurációt, és megnézik, hogy az a szabad térben van-e. Ehhez alkalmazzák a robot kinematikáját, és annak alapján már látszik, ha az adott konfiguráció ütközéssel jár a munkatérben.

## Celladekompozíciós módszerek

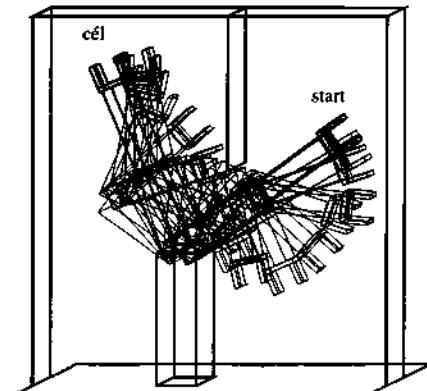
Első megközelítésben **celladekompozíciót** (cell decomposition) használunk pályatervezéshez, ami azt jelenti, hogy a szabad teret véges számú folytonos régióra, cellákra osztjuk fel. Fontos tulajdonsága ezeknek a régióknak, hogy a pályatervezés egyszerű módszerekkel megoldható egy-egy cellán belül (pl. egyenes vonalú mozgással). Ezután a pályakeresés diszkrét gráfkeresési problémává egyszerűsödik, nagyon hasonlóvá ahhoz, amit a 3. fejezetben már bemutattunk.

A legegyszerűbb celladekompozíció egy szabályos osztástávolságú rácsból áll. Ilyen négyzethálós dekompozíciót mutat be a 25.14. (a) ábra, az ehhez a sűrűségű felosztás-hoz tartozó optimális útvonalallal. Különböző árnyalatokat használtunk a szabad tér egyes celláihoz tartozó értékek jelölésére. Ez az érték mutatja például, hogy milyen messze van az adott cella a céltól. (Kiszámításához használhatjuk a 17.4. ábrán ismertetett ÉRTÉKITERÁCIÓ algoritmus egy determinisztikus változatát.) A 25.14. (b) ábrán látható a kar mozgása a munkatérben.

Az ilyen felbontásnak a nagyon könnyű megvalósíthatóság az előnye. Ugyanakkor, használatakor két korlátozás is él. Először is, csak kis dimenziójú munkaterek esetén működik, mert a cellák száma  $d$ -vel, a dimenziók számával exponenciálisan nő. Másodszor, felmerül a kérdés, hogy mit csináljuk azokkal a cellákkal, amelyek vegyesek, vagyis részben a szabad térhez, részben a foglalthoz tartoznak. Egy ilyen cellán keresztül menő megoldás nem feltétlenül működik, mert lehet, hogy az adott irányban nem lehet át-haladni a cellán. Ez az úttervezőt *bizonytalanná* tenné. Ugyanakkor, ha egyáltalán nem



(a)



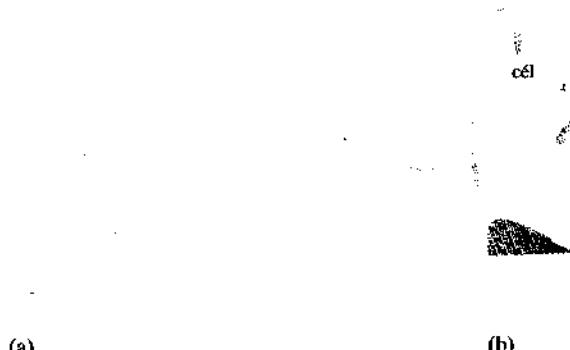
(b)

**25.14. ábra.** (a) A költségfüggvény és az előállított trajektória a konfigurációs térben, diszkrét hálós celladekompozíciós közelítés esetén. (b) Ugyanaz az útvonal a munkatérben ábrázolva. Érdemes megfigyelni, hogy a robot behajtja a könyököt, hogy elkerülje a függőleges akadályt.

engedjük meg, hogy a pálya ilyen „vegyes” cellákon menjen keresztül, tervező algoritmusunk *hiányos* lesz. Előfordulhat, hogy az egyetlen lehetséges út a célohoz egy ilyen cellán megy át – különösen akkor, ha a cella mérete összehasonlítható a térközök és átkelők fizikai méretével.

A fenti problémák elkerülése érdekében a celladekompozíciós módszeren kétféleképpen lehet javítani. Az első, ha megengedjük a vegyes cellák *további osztását* – mondjuk az eredeti méret felére. Ezt rekurzívan folytathatjuk, amíg nem találunk olyan utat, ami csak szabad térben lévő cellákon megy át. (Természetesen a módszer csak akkor működik, ha egy adott celláról egyértelműen el tudjuk dönteni, hogy vegyes-e. Ez viszonylag könnyű, ha a konfigurációs tér határai matematikailag egyszerűen leírhatók.) Akkor teljes a módszerünk, ha létezik korlát a legkeskenyebb átkelőra, amin az útvonal bezárható. Annak ellenére, hogy a konfigurációs téren belül alapvetően a bonyolultabb, trükkösebb részekre fókusztálódik a számítási igény, még mindig nem leszünk képesek sokdimenziós problémák kezelésére, mivel a cellák minden rekurzív újraosztása  $2^d$  kisebb cellát hoz létre. A másik megoldás során azzal tesszük teljessé az algoritmusunkat, hogy ragaszkodunk a szabad tér *egy egzakt celladekompozíciójához* (*exact cell decomposition*). Ez a módszer megengedi, hogy a cellák szabálytalan alakúak legyenek, de kizártlag a szabad tér határán. Az alakzatoknak viszonylag egyszerűeknek kell maradniuk, hogy könnyen meg lehessen határozni egy szabad cella átlóját. Ez a technika bizonyos magasabb szintű geometriai ismereteket igényel, ezért részletesebben nem tárgyaljuk.

Ha megvizsgáljuk azt a pályát, amit a 25.14. ábrán kaptunk, láthatjuk, hogy még további nehézségeket kell leküzdenünk. Először is, vegyük észre, hogy a pálya mentén nagyon éles kanyarok vannak. A robot bármilyen véges sebesességgel mozog, nem tudja ezeket „bevenni”. Másodsor, az út helyenként nagyon közel halad el az akadályok mellett. mindenki, aki már vezetett autót, tudja, hogy egy olyan parkolóhely, ahol csak egy-egy milliméter szabad hely van minden oldalon, valójában nem is parkolóhely.



**25.15. ábra.** (a) A taszító potenciáltér távol tartja a robotot az akadályoktól. (b) A megoldásként kapott pálya, ha egyszerre minimalizáljuk az úthosszt és a potenciálfüggvényt.

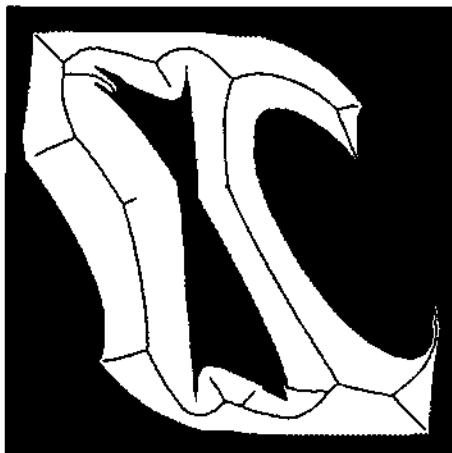
Éppen ezért olyan megoldásokra törekszünk, amelyek robusztusak a kis mozgási hibákra nézve.

Szeretnénk maximalizálni az útvonal távolságát az akadályoktól, ugyanakkor minimalizálni annak hosszát. Mindezt a **potenciáltér** (**potential field**) módszer vezetésével érhetjük el. A potenciáltér az állapotterben definiált függvény, amely nagy értéket vesz fel bármely akadály közelében, és attól távolodva egyre csökken. Ilyen potenciáltér látható a 25.15. (a) ábrán: minél sötétebbre van színezve a konfigurációs tér, annál közelebb van egy akadályhoz. Pályatervezéshez használva a potenciáltér még egy költségfüggvényt jelent az optimalizálás során. Ez érdekes kompromisszumot rejt magában. Egyfelől a robot igyekszik lerövidíteni az út hosszát a célig, másfelől próbál minél távolabb maradni az akadályoktól, azáltal hogy minimalizálja a potenciálfüggvényt. A két szempontot megfelelően súlyozva az eredmény a 25.15. (b) ábrán látható pályához hasonló lesz. Az ábra az egyesített költségfüggvényből értékiteracióval számított értékfüggvényt is mutatja. Világos, hogy így kapott megoldási útvonal hosszabb, viszont biztonságosabb.

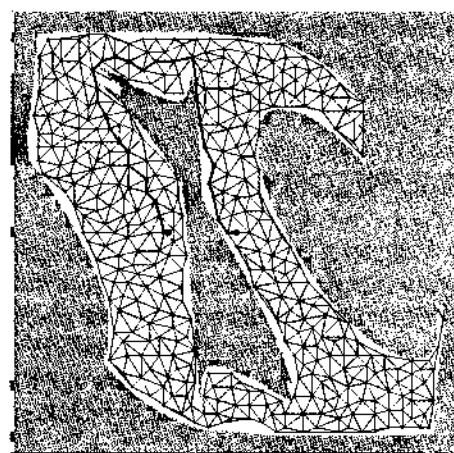
## Szkeletonizációs módszerek

Az útkereső algoritmusok másik fő csoportja a szkeletonizálás (**skeletonization**) elvén alapul. Ezek a módszerek a robot szabad terének egydimenziós leképezésével dolgoznak, amely alapján a tervezési probléma már könnyebben megoldható. Ezt az alacsonyabb dimenziójú reprezentációt hívják a konfigurációs tér vázának (**skeleton**).

A 25.16. ábrán a szkeletonizációra látható egy példa: a szabad tér **Voronoi-gráfja** (**Voronoi graph**) azokból a pontokból áll, amelyek kettő vagy több akadálytól egyenlő távolságra vannak. Ahhoz, hogy a pályatervezéshez a Voronoi-gráfot lehessen használni,



(a)



(b)

**25.16. ábra.** (a) A Voronoi-diagram azon pontok halmaza a konfigurációs téren, amelyek egyenlő távolságra vannak két vagy több akadálytól. (b) A szabad tér 400 véletlenszerűen kiválasztott pontjából felépített valószínűségi úthálózat.

a robotnak először is meg kell változtatnia aktuális konfigurációját úgy, hogy rajta legyen a gráfban. Könnyen belátható, hogy a konfigurációs téren ez minden megoldható egyenes vonalú mozgással. Ezután a robot a gráf mentén halad mindenkor, amíg el nem éri a célkonfigurációhoz legközelebb eső pontot. Ekkor elhagyja a Voronoi-gráfot, és elmegy a célig. A konfigurációs téren ez egyenes vonalú mozgással ismét megvalósítható.

Ilyen módon az eredeti útkeresési feladatot a Voronoi-gráfon lévő útvonal megtalálására redukáltuk. A Voronoi-gráf (néhány különleges esetet leszámítva) egydimenziós, és véges sok olyan pontja van, ahol három vagy több görbe metszi egymást. Így azon a legrövidebb út megtalálása egy diszkrét gráfkeresési probléma, amilyet a 3. és 4. fejezetekben már tárgyalunk. Ha a Voronoi-gráf mentén haladunk, akkor nem a legrövidebb utat kapjuk, de az egyik legbiztonságosabbat. A hátránya ennek a módszernek, hogy nehéz többdimenziós konfigurációs terekre alkalmazni, és sokszor túl nagy kitérőket tesz, főleg ha a szabad tér nagyon tág. Ezenfelül a Voronoi-diagram számítása bonyolult lehet a konfigurációs téren, különösen ahol az akadályok alakja nagyon összetett.

Alternatívát jelent a Voronoi-diagramokkal szemben a **valószínűségi úthálózat** (**probabilistic roadmap**) módszere. Ez a szkeletonizációs technika több lehetséges útvonallal dolgozik, így jobb megoldást jelent nagy, nyitott terek esetén. A valószínűségi úthálózatra látható példa a 25.16. (b) ábrán. A gráfot véletlenszerűen generált, a szabad tére eső, nagyszámú konfiguráció alkotja. Ezek után két csomópontot összekötünk egy éllel, amennyiben „könnyű” eljutni egyikból a másikba. Alapvetően egyenes élekkel dolgozunk, és azok természetesen csak a szabad téren futhatnak. Eredményül egy véletlenszerű gráfot kapunk a robot szabad terében. Ha ehhez még hozzávesszük a kezdeti és a célkonfigurációt, az útkeresés diszkrét gráfkeresési problémává válik. Elméletileg a módszer nem teljes, mivel rosszul felvett ponthalmazzal lehet, hogy soha nem találunk megoldást. A felvett pontok száma és a konfigurációs tér alakja alapján korlátozhatjuk a kudarc valószín-

nőségét. Azt is megtehetjük, hogy főként azon a területen generálunk pontokat, ahol egy előzetes, minden végpontból lefuttatott részleges keresés alapján sejtjük, hogy a helyes út visz majd. Ezekkel a kiegészítésekkel a valószínűségi úthálózat módszere jobban használható más technikáknál magasabb dimenziójú konfigurációs terek esetében is.

## 25.5. BIZONYTALAN MOZGÁSOK TERVEZÉSE

Az eddig tárgyalt pályatervezési algoritmusok közül egyik sem foglalkozott a robotika egyik központi problémájával: a *bizonytalansággal*. A robotikában a bizonytalanság a környezet részleges megfigyelhetőségből és a robot mozgásának sztochasztikus (vagy nem modellezett) összetevőiből ered. Hibát okozhat az is, ha valamelyen közelítések módszert, például részecskeszűrést használunk. Ilyen esetekben a robotnak nincs pontos információja a helyzetéről, még akkor sem, ha a környezet sztochasztikus vonásait tökéletesen modelleztük.

A legtöbb mai robot a döntéshozatalhoz determinisztikus algoritmusokat használ, mint például az eddig tárgyalt útkereső algoritmusokat. Gyakori megoldás, hogy a helymeghatározó algoritmus által előállított állapoteloszlásból a **legvalószínűbb állapotot** (most likely state) választják ki. Ennek a módszernek az előnye a számítások során mutatókozik meg. A pályatervezés már a konfigurációs térben is kihívás, és még nehezebb lenne, ha az állapotok teljes valószínűség-eloszlásával kellene dolgoznunk. Csak akkor lehetjük meg, hogy nem veszünk tudomást a bizonytalansági tényezőről, ha az kellően kicsi.

Sajnálatos módon azonban nem hagyhatjuk minden figyelmen kívül a bizonytalanságot. Egyes problémák esetén egyszerűen túlságosan nagy! Hogyan is használhatnánk determinisztikus pályatervezést egy mobil robot irányításához, ha fogalma sincs a robotnak arról, hogy hol van? Általában, ha a robot valódi állapota nem egyezik meg a maximális valószínűségi szabály által számítottal, akkor az irányító algoritmus nem lesz optimális. Az eltérés nagyságától függően ez egy sor nem kívánatos eseményt idézhet elő, például ütközést más tárgyakkal.

A robotikában számos technikát dolgoztak ki a bizonytalanság kezelésére. Néhányuk a 17. fejezetben tárgyalt bizonytalansági döntéshozó algoritmusokból ered. Ha a robotnak csak az állapotátmenetek kapcsán kell bizonytalansággal számolnia, máskülönben állapota teljesen megfigyelhető, akkor a problémát legjobban a **Markov döntési folyamat** (MDF) lehet modellezni. Az MDF által adott megoldás optimális **eljárásmód** (policy), amely minden lehetséges állapotban megmondja a robotnak, hogy mit csináljon. Ezzel a módszerrel képes bármiféle mozgási hiba kezelésére, ahol a determinisztikus úttervező egyetlen megoldása sokkal kevésbé lenne robusztus. Robotikában a stratégiákat általában **navigációs függvényeknek** (navigation functions) hívják. A 25.14. (a) ábrán bemutatott értékfüggvény könnyen átalakítható ilyen navigációs függvényé, ha egyszerűen követjük a gradienst.

Csakúgy, mint a 17. fejezetben, a részleges megfigyelhetőség itt is tovább nehezíti a problémát. Az így keletkező robotirányítási problémát a **részlegesen megfigyelhető MDF-fel (RMMDF)** modellezzük. Ilyen esetekben a robotnak általában van egy belső képe az állapotáról, ahogy azt már a 25.3. alfejezetben megtárgyalunk. Az RMMDF-re a megoldás a robot belső állapotán definiált stratégia. Másképp megfogalmazva, a stratégia bemenete egy teljes valószínűség-eloszlás. Ez a robot számára lehetővé teszi, hogy ne csak

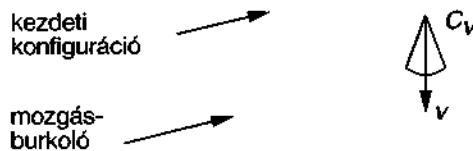
az alapján tudjon döntést hozni, amit tud, hanem azt is felhasználhassa, amit nem tud. Például ha nem ismeri egy kritikus állapotváltozó értékét, akkor ésszerűen **információgyűjtést** (**information gathering**) fog végrehajtani. Az MDF keretein belül erre van lehetőség, mivel az MDF teljes megfigyelhetőséget feltételez. Sajnálatos módon azok a technikák, amelyek az RMMDF-eket pontosan megoldják, nem alkalmazhatók a robotikában, mint-hogy nem ismeretes folytonos idejű változatuk. A diszkretizálás általában olyan hatalmas méterű RMMDf-eket eredményez, amelyek messze meghaladják a mai eszközök feldolgozási kapacitását. Jelen esetben a legtöbb, amit tehetünk, hogy megpróbáljuk a helyzet bizonytalanságát minimumra csökkenteni. A tárgy menti navigálás (*coastal navigation*) heurisztikus módszerével például jól csökkenthető a pozíció ismeretének bizonytalansága, ha a robot végig ismert referencia pontok közelében halad. Ugyanakkor fokozatosan csökkenti más, közeli referencia pontok helyzetének bizonytalanságát is, ami aztán lehetővé teszi a robot számára, hogy újabb területeket térképezzen fel.

## Robusztus módszerek

Valószínűségi módszerek helyett kezelhetjük a bizonytalanságot úgynevezett **robusztus** (**robust**) módszerekkel is. A robusztus módszer csak *korlátozott* mértékű bizonytalanságot enged meg a probléma minden egyes paraméterével szemben, ugyanakkor nem tárít valószínűségeket az egyes értékekhez a megengedett határon belül. A robusztus megoldás a bemeneti értékektől függetlenül megfelelően működik mindenkor, amíg azok nem lépnek át egy határt. A robusztus módszerek egy extrém formája a 12. fejezetben tárgyalta **alkalmazkodó tervezés** (*conformant planning*), ami úgy alakít ki terveket, hogy egyáltalán nincs információja az állapotról.

Itt egy olyan robusztus technikát ismertetünk, amelyet összeszerelési feladataknál finom mozgások tervezésére (FMT; *fine motion planning*) használnak. FMT-re például akkor van szükség, amikor egy robot karai egy nagy, merev tárgyhoz nagyon közel kell mozgatni. A fő probléma az FMT-vel az, hogy a környezet kisméretű, lényeges elemeivel kell dolgozni, nagyon apró mozdulatokkal. Ilyen kis lépték esetén a robot már képtelen pontosan mérni vagy irányítani pozícióját, sőt adott esetben még a környezet alakjáról sincs pontos képe. Feltételezzük, hogy ezek a bizonytalanságok mind korláatosak. Az FMT-problémákra a megoldást legtöbbször feltételes tervek és irányelvek kidolgozása jelenti, amelyek a végrehajtás közben visszacsatolásként használják a szenzorok adatait, és a megadott bizonytalansági korlátokon belül minden esetben megbízhatóan működnek.

Az FMT-k **felügyelt mozgások** (**guarded motion**) sorából tevődnek össze. minden egyes felügyelt mozdulat (1) egy mozgásparancsból és egy (2) leállási feltételből áll, ami a robot szenzorainak adatain alapul. A folyamat igaz értékkel tér vissza, ha sikeresen befejeződött a felügyelt mozgás. A mozgásutasítások jellemzően engedékeny, azaz **önbeálló mozgások** (**compliant motion**), amelyek lehetővé teszik, hogy a robot kitérjen, ha az utasítás végrehajtása ütközést eredményezne. A 25.17. ábrán látható példa egy kétdimenziós konfigurációs teret mutat, középen egy szűk, függőleges lyukkal. Ez egy olyan feladat konfigurációs tere lehetne, amikor egy szögletes csapot kell beilleszteni egy nála alig szélesebb lyukba. A mozgásutasítások konstans sebességeket írnak elő. A leállási feltétel a felülettel történő érintkezés. A vezérlés bizonytalanságának modellezésénél feltételezzük, hogy a robot nem pontosan az utasításban megadott irányba



**25.17. ábra.** Egy kétdimenziós környezet, a sebesség vektor bizonytalanságát ábrázoló kúppal. A vezérlés bizonytalansága miatt az eredeti szándéknak megfelelő  $v$  sebesség valójában bármilyen vektor lehet a  $C_v$  kúpon belül. Ennek eredményeképpen a robot végső konfigurációja bárhol lehet a burkolón (a világosabban árnyalt területen), azaz nem tudjuk, hogy eljutottunk-e a lyukba, vagy sem.

mozdul el, hanem a körül egy  $C_v$  kúpon belül, tetszőlegesen. A 25.17. ábra azt mutatja, mi történne, ha adott sebességgel elindítanánk a műveletet az  $s$  kezdőpontból. A mozgás bizonytalansága miatt a robot a szürke területen belül bárhol megérkezhet a felülethez. Van rá esély, hogy beletalál a lyukba, de nagyobb valószínűséggel valamelyik oldalán áll meg. Mivel a robot nem fogja tudni, hogy a lyuk melyik oldalán van, ezért azt sem fogja tudni, hogy merre mozduljon.

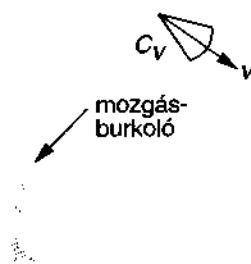
Egy kifinomultabb stratégia látható a 25.18. és a 25.19. ábrán. A 25.18. ábra esetében a robot szándékosan a lyuknak az egyik oldala felé mozog. Az ábrán látható a mozgásparancs és a leállás feltétele: valamely felülettel való érintkezés. A 25.19. ábrán olyan a mozgásparancs, hogy a robot a felület mentén mozdul el, míg bele nem talál a lyukba. Ez önbeálló mozgásvezérlés alkalmazását tételezi fel. Mivel a mozgás iránya (a bizonytalansággal együtt is) jobbra mutat, a robot mindenkor jobbra fog csúszni, valahányszor érintkezésbe kerül a vízszintes felülettel. Amikor eléri a lyukat, lecsúszik annak jobb oldali széle mentén, mivel a függőleges felülethez képest valamennyi lehetséges vektor lefelé mutat. Addig fog mozogni, amíg el nem éri a lyuk alját, mivel ez számára a leállási feltétel. A vezérlés bizonytalansága ellenére a robot valamennyi lehetséges pályájára a lyuk aljával történő érintkezésnél ér véget. mindenkor ez történik, hacsak a felület szabálytalanságai miatt a robot valahol meg nem akad.

Láthatjuk, hogy a finom mozgások tervezése nemtriviális; valójában sokkal nehezebb, mint a pontos mozgások tervezése. Vagy minden egyes mozgáshoz rögzített számos diszkrét értéket választunk, vagy a környezet geometriáját használjuk fel azon irányok megállapításához, amelyek minőségeleg különböző viselkedést eredményeznek. A finommozgás-tervező a konfigurációs tér leírását, a sebességbizonytalansági kúp szögét és azt a specifikációt tekinti inputnak, amely megadja a lehetséges leállásifeltétel-érzékeléseket (ebben az esetben a felülettel való érintkezést). Az eredménynek egy olyan többlepéses feltételes tervnek vagy stratégiának kell lennie, ami garantálja a sikert, ha egáltalán létezik ilyen terv.

Példánkban feltételezzük, hogy a tervezőnek pontos képe van a környezetről, de korlátos hiba is bevezethető a modellbe a következők szerint: amennyiben a hibák paramétereikkel leírhatók, úgy azok a paraméterek szabadságfokok formájában hozzáadhatók a konfigurációs térhoz.



**25.18. ábra.** Az első mozgásparancs és a lehetséges robotmozgások burkolója. A hibától függetlenül tudjuk, hogy a végső konfiguráció a lyuk bal oldalán lesz.

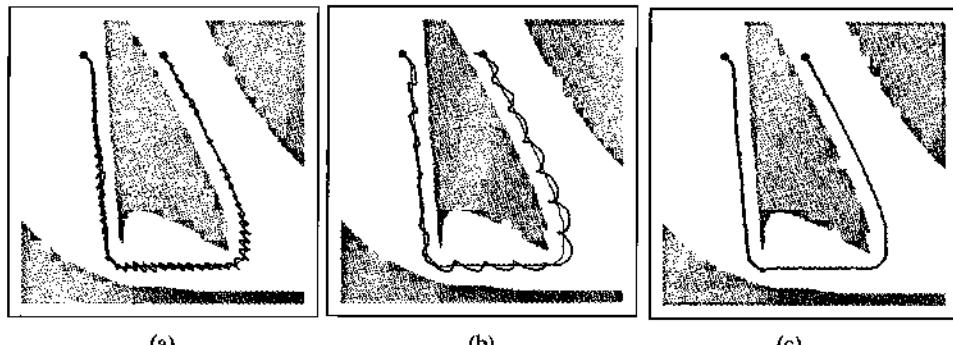


**25.19. ábra.** A második mozgásparancs és a lehetséges mozgások. Még hibák esetén is végül bele fogunk találni a lyukba.

Az utolsó példában, amennyiben a lyuk mélysége és szélessége bizonytalan, úgy ezeket újabb két szabadságfok formájában hozzávehetjük a konfigurációs térehez. Az nem lehetséges, hogy a robotot ezekbe az irányokba elmozdítsuk a konfigurációs térben, és a robot pozícióját sem tudjuk közvetlenül érzékelni. Azonban mindenkor korlátozást figyelembe vehetjük, amikor ezt egy FMT-problémaként írjuk le, megfelelően specifikálva a vezérlés és az érzékelés bizonytalanságait. Ez egy összetett, négydimenziós tervezési problémához vezet, azonban azonos tervezési technikák alkalmazhatók. Megjegyezzük, hogy a 17. fejezetben ismertetett döntéselméleti módszerktől eltérően ez a fajta megközelítés a legrosszabb esetre vonatkozó tervezést eredményezi ahelyett, hogy a terv várható minőségét maximalizálná. A legrosszabb esetre vonatkozó tervezek döntéselméleti értelemben csak akkor optimálisak, ha a végrehajtás közbeni kudarc sokkal költségebb, mint bármely más, a végrehajtás során felmerülő költség.

## 25.6. MOZGÁS

Beszélünk már arról, hogyan *tervezzük* meg a mozgást, de arról nem, hogy hogyan *mozogunk*. Az eddigi terveink – különösen azok, amelyeket determinisztikus pályatervezővel állítottunk elő – feltételezik, hogy a robot egyszerűen képes követni bármilyen megtervezett pályát. A valóságban természetesen nem ez a helyzet. A robotoknak van



**25.20. ábra.** Robotkarvezérlés (a) egységesi erősítésű arányos szabályozóval; (b) 0,1-es erősítési tényezőjű arányos szabályozóval; (c) PD szabályozóval, 0,3-as erősítésű arányos taggal és 0,8-as differenciáló komponenssel. Mindhárom esetben a robotkar a szürkével jelölt pályát próbálta követni.

tehetetlenségük, ezért nem tudnak tetszőleges pályát követni, csak nagyon kis sebesség esetén. A legtöbbször erőt kell kifejtenie a robotnak, nem pedig meghatározott pozícióba eljutnia. Ez a fejezet azokról a módszerekről szól, amelyekkel ki lehet számítani ezeket az erőket.

## Dinamika és vezérlés

A 25.2. alfejezetben már megismerkedtünk a **dinamikus állapot** (**dynamic state**) fogalmaival, amely a robot sebességmagyarájával kiterjeszti a robot kinematikai állapotát. Például egy csukló szögének értéke mellett a dinamika magában foglalja a szög változási sebességet is. A dinamikai állapot reprezentálására szolgáló állapotátmenet-modell tartalmazza az erők erre a változási sebességre gyakorolt hatását. Ezek a modellek általában **differenciálegyenleteket** (**differential equations**) használnak, amelyek egy kinematikai mennyiséget (például egy kinematikai állapotot) viszonyítanak annak időbeli megváltozásához (például a sebességezhez) is. Elvileg a mozgástervezéshoz a kinematikai helyett dinamikai modellt is használhattunk volna. Ez a metodika magasabb szintű teljesítményt eredményezne a robotnál, ha egyáltalán meg tudnánk alkotni a terveket. Azonban a dinamikai állapot sokkal bonyolultabb a kinematikainál. A dimenziók magas száma miatt ezt a módszert csak a legegyszerűbb robotok esetében lehetne sikeresen alkalmazni. Éppen ezért a gyakorlatban leginkább az egyszerűbb kinematikai pályatervező algoritmusokra hagyatkoznak.

A kinematikai pályatervező algoritmusok hiányosságainak kompenzáciásra gyakran használnak különálló **szabályozókat** (**controllers**), hogy pontosan a pályán tartásak a robotot. Szabályozók segítségével a környezetből vett visszacsatolással valós időben generálhatók a vezérlőjelek az irányítási célok eléréséhez. **Referenciaszabályozóról** (**reference controller**) beszélünk, ha az a feladat, hogy a robot egy előre kitűzött pályán maradjon – ezt hívják **referenciapályának** (**reference path**). Azokat a szabályozókat, amelyek globális költségfüggvényeket optimalizálnak, **optimális szabályozóknak** (**optimal controllers**) hívjuk. Az optimális stratégiák az MDF-ek számára valójában optimális szabályozók.

Első ránézésre egyszerűnek tűnik az a feladat, hogy adott pályán tartson a robotot. A gyakorlatban azonban még ennek a látszólag könnyű feladatnak is vannak buktatói. A 25.20. (a) ábrán láthatjuk, hogy mik adódhatnak. A fekete vonal a robot által befutott út, ahogy próbálja a kinematikai pályát követni. Ha eltérés keletkezik – zaj vagy a robot mozgása során fellépő erőkre vonatkozó korlátozások miatt –, a robot ellenkező irányú, az eltéréssel arányos nagyságú erővel igyekszik azt kompenzálni. Elsőre ez a megoldás kézenfekvőnek tűnik, mivel az eltéréseket ellenerőkkel kompenzáljuk, így a robot a pályán marad. Ugyanakkor minden az eredményezi, hogy robotunk erőteljesen vibrálni kezd, ahogy az a 25.20. (a) ábrán is látszik. Ez a vibráció a robotkar természetes tehetetlenségeből következik: amikor a robot visszaér a referencia pálya adott pontjára, túllandul, ami a korábbihoz hasonló, de ellentétes előjelű hibát eredményez. Ahogy az ábrán is látható, ez a fajta túllandulás akár az egész trajektória mentén folytatódhat, és az eredményül kapott mozgás nem éppen ideális. Világos, hogy jobb szabályozásra van szükség.

Mielőtt rátérnénk a megfelelőbb szabályozásra, formálisan írjuk le a túllandulést okozó szabályozót. Azokat a szabályozókat, amelyek ellentétes irányú, arányos erővel reagálnak a megfigyelt hibára **P szabályozónak (P controller)** hívjuk. A P betű *arányosat (proportional)* jelent: a pillanatnyi vezérlés minden arányos a robotkar hibájával. Még formálisabban kifejezve: legyen  $y(t)$  a referencia pálya idő  $(t)$  szerint paraméterezve. A P szabályozó által generált  $a_t$  vezérlőjel:

$$a_t = K_p(y(t) - x_t)$$

ahol  $X_t$  a robotkar  $t$  időpontbeli állapota.  $K_p$  a szabályozó úgynevezett **erősítési tényezője (gain parameter)**, ami azt határozza meg, hogy milyen hevesen reagál a szabályozó az aktuális  $x_t$  állapot és a kívánt  $y(t)$  között lévő eltérésre. A példánkban  $K_p = 1$ . Első pillantásra azt gondolhatja valaki, hogy ha  $K_p$ -t kis értékűre választjuk, akkor az megoldja a problémát. Sajnos nem így van. A 25.20. (b) ábrán látható  $K_p = 0,1$  esetben a trajektória, amely továbbra is oszcillál. A kisebb erősítési tényező csak lelassította az oszcillációt, de egyáltalán nem oldotta meg a problémát. Ha elhanyagoljuk a súrlódást, a P szabályozó tulajdonképpen egy rugó, és a végtelenségig oszcillál a rögzített egyensúlyi pont körül.

Hagyományosan az ilyen problémákat az **irányításelmélet (control theory)** tárgykörébe sorolják. Ennek a tudományagnak egyre nagyobb a jelentősége a mesterséges intelligenciával foglalkozó kutatók számára. Évtizedek kutatásai során számos olyan szabályozót alkottak meg, amelyek lényegesen bonyolultabbak a fent használt egyszerű szabályozónál. Egy referencia szabályozó **stabil (stable)**, ha kis megváltozások csak korláatos hibát eredményeznek a referenciajelhez képest. **Szigorúan stabil (strictly stable)**, ha a robot a kis megváltozások ellenére is vissza tud térti a referencia pályára. Látható, hogy a P szabályozónak stabil, de nem szigorúan stabil, mivel képtelen visszatérni a referencia trajektoriára.

A legegyszerűbb szabályozó, ami eleget tesz a szigorú stabilitás követelményeinek a mi esetünkben a **PD szabályozó (PD controller)**. A P betű továbbra is az arányosat jelöli, míg a D a *differenciálót*. A PD szabályozót az alábbi egyenlet írja le:

$$a_t = K_p(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} \quad (25.3)$$

Ahogy az már a képletből is látszik, a PD szabályozó egy differenciális taggal egészíti ki a P szabályozót, ami az  $a_t$  t egy olyan taggal bővíti, ami arányos az  $y(t) - x_t$  hiba idő szerinti első deriváltjával. Mi a hatása egy ilyen új tagnak? Általában a derivátor csillapítja a szabályozott rendszert. Tegyük fel, hogy egy adott rendszerben időben gyorsan változik az  $(y(t) - x_t)$  hiba, mint ahogyan a fentebb tárgyal esetben is. A differenciális tag ilyenkor az arányos ellenében hat, és összességében csökkenteni fogja a változásra történő reakciót. Ugyanakkor, ha a hiba állandósul, a differenciális tag háttérbe szorul, és az arányos tag fogja meghatározni a szabályozást.

PD szabályozót használva  $K_P = 0,3$  és  $K_D = 0,8$  paraméterekkel a 25.20. (c) ábrán lévő trajektóriát kapjuk. A pálya sokkal simább, és nem látható oszcilláció sem. Ahogy a példa is mutatja, egy differenciál tag stabilizálhat egy szabályozót, amely enélkül nem az.

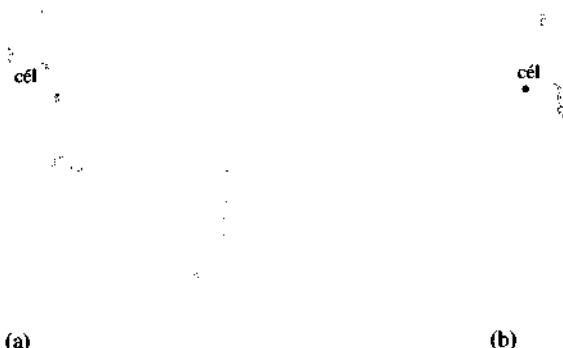
A gyakorlatban a PD szabályozónak is lehetnek hibái. Van úgy, hogy a PD szabályozó akkor sem tudja nullára redukálni a hibát, ha a külső eltérítés megszűnik. Ez a példánkból nem nyilvánvaló, de néha szükség van egy másik visszacsatolásra is az arányoson felül, hogy a hibát meg lehessen szüntetni. A megoldás erre a problémára egy harmadik tag hozzáadása a szabályozáshoz, ami a hiba idő szerinti integrálján alapul:

$$a_t = K_P(y(t) - x_t) + K_I \int (y(t) - x_t) dt + K_D \frac{\partial(y(t) - x_t)}{\partial t} \quad (25.4)$$

$K_I$  egy újabb erősítési tényező. A kifejezés kiszámítja a hiba idő szerinti integrálját. Ennek hatására a referenciajel és az aktuális állapot között hosszan fennálló eltérés megszűnik. Például ha  $x_t$  hosszabb ideig kisebb, mint  $y(t)$ , akkor az integrál addig fog nőni, amíg az  $a_t$  el nem tünteti a hibát. Az integrátor biztosítja, hogy a szabályozónak ne legyen rendszeres hibája, azon az áron, hogy nő az oszcilláció veszélye. **PID szabályozónak (PID controller)** hívjuk azt a szabályozót, amelyik minden tagot tartalmazza. Széles körben használják különböző ipari alkalmazásokhoz és irányítási feladatokhoz.

## Potenciáltér-vezérlés

A potenciáltereket mint újabb költségfüggvényeket vezettük be a robotmozgás-tervezés során, de alkalmasak arra is, hogy közvetlenül generálják a robot mozgását, kiegészítve a pályatervezési fázist. Ahhoz, hogy ezt elérjük, definiálnunk kell egy vonzó erőt, ami a robotot a célkonfiguráció felé vonzza, valamint egy tasztító potenciálteret, amely távol tartja az akadályoktól. Egy ilyen teret már bemutattunk a 25.21. ábrán. Az egyetlen globális minimumhelye a célkonfiguráció, értéke pedig az e céltól való távolság, valamint az akadályok közelségének összege. Az ábrán bemutatott potenciáltér kialakítását nem előzte meg tervezés. Éppen ezért a potenciálterek jól alkalmazhatók valós időjű megoldásokhoz. A 25.21. ábrán két trajektória látható, ahogy egy robot két különböző kezdeti konfigurációból megpróbálja megmászni a potenciáltteret. Nagyon sok alkalmazás esetén a potenciáltteret hatékonyan ki lehet számítani bármilyen adott konfiguráció esetében. Sőt adott robotkonfiguráció esetén a potenciálok kialakulásával a potenciálgradienst is meghatározhatjuk. Ezek a számítások általában különösen hatékonyak, főleg a pályatervező algoritmusokhoz hasonlítva, amelyek mind exponenciálisak a konfigurációs tér dimenziójában (a szabadságfokokban).



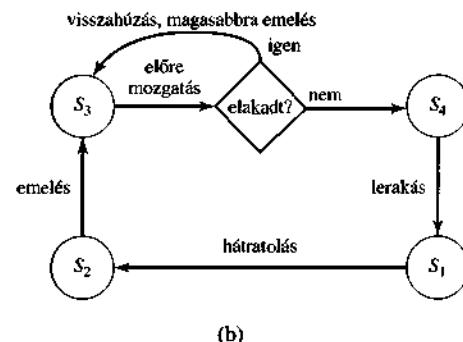
**25.21. ábra.** Potenciáltér-vezérlés. A robot egy potenciáltérben halad, amely az akadályuktól tasztító erőkből és a célkonfigurációhoz vonzóból áll. (a) Sikeres útvonal. (b) Lokális optimum.

Az a tény, hogy a potenciálteres módszer ilyen hatékonyan megtalálja az utat a célohoz a konfigurációs térben még nagy távolságok esetén is, megkérdőjelezni a tervezés szükségességét. Vajon csak szerencsénk volt a példánk esetében, vagy tényleg elegendő a potenciálterek használata? A válasz az, hogy szerencsénk volt. A potenciáltereknek számos olyan lokális minimumhelye lehet, amely csapdába ejti a robotot. Ebben az esetben a robot úgy közelíti meg az akadályt, hogy csak vállcsuklóját mozgatja egészen addig, amíg bele nem ütközik az akadályba a rossz oldalon. A potenciáltér nem ad annyi információt, hogy a robot behajlítsa a könyökét, hogy átférjen az akadály alatt. Más szavakkal, a potenciáltéren alapuló technikák nagyon jók lokális robotirányításra, de globális tervezést igényelnek. A másik hátulütője ennek a módszernek az, hogy az erők, amiket generál, csak az akadályuktól és a céltól való távolságtól függnek, nem veszik figyelembe a robot sebességét. Így a potenciáltér-alapú vezérlés tényleg egy kinematikai módszer, és nem feltétlenül működik, ha a robot gyorsan mozog.

## Reaktív irányítás

Eddig olyan irányítási módszerekkel foglalkoztunk, amelyek megkívánták a környezet bizonyos fokú ismeretét ahhoz, hogy akár referencia pályát, akár potenciálteret alkothassunk. Ezzel a megközelítéssel kapcsolatban azonban felmerülnek bizonyos problémák. Először is, sokszor nagyon nehéz pontos modellt alkotni a környezetről, különösen ha az komplex vagy nagyon távol van, mint mondjuk a Mars felszíne. Másodszor, még ha sikrül is pontos modellt készítenünk, a számítási nehézségek és a lokalizációs technikák hibája folytán alkalmazhatatlanná válnak ezek a módszerek. Bizonyos esetekben megfelelőbb a reflexszerű ágensek használata, ezt hívják **reaktív irányításnak** (**reactive control**).

Erre példa a 25.22. (a) ábra hexapodja, azaz hatlábú robotja, amelyet nehéz terepen való közlekedésre terveztek. A robot szenzorai többnyire alkalmatlanok arra, hogy



(a)

(b)

25.22. ábra. (a) Hexapod robot. (b) Kiterjesztett véges automata (KVA) (augmented finite state machine, AFSM) egyetlen láb vezérléséhez. Fontos, hogy ez a KVA-szenzoros visszacsatolásra reagál: ha a láb megakad az előremozgatás során, akkor magasabba emeli.

megfelelő pontosságú modellt alkossanak a környezetről, hogy a robot a korábban tár-gyalt pályatervezési algoritmusok bármelyikét használhassa. De hiába látnánk el megfelelő szenzorokkal, a tizenkét szabadságfok (kettő lábanként) túl nagy számítási kapacitást igényelne pályatervezés esetén.

Mindazonáltal a környezet explicit ismerete nélkül is lehetséges közvetlenül szabályozót tervezni a feladathoz. (Már láttuk ezt a PD szabályozó esetében, amely képes volt a komplex robotkart a megadott pályán tartani a robot dinamikájának pontos ismerete *nélküli*. Ugyanakkor szükség volt a kinematikai modell alapján megalkotott referenciapályára.) A hatlábú robot esetében megfelelő absztrakciós szinten meglepően könnyű meghatározni az irányítási törvényt. A megfelelő algoritmus ciklusosan mozgathatná a lábakat. Egy ideig a levegőben mozognak, utána pedig a földön támaszkodnak. A hat lábat úgy kell koordinálni, hogy egyszerre három (ellenétes oldalakon) minden a földön legyen, biztosítva a robot fizikai stabilitását. Egy ilyen irányítási algoritmust könnyű programozni, és remekül működik sík felületen. Nehéz terepen az egyes akadályok meggátolhatják a lábakat az előremozdulásban. Ezen a problémán egy egészen egyszerű irányítási szabálytal segíthetünk: *ha a láb mozgását valami blokkolja, a robot egyszerűen húzza vissza, emelje magasabbra, és próbálja újra!* Az eredményül kapott vezérlést mint véges automatát mutatja a 25.22. (b) ábra. Egy reflexszerű ágenshez állapotokat társít, ahol a belső állapot a gép aktuális állapotának indexe ( $s_1$ -től  $s_4$ -ig) jelöli.

Ennek az egyszerű visszacsatoláson alapuló szabályozónak a különböző változatai figyelemre méltóan robusztus lépegető vezérlést valósítanak meg, amellyel a robot görögös terepen is képes előrejutni. Világos, hogy ehhez a szabályozóhoz nem kell környezeti modell, és nem kell semmiféle keresést alkalmazni a vezérlés megalkotásához. Amikor egy ilyen vezérlőt használunk, a robot viselkedésének kialakításában nagyon fontos szerepet kapnak a környezetből visszacsatolásként érkező jelek. A szoftver magától még nem határozza meg, hogy éppen mi fog történni a robottal, ha belehelyezik az adott környezetbe. Az olyan viselkedést, ami egy (egyszerű) szabályozó és a (komplex) környezet kölcsönhatásaként jön létre **kibontakozó viselkedésnek (emergent behavior)** hívjuk. Szigorúan véve bármelyik eddig tárgyalt robotra mondhatnánk, hogy kibontakozó viselkedést mutat, hiszen egyetlen modell sem tökéletes. Történeti okok

miatt azonban ezt a kifejezést csak arra az irányítási technikára mondják, amelyik nem használ explicit környezeti modellt. A kibontakozó viselkedés jellemző tulajdonsága számos biológiai organizmusnak is.

Technikailag a reaktív irányítás csak egy implementációja az MDF-ek (illetve ha vannak belső állapotok, akkor POMDF-ek) stratégiáinak. A 17. fejezetben számos technikát ismertetünk, amelyek alkalmasak a robot modelljéből és a robot környezetének a modelljéből stratégiák generálására. A robotikában az ilyen stratégiák kézzel történő megszerkesztésének nagy gyakorlati jelentősége van, mivel képtelenek vagyunk tökéletes modelleket alkotni. A 21. fejezetben leírt megerősítéses tanulási módszerek segítségével tapasztalati úton lehet stratégiákat szerkeszteni. Néhány ezen módszerek közül – például a Q-tanulás és a stratégiakereső módszerek – nem igényel környezeti modellt, hanem a tanulás során összegyűjtött, nagy mennyiségi adatra támaszkodik. Így alkalmas jó minőségű szabályozók létrehozására robotok számára.

## 25.7. SZOFTVERARCHITEKTÚRÁK A ROBOTIKÁBAN

**Szoftverarchitektúrának (software architecture)** hívják az algoritmusok strukturálásának módszertanát. Egy architektúra általában nyelveket és eszközöket tartalmaz programok írásához, valamint egy átfogó szisztemát, amely alapján a programokat össze lehet illeszteni.

A modern robotikai szoftverarchitektúráknak sikeresen kell ötvözniük a reaktív irányítást és a modellalapú, tudatos vezérlést. Sok szempontból a reaktív és a tudatos irányítás eltérő előnyökkel és hátrányokkal rendelkezik. A reaktív szabályozás szenzoros információt alapul, és nagyon jól használható valós idejű, alacsony szintű döntéshozásra. Ugyanakkor ritkán ad globális szinten elfogadható megoldást, mert a globális döntések olyan információt igényelnek, amely a döntés meghozatalakor még nincs a robot birtokában. Ilyen esetekben a modellalapú tervezés a megfelelő.

Következésképpen a legtöbb robotarchitektúra reaktív technikákat használ az alacsony szintű irányításhoz, és modellalapút a magasabb szintűhöz. Amikor a PD szabályozóról beszélünk, számításba vettünk egy ilyen kombinációt, ahol a (reaktív) PD szabályozó együtt dolgozik a (modellalapú) pályatervezővel. **Hibrid architektúrának (hybrid architecture)** hívják általában azokat az architektúrákat, amelyek ötvözik a reaktív és a modellalapú technikákat.

### Alárendelt architektúra

Az **alárendelt architektúra (subsumption architecture)** (Brooks, 1986) egy olyan keretrendszer, amelyen belül lehetőség nyílik reaktív szabályozók véges automatákból történő összeállítására. Ezeknél az automatáknál, állapotgépeknél a csomópontok egyes szenzoros változók vizsgálatát tartalmazzák olyan esetekben, ahol a véges automatánál is a változó vizsgálatának eredménye határozza meg a végrehajtás menetét. Az éleket el lehet látni címkékkel, amelyek akkor generálódnak, amikor az éleken áthaladunk, és amelyeket a robot motorjához vagy egy másik véges automatához küldünk. Azonfelül a véges automatáknak vannak belső időzítőik (óráik), amik irányítják, hogy mennyi idő alatt lehet átutni egy-egy élen. Az így kapott állapotgépet szokás **kiterjesztett véges**

**automatának (KVA) – (augmented finite state machine, AFSM)** hívni, ahol a kiterjesztés a belső órajel használatára vonatkozik.

Egy egyszerű AFSM használatára példa a 25.22. (b) ábrán látható négyállapotú gép, amely ciklikus mozgást generál a hexapod robot számára. Ez az AFSM ciklikus szabályozót használ, aminek működése tulajdonképpen nem függ a környezetből jövő visszacsatolástól. Az előrenyújtási szakasz során viszont már használja a szenzoros információt. Ha a láb elakad, azaz nem tudja végrehajtani az előremozgást, a robot visszahúzza a lábat, majd magasabbra emelve újra próbálkozik. Azaz a szabályozó képes *reagálni* a robot környezettel való fizikai érintkezésére.

Az alárendelt architektúra további primitíveket szolgáltat az AFSM-ek szinkronizálásához, és a többszörös, esetleg ellentmondásban lévő AFSM-ek kimenetének összehangolásához. Ezáltal lehetővé válik a programozó számára, hogy egyre bonyolultabb szabályozókat építsen fel bottom-up, alulról felfelé technikával. Példánkban egy-egy AFSM-mel kezdhettünk az egyes lábak vezérléséhez, majd a következő AFSM már több láb működésének összehangolását végezheti. Mindezekfölött megvalósíthatunk valamilyen magas szintű viselkedést, mondjuk, egy ütközés elkerülését, amely már hátrálást és visszafordulást is tartalmaz.

Az ötlet, hogy AFSM-ekkel valósítsuk meg a robot szabályozását, igen nagy előrelépés. Képzeljük el, milyen nehéz lenne ugyanezt a viselkedést leírni a konfigurációs térben, bármelyik korábban tárgyalt pályatervező algoritmussal! Először is szükségünk lenne a terep pontos modelljére. A hatlábú robot konfigurációs tere összesen tizennyolc dimenziós (tizenkét dimenzió a lábankénti két-két motortól származik, hat pedig a robot pozíciója és orientációja a környezethez képest). Még ha a számítógépeink elég gyorsak is lennének, hogy kiszámoljanak egy pályát egy ilyen kiterjedt térben, további problémát jelentenének az egyéb előforduló hatások, mint például az, ha a robot megszűnik a lejtőn. Az ilyen sztochasztikus hatások miatt egyetlen út a konfigurációs térben csaknem bizonyosan túl törékeny lenne, és még egy PID szabályozó sem tudna megbirkózni a felmerülő problémákkal. Más szavakkal, a modellalapú mozgástervezés túl bonyolult feladat a mai robotok pályatervező algoritmusainak.

Sajnos azonban az alárendelt architektúrának is megvannak a maga problémái. Először is, az AFSM-ek általában feldolgozatlan szenzoros információt kapnak, ami csak akkor elegendő a döntéshozáshoz, ha az adatok megbízhatók, és tartalmazzák az összes szükséges információt. De ha a szenzoros adatokat nem triviális módon kell időben integrálni, akkor sikertelen lehet a döntéshozás. Éppen ezért az alárendelt típusú szabályozókat lokális feladatokra használják, mint például fal mentén haladás vagy egy látható fényforrás követése. Másodszor, a modell hiánya nehézkessé teszi a robot feladatának megváltoztatását. Az alárendelt típusú robot általában csak egy feladatot hajt végre, és nincs is elképzelése arról, hogy hogyan módosíthatná vezérlését úgy, hogy más feladatokat is el tudjon látni (mint a ganajtúró bogár a 2.2. alfejezetben). Végezetül, az alárendelt típusú vezérlést elég nehéz átlátni. A gyakorlatban több tucat bonyolult, együttműködő AFSM és a környezet kölcsönhatásait a legtöbb programozó már nem tudja követni. Ezen okok miatt az alárendelt architektúrát – nagy történeti jelentősége ellenére – ritkán szokták kereskedelmi forgalomba kerülő robotoknál használni. Ugyanakkor néhány lezármazottját már igen.

## Háromréteges architektúra

A hibrid architektúra modellbe épített tudással ötvözi a reaktivitást. Messze a legnépszerűbb hibrid architektúra az úgynevezett **háromréteges architektúra (three-layer architecture)**, amely egy reaktív, egy végrehajtó és egy modellező rétegből áll.

A **reaktív réteg (reactive layer)** alacsony szintű vezérlést biztosít a robot számára. Ezt egy szoros szenzorcselékvés-hurok jellemzi. Döntési ciklusa gyakran milliszekundum nagyságrendű.

A **végrehajtó vagy szekvenciális réteg (executive/sequencing layer)** kapcsolja össze a reaktív réteget a modellezővel. Direktívákat fogad a modellező rétegtől, és egy más után továbbítja azokat a reaktív réteg számára. A végrehajtó réteg kezelheti például a modellező réteg pályatervező algoritmusá által generált útvonalpontokat, és eldönti, hogy melyik reaktív viselkedést hívja meg. A döntési idő a végrehajtó réteg esetében átlagosan másodperc nagyságrendű. Ez a réteg felelős továbbá azért is, hogy a szenzoros információkat egy belsőállapot-reprezentációba integráljuk. Kezeli például a robot lokalizációs és online térképezési rutinjait.

A **modellező réteg (deliberate layer)** a beépített tudással komplex feladatokra globális megoldásokat generál, tervezéssel. A bonyolult feladatok nagy számítási igénye miatt a döntési ciklusidő perc nagyságrendű is lehet. A modellező vagy tervező réteg-modelleket használ a döntések meghozásához. Ezek a modellek lehetnek előzetesen betápláltak vagy adatokból tanulás útján létrejöttek, és általában a végrehajtó modell által gyűjtött állapotinformációt haszhálják.

A háromrétegű architektúra különböző változatait találhatjuk meg a legtöbb mai robot szoftverrendszerében. A három rétegre való bontás nem kötelező. Egyes robotszoftverrendszerök további rétegeket is tartalmaznak, például felhasználói interfész réteget – az emberekkel való kommunikálás vezérlésére –, vagy olyan rétegeket, amelyek a cselekvésnek a többi, ugyanabban a környezetben tevékenykedő robottal való összehangolásáért felelősek.

## Robotprogramozási nyelvek

Sok robotvezérlőnek megvan a saját külön programozási nyelve. Például az alárendelt architektúra sok programja a Brooks (Brooks, 1990) által megalkotott **viselkedési nyelvet (behavior language)** alkalmazza. Ez egy szabályalapú, valós idejű nyelv AFSM vezérlésekhez. A LISP-hez hasonló szintaxisú, egyedi szabályokat fordít AFSM-ekra, valamint több AFSM lokális és globális üzenetküldő mechanizmussal van integrálva.

Csakúgy, mint az alárendelt architektúra, a viselkedési nyelv is csak egyetlen AFSM-re képes fókusználni, és relatív alacsony szintű kommunikációt enged meg az egységek között. Az újabb kutatások erre az ötletre építve olyan nyelvek sorát fejlesztették ki, amelyek alapvetően hasonlítanak a viselkedési nyelvre, de nagyobb teljesítményük, és gyorsabban futnak. Ilyen nyelv például az általános **robot nyelv (generic robot language, GRL)**, amely Horswill (Horswill, 2000) nevéhez fűződik. A GRL egy funkcionális programozási nyelv nagyméretű moduláris irányítási rendszerekhez. Ugyanúgy, mint a viselkedési nyelv, a GRL is véges automatákból építkezik. Ezenfelül sokkal nagyobb lehetőséget hagy a különböző modulok közötti kommunikációs formák és

szinkronizációs lehetőségek definiálására. A GRL-programok hatékony imperatív nyelvnek fordítódnak le, mint amilyen például a C.

Egy másik fontos, modern programozási nyelv (és hozzá tartozó architektúra) a Firby (Firby, 1994) által megalkotott reaktív akciótervrendszer (reactive action plan system, RAPS). A RAPS lehetővé teszi, hogy a programozók célokot és ezekhez tartozó terveket (vagy részleges stratégiákat) definiáljanak, valamint feltételeket, amelyek esetén az adott terv nagy valószínűséggel sikeres lesz. Ami nagyon fontos, hogy a RAPS lehetőséget ad az esetleges *kudarc* kezelésére is, ami szükségszerű velejárója a valódi robotikai rendszereknek. A programozó számos érzékelő rutint specifikálhat a különböző típusú kudarcok esetére, és mindegyikhez definiálhat egy kivételkezelési rutint. Hárromrétegű architektúráknál gyakran használják a RAPS-ot a végrehajtó rétegben, az újratervezést nem igénylő kis eltérések, váratlan események kezelésére.

Számos más nyelv létezik, amely lehetővé teszi a robot számára, hogy mérlegeljen és tanuljon. Ilyen például a GOLOG (Levesque és társai, 1997b), amely akadályok nélkül egyesíti a modellalapú problémamegoldást (tervezést) és a reaktív irányítás közvetlen megvalósítását. A GOLOG-programok szituációkalkulus (10.3. alfejezet) formájában jutnak kifejezésre, nemdeterminisztikus mozgásoperátorok hozzáadásának a lehetőségével. Ha a vezérlőprogramban nemdeterminisztikus mozgások is specifikáltak, a programozónak teljes modellt kell szolgáltatnia a robotról és környezetéről. Amikor a vezérlőprogram egy nemdeterminisztikus választási lehetőséghöz ér, egy tervező algoritmust hív meg (szabálykészítő formában), hogy az meghatározza a következő lépést. Ilyen módon a programozó részleges vezérlőket fr, és a végső irányítási döntéshozást a beépített mechanizmusra bízza. A GOLOG szépsége a modellalapú és a reaktív vezérlés probléma nélküli összeillesztésében van. Annak ellenére, hogy a GOLOG alkalmazásának több feltétele van (teljes megfigyelhetőség, diszkrét állapotok, teljes modell), sok beltéri robot számára nyújt megfelelő, magas szintű irányítást.

A CES, C++ beágyazott rendszerekhez (C++ for embedded systems), a C++ nyelv kiterjesztése, amely már valószínűségeket és tanulást is tartalmaz (Thrun, 2000). A CES adattípusai valószínűségi eloszlások, amelyek a programozó számára lehetővé teszik a bizonytalan információk kezelését a korábbi nehézségek nélkül. Ami még fontosabb, hogy a CES lehetőséget nyújt a robot szoftverpéldákkal való tanítására, nagyon hasonlóan a 20. fejezetben tárgyalt tanulási algoritmusokhoz. Ennek kihasználásával a programozók hagyhatnak helyet a programban a később megtanulható függvények számára – ezek tipikusan differenciálható parametrikus reprezentációk, mint például a neurális hálók. Ezeket a függvényeket a külön tanulási fázisban tanulja meg a rendszer, ahol a tanító határozza meg a kívánt kimeneti viselkedést. A CES sikeresnek bizonyult részlegesen megfigyelhető és folytonos esetekben is.

Az ALisp (Andre és Russell, 2002) a LISP továbbfejlesztett változata. Az Alisp lehetővé tesz nemdeterminisztikus döntési pontok megadását, hasonlóan a GOLOG döntési pontjaihoz. De ahelyett, hogy a tételezbizonyítóra bízná a döntést, az ALisp induktív módon, megerősítéssel tanulással megtanulja a helyes cselekvést. Az ALisp-ot rugalmas eszközöknek tekinthetjük a tárgytartományról szóló – főleg a megfelelő döntések szubrutinjainak hierarchikus szerkezetére vonatkozó – tudásnak megerősítéssel tanulással tanuló rendszerbe való beépítésére. Eddig csak szimulációk során alkalmazták az ALisp-ot, de nagyon ígéretes módszernek tűnik olyan robotok építéséhez, amelyek tanulnak a környezetükkel való kölcsönhatásból.

## 25.8. ALKALMAZÁSI TERÜLETEK

Most pedig felsorolunk néhányat a robottechnológia fontos alkalmazási területei közül.

**Ipar és mezőgazdaság.** Hagyományosan azok az ágazatok használnak robotokat, amelyek nehéz, fizikai jellegű emberi munkát igényelnek, de mégis megfelelően strukturálhatók és automatizálhatók. A legjobb példa az összeszerelő üzemi gyártósor, ahol a manipulátorok rutinszerűen végzik feladatukat, az összeszerelést, az alkatrészek behelyezését, az anyagkezelést, a hegesztést és a festést. Sok esetben a robotok sokkal költséghatékonyabbak, mint a humán munkaerő.

A szabadban sok nehéz gép helyét robotok vették át a betakarításnál, a bányászatban, a földmunkáknál. Például nemrégiben a Carnegie Mellonon, egy projekt során bemutatták, hogy a robotok 50-szer gyorsabban képesek levakarni a festéket egy hajóról, mint az emberek, és sokkal kevesebb kárt tesznek a környezetben. Autonóm bányász-robotok prototípusai bizonyították, hogy sokkal gyorsabban és pontosabban képesek kitermelni az ércet a föld alatti bányákban, mint az emberek. Robotokat használnak az elhagyatott bányák és csatornarendszerek nagy pontosságú feltérképezéséhez is. Ugyan sok rendszer még csak kísérleti stádiumban van, pusztán idő kérdése, hogy a robotok mikor veszik át az emberektől a félig mechanikus munkák többségét.

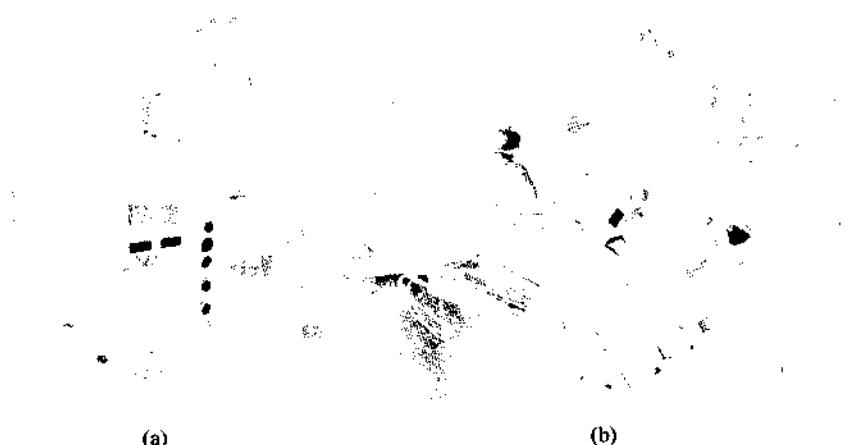
**Szállítás.** A robotizált szállításnak sok oldala van. Az autonóm helikopterekkel kezdve, amelyek nehezen megközelíthető helyekre juttathatnak el szállítmányokat, az automata kerekesszéken át (amik olyan embereket szállítanak, akik nem tudnák irányítani a hagyományos kerekesszéket), egészen az autonóm rakodógépekig, amelyek felülműlják a dokkokban a szakképzett humán sofőrt is a konténerek hajóra és teherautóra való bepakolásánál. Jó példa a beltéri szállító robotokra a 25.23. (a) ábrán látható Helpmate nevű robot. Ezt a robotot kórházak tucatjaiban használják étel és egyéb orvosi felszerelés szállítására. A kutatók olyan autószerű robotot is kifejlesztettek, amely képes önállóan navigálni az autópályán, és terepen is megállja a helyét. Üzemű körülmények között a szállítórobotok ma már rutinszerűen látják el a szállítási feladatokat raktárakban és a gyártósorok között.

Ezen robotok közül sok igényli, hogy munkája elvégzéséhez változtassunk valamit környezetén. A leggyakoribb, hogy tájékozódást segítő elemeket kell felszerelni, például induktív köröket a padlóba, aktív fényjeleket, vonalkódokat vagy GPS-adókat. Jelenleg még mindig nagy kihívás olyan robotot tervezni, amely képes a természetes környezetben tájékozódni, és nem igényel mesterséges eszközöket. Ez különösen fontos olyan eseteken, mint például a mélytengeri kutatások, ahol a GPS nem elérhető.

**Veszélyes környezet.** Robotok segítettek az embereknek a nukleáris hulladékok eltarthatásában. A legnevezetesebb helyszín Csernobil és Three Mile Island volt. A robotok ott voltak a World Trade Center összeomlása után is, és olyan helyekre is bementek, amelyek a mentőcsapatok számára túl veszélyesek voltak.

Nehány országban lőszér szállítására és – ami különösen veszélyes feladat – bombák hatástalanítására használnak robotokat. Számos kutatás irányul napjainkban aknaszedő robotok kifejlesztésére – minden szárazföldi, minden tengeri használatra. A legtöbb ilyen célú robot távirányítással működik, azaz emberek vezérlik messzirol őket. Ezen robotok önállóvá tétele a következő fontos fejlesztési lépés.

**Felfedezés.** A robotok eljutottak oda is, ahová az ember még soha, például a Mars felszínére (lásd 25.1. (a) ábra). Robotkarok segítik az űrhajósokat a szatellitek befo-



**25.23. ábra.** (a) A Helpmate robot élelmiszeret és egyéb orvosi felszerelést szállít több tucat kórházban, világszerte. (b) Sebészrobotok a műtőben (da Vinci Surgical Systems).

gásában és pályára állításában, valamint a Nemzetközi Ūrálomás (ISS) építésében. Robotok segítenek a tenger alatti kutatásokban. Gyakran használják őket elsüllyedt hajók feltérképezésére. A 25.24. ábrán egy elhagyatott szénbányát feltérképező robot látható, a távolságszenzorai által a járatról alkotott 3D modellel együtt. 1996-ban tudósok egy járórobotot engedtek le egy működő vulkán kráterébe, és fontos adatokat gyűjtöttek klímataligiai kutatásokhoz. Ember nélküli légi járműveket, más néven drónokat (*drones*) használnak katonai műveletekhez. A robotok egyre hatékonyabbak az információgyűjtésben olyan területeken, amelyek nehezen megközelíthetők (vagy veszélyesek) az ember számára.

**Egészségügy.** A sebészorvosok munkáját segítendő egyre többször használnak robotokat az orvosi eszközök pontos mozgatásához olyan kényes szerveket érintő műtétek esetében, mint az agy, a szív vagy a szem. A 25.23. (b) ábra egy ilyen rendszert mutat be. A robotok – nagy pontosságuknak köszönhetően – nélkülvilágosan eszközökkel váltak bizonyos csípőprotézisek beültetésénél. Pilottanulmányok kimutatták, hogy végbéltükrözés esetén a robotizált eszközök használata csökkenti a sérülés veszélyét. A műtőn kívül, a kutatók megkezdték olyan robotok kifejlesztését, amelyek idős vagy fogyatékos emberek segítségére lehetnek, mint például az intelligens robotizált járogépek vagy intelligens játékok, amelyek figyelmeztetnek a gyógyszerek bevételére.

**Személyi kiszolgálók.** A szolgáltatás a robotok nagy, igéretes alkalmazási területe. A szervizrobotok a minden nap teendők elvégzésében segédkeznek. A kereskedelmi forgalomban is kapható szervizrobotok között vannak autonóm porszívók, fűnyírók és golflabdászedők. Ezen robotok mindegyike önállóan, emberi segítség nélkül képes navigálni, és teljesíteni feladatát. Néhány szervizrobot nyilvános helyeken tevékenykedik, mint például az idegenvezetőként alkalmazott robotizált információspult a bevásárlóközpontokban, a vásárokon vagy a múzeumokban. A szervízfeladatok ellátása emberi interakciót kíván, és azt, hogy a robot megfelelően boldoguljon kiszámíthatatlan és dinamikus környezetekben.



**25.24. ábra.** (a) Egy elhagyatott bánya feltérképezése egy robottal. (b) A bányának a robot által kapott 3D képe.

**Szórakozás.** A robotok megkezdték a játék- és szórakoztatóipar meghódítását is. A 25.4. (b) ábrán már találkoztunk a Sony AIBO-val, ami bár kutyaszerű játék, világszerte használják MI-kutatólaborokban különböző projektek platformjaként. Az egyik, nagy kihívást jelentő feladat, ahol az AIBO-kat fizikai eszközökkel használták, a **robotfoci (robotic soccer)**: versengés két csapat között, az emberi fociohoz hasonló szabályokkal, de autonóm mobil robotokkal. A robotfoci nagyon jó lehetőségeket kínál az MI-kutatásokhoz, mert sok olyan problémát foglal magában, ami másol, komolyabb alkalmazásokban is megjelenik. Az évente megrendezésre kerülő robotfoci-bajnokságok sok MI-kutatót vonzzanak, és érdekesebbé, izgalmasabbá tették a robotikának ezt az ágát.

**Emberi kiegészítők.** Az utolsó alkalmazás, amiről szó ejtünk, az emberi kiegészítők. A kutatók kifejlesztettek már a kerekesszékhez hasonló lépegető járógépet, amelyek alkalmasak emberek szállítására. Számos helyen jelenleg arra koncentrálnak, hogy olyan eszközöket alkossanak, amelyek megkönnyítik az embernek a járást vagy a karok mozgatását egy, a testhez kívülről csatlakoztatott vázzal, amely képes pluszerőt kifejni. Ha valakire tartósan rögzítenek egy ilyen szerkezetet, akkor mesterséges végtagnak is tekinthető. A robotok általi távjelenlét, illetve teleoperáció is felfogható emberi kiegészítésnek. A teleoperáció azt jelenti, hogy robotikai eszközök segítségével nagy távolságból tudunk megadott feladatokat végrehajtani. A teleoperáció népszerű megvalósítása a mester–szolga (master-slave) elvű konfiguráció, amikor is a robot manipulátor a távolban lévő, humán operátor mozdulatait követi, egy heptikus (érzetet is közvetítő) interfésszel keresztül. Ezek a rendszerek mind jobban kiterjesztik az emberi képességeket a környezettel való tökéletesebb kölcsönhatás érdekében. Néhány projekt azt tűzte ki célul, hogy lemásolja az embert, legalábbis felületes szinten. Japánban már számos cégnél kaphatók humanoid robotok.

## 25.9. ÖSSZEFOLGLALÁS

A robotika tulajdoképpen nem más, mint intelligens ágensek, amelyek képesek megváltoztatni a fizikai világot. Ebben a fejezetben a következő alapvető ismereteket sajtáltottuk el a robothardverrel és -szoftverrel kapcsolatban.

- A robotok szenzorokkal vannak felszerelve, hogy érzékelhessék az őket körülvevő világot, és beavatkozó szervekkel, hogy fizikai erőkkel hathassanak környezetükre. A legtöbb robot vagy manipulátor, amely egy adott helyhez van rögzítve, vagy mobil robot, ami képes mozogni.
- A robotikai érzékelés a döntéshez kapcsolódó mennyiségeknek a szenzoros adatokból való becslése. Ehhez szükségünk van egy belső reprezentációra, valamint egy olyan módszerre, amivel azt időnként frissíthetjük. A nehéz érzékelési problémák gyakori példái a helymeghatározás és a térképezés.
- Valósínnőség-alapú szűrő algoritmusok, mint például a Kalman-szűrők és a részcecskeszűrők hasznosak a robotérzékelés szempontjából. Ezek a technikák frissítik a belső világképet, például az állapotváltozók a posteriori eloszlását.
- A robot mozgásának tervezése általában a konfigurációs térben történik, ahol minden pont egyértelműen meghatározza a robot egy adott pozícióját és orientációját, csuklóinak szögét.
- A konfigurációs térben kereső algoritmusok között fontos megemlíteni a celladekompozíciót, amely a teljes konfigurációs teret véges sok cellára bontja fel, valamint a szkeletonizációs technikákat, amelyek az adott konfigurációs teret egy alacsonyabb dimenzióba vetítik le. A mozgástervezés problémáját ezután már egyszerűbb struktúrákban való kereséssel is meg lehet oldani.
- A kereső algoritmus által megtalált pályán a robot akkor tud végighaladni, ha egy PID szabályozót teszünk a rendszerbe, aminek referencia pályája a kívánt útvonal.
- A potenciáltér-alapú technikák a céltól és az akadályoktól való távolság szerint definiált potenciálfüggvények segítségével navigálják a robotot. Ezek a technikák megakadhatnak lokális minimumhelyeken, de képesek mozgásgenerálásra előzetes tervezés nélkül.
- Néha egyszerűbb egy robotszabályozót közvetlenül megtervezni, mint a pályát a környezet explicit modelljéből levezetni. Egy ilyen szabályozót legtöbbször leírtunk véges automataként.
- Az alárendelt architektúra a programozók számára lehetővé teszi, hogy belső órajellel kiegészített véges automaták összekapsolásával robotvezérlőket alkossanak.
- A népszerű háromrétegű architektúra olyan robotszoftver fejlesztéséhez is keretet ad, amely integrálja a modellalapú tervezést, a részfeladatokra bontást és a szabályozást.
- Léteznek feladatspecifikus robotprogramozási nyelvek, amelyek megkönnyítik a robotszoftverek fejlesztését. Ezek a nyelvek kész szerkezetekkel segítik a többszálú programok írását, a szabályozási irányelveknek a tervezésbe történő integrálását és a tapasztalatalapú tanulást.

## Irodalmi és történeti megjegyzések

A **robot** szót a cseh drámaíró, Karel Čapek népszerűsítette 1921-ben írt *R.U.R.* (Rossum's Universal Robots) c. színművével. Az ő robotjait nem mechanikusan szerelték össze, hanem kémiai úton állították elő. A műben a robotok megharagszanak mestereikre, és elhatározzák, hogy átveszik az uralmat. Úgy tűnik (Glanc, 1978), hogy valójában Čapek bátyja, Josef alkotta meg a robot szót a cseh „robota” (robotolás) és „robotník” (jobbágy) szó összevonásával, 1917-es *Opilec* c. novellájában.

A robotika kifejezést Asimov használta először 1950-ben, ugyanakkor a robotika története – más megnevezéssel – sokkal régebbre nyúlik vissza. Az ókori görög mitológiában Taloszt, a mechanikus embert állítólag Héphaistosz, a kovácsisten készítette. Jacques Vaucanson egy csodálatos automatát épített a 18. században. Az 1738-as mechanikus kacsa a korai robotok egy példája, aminek összetett viselkedése még fixen volt beleépítve. Az első programozható, robotszerű alkotás talán Jacquard szövőszéke volt (1805), amiről már az 1. fejezetben már szót ejtettünk.

Az első kereskedelmi forgalomba került robot a Unimate (universal automation) manipulátor volt. A Unimate-et Joseph Engelberger és George Devol fejlesztette. 1960-ben adták el az első példányt a General Motorsnak, ahol tv-képcsövek gyártásánál használták. Szintén 1960-ben nyújtotta be Devol az első robotokra vonatkozó szabadalmat az Egyesült Államokban. 11 évvel később a Nissan Corp. az elsők között oldotta meg egy teljes szerelősor automatizálását. A fejlesztést a Kawasaki végezte Engelberger és Devol Unimation nevű cégtől származó robotokkal. Főként Japánban és az USA-ban ez az előrelépés jelentős átalakulásnak nyitott utat, amely mind a mai napig tart. A Unimationt a PUMA robot fejlesztése követte 1978-ban. A PUMA a Programmable Universal Machine for Assembly (programozható univerzális gép szereléshez) névből származik, és eredetileg a General Motors számára fejlesztették. A következő évtizedekben ez lett a robotmanipulátorok *de facto* szabványa. Manapság egymilliára becsülük a világon dolgozó robotok számát, és ezeknek több mint fele Japánban tevékenykedik.

A robotikai kutatásokról szóló szakirodalom nagyjából két részre osztható: mobil robotokra és rögzített manipulátorokra. A Grey Walter által 1948-ban készített „teknőst” tekinthetjük a világ első autonóm mobil robotjának annak ellenére, hogy a rendszer nem volt programozható. A '60-as években, a Johns Hopkins Egyetemen készített „Hopkins Beast” már sokkal kifinomultabb volt. Mintafelismerő hardverével azonosította a szabványos fali AC-csatlakozókat, és képes volt mozgása során megtalálni azokat, rácsatlakozni és feltölteni akkumulátorát. Mégis, a Beastnek nagyon limitált képességei voltak. Az első általános célú robot a „Shakey” volt, amit a Stanford Research Institute-ban (SRI) fejlesztettek ki az 1960-as évek végén (Fikes és Nilsson, 1970; Nilsson, 1984). Shakey volt az első robot, amelybe integrálták az érzékelést, a tervkészítést és a végrehajtást. Ez a jelentős eredmény számos további MI-kutatást inspirált. Szintén jelentős hatású projektek voltak még a Stanford Cart és a CMU Rover (Moravec, 1983). A mobil robotokkal kapcsolatos klasszikus munkákat Cox és Wilfong foglalták össze (Cox, és Wilfong, 1990).

A robottérképezés gyökerei két különböző helyről erednek. Az egyik szál Smith és Cheeseman munkásságával kezdődött (Smith és Cheeseman, 1986), akik Kalman-szűrőt alkalmaztak szimultán helymeghatározási és térképezési problémákhoz. Az algoritmust először Moutarlier és Chatila implementálta 1989-ben, majd Leonard és Durrant-Whyte terjesztették ki (Leonard és Durrant-Whyte, 1992). Dissanayake és társai 2001-ben írták le az ágazat jelenlegi fejlettségi szintjét (Dissanayake és társai, 2001). A másik szál a **foglalási hálózat** (**occupancy grid**) reprezentáció megalkotásával kezdődött, amelyet a valószínűségi térképezésnél használnak. A foglalási hálózat minden  $(x, y)$  pontra meghatározza annak a valószínűségét, hogy ott egy akadály van (Moravec és Elfes, 1985). A modern robottérképezésről Thrun írt összefoglalót (Thrun, 2002). Kuipers és Levitt (Kuipers és Levitt, 1988) az elsők között voltak, akik metrikus helyett topológikus térképezést javasoltak, az ember térbeli tájékozódását véve alapul.

A korai mobil robot helymeghatározási technikákat Borenstein és társai tekintették át (Borenstein és társai, 1996). Annak ellenére, hogy a Kalman-szűrő évtizedeken át jól ismert helymeghatározási módszer volt az irányításelméletben, a helymeghatározás problémájához tartozó általános valóságossági szabály csak jóval később jelent meg az MI-irodalomban, Tom Dean és kollégái (Dean és társai, 1990), valamint Simmons és Koenig (Simmons és Koenig, 1995) munkássága révén. Ez utóbbi páros vezette be a **Markov-helymeghatározás (Markov localization)** fogalmát. A technika első, valósvilág-beli alkalmazását Burgard és társai mutatták be (Burgard és társai, 1999) egy sor roboton keresztül, amelyeket múzeumokba telepítettek. A részecskeszűrő-alapú Monte Carlo helymeghatározás, amelyet Fox és társai fejlesztették ki (Fox és társai, 1999), manapság széles körben használatos. A **Rao-Blackwellized részecskeszűrő** kombinálja a részecskeszűrést a robot lokalizációval, valamint az egzakt szűrést a térképalkotással (Murphy és Russell, 2001; Montemerlo és társai, 2002).

A mobil robotokhoz kapcsolódó kutatások szimulálásának az elmúlt évtizedben két jelentős verseny is teret adott. Az AAAI éves robotbajnokságát 1992 óta rendezik meg. Az első verseny győztese a CARMEL volt (Congdon és társai, 1992). A fejlődés azóta igen jelentős és töretlen: a 2002-es verseny során például a robotoknak be kellett járniuk a konferenciaközpontot, megtalálni a regisztrációs pultot, regisztrálni a konferenciára, és végül tartani egy beszédet. Az 1995-ben, Kitano és kollégái (Kitano és társai, 1997) által elindított **Robocup** kezdeményezés célja, hogy 2050-re olyan teljesen autonóm humanoid robotokból álló futballcsapatot fejlesszen, „amely képes megvern az aktuális emberi világbanokcsapatot”. Különböző bajnokságokat szerveznek kerekess, eltérő méretű, szimulált robotoknak és négylábú Sony Aibóknak. 2002-ben a világbanokságra több mint 30 országból érkeztek csapatok, és a rendezvény legalább 100 000 nézőt vonzott.

A robotmanipulátorok, eredeti nevükön kéz–szem gépek (**hand-eye machine**) tanulmányozása meglehetősen más vonalon fejlődött. Egy ilyen kéz–szem gép megalkotására az első igazi kísérlet Heinrich Ernst MH-1-eze volt, amelyet PhD-téziseiben írt le az MIT-n (Ernst, 1961). Az edinburghi Gépi Intelligencia (Machine Intelligence) program már igen korán jelentős eredményeket mutatott fel látásalapú összeszerelő rendszerükkel, amit FREDDY-nek hívtak (Michie, 1972). Ezen úttörő kísérletek után rengetegen foglalkoztak geometriai algoritmusokkal determinisztikus és teljesen megfigyelhető mozgástervezési problémához. A robotmozgás-tervezés P-TÁR nehézségét Reif egy termékenyítő cikke mutatta be (Reif, 1979). A konfigurációtér-reprezentációt Lozano-Perez-nek (Lozano-Perez, 1983) köszönhetjük. Nagy befolyást gyakoroltak Schwartz és Sharir publikációi az általuk **zongoraszállítóknak (piano movers)** nevezett problémáról (Schwartz és társai, 1987).

A konfigurációs tér rekurzív cellákra bontásával való tervezés Brookstól és Lozano-Perez-től származik (Brooks és Lozano-Perez, 1985), ezt később Zhu és Latombe fejlesztették jelentősen tovább (Zhu és Latombe, 1991). A legkorábbi szkeletonizációs algoritmusok a Voronoi-diagramokon (Rowat, 1979) és a **láthatósági gráfokon (visibility graph)** alapultak (Wesley és Lozano-Perez, 1979). Guibas és társai (Guibas és társai, 1992) hatékony technikát fejlesztettek ki a Voronoi-diagramok inkrementális számítására. Choset (Choset, 1996) általánosította sokkal átfogóbb mozgástervezési problémára a Voronoi-diagramokat. John Canny alapozta meg PhD-tézisével (Canny, 1991) az első különálló exponenciális algoritmust pályatervezésre. Másfélé szkeletonizációs módszert használt, amit **szíluett (silhouette)** algoritmusnak hívnak. Jean-Claude

Latombe írása (Latombe, 1991) a különböző mozgástervezési metódusok széles skáláját fedi le. Kavraki és társai valószínűségi útitervet fejlesztettek (Kavraki és társai, 1991), amely még ma is a leghatékonyabb. Lozano-Perez és társai (Lozano-Perez és társai, 1984), valamint Canny és Reif (Canny és Reif, 1987) is foglalkoztak a finom mozgások tervezésével korlátozott érzékelés esetén, és intervallumra vonatkoztatott bizonytalanságot használtak valószínűségi bizonytalanság helyett. A referenciapont-alapú navigálás a mobil robotok esetében sok hasonló ötletet használ (Lazanas és Latombe, 1992).

A robotok mint dinamikus rendszerek irányítása, legyen szó manipulátorról vagy mobil robotról, hatalmas irodalommal rendelkezik, amelynek anyagáról ebben a fejezetben alig-alig esett szó. A fontos művek közé tartozik Hogan trilógiája (Hogan, 1985) az impedancia szabályozásáról és egy általános tanulmány a robotok dinamikájáról Featherstone-tól (Featherstone, 1987). Dean és Wellman az elsők között voltak, akik megpróbálták összekötni az irányításelméletet és az MI-tervezőrendszereket (Dean és Wellman, 1991). Három klasszikus tankönyv a robotkarok matematikájáról Paultól (Paul, 1981), Craigtól (Craig, 1989) és Yoshikawától (Yoshikawa, 1990) származik. A megfogás (grasping) tárgykörre szintén nagyon fontos a robotikában: egy stabil megfogás meghatározásának problémája meglehetősen nehéz (Mason és Salisbury, 1985). A megfelelő megfogás érintkezésérzékelőket vagy **heptikus visszaesatolást (haptic feedback)** kíván a fellépő erők és az esetleges csúszás meghatározására (Fearing és Hollerbach, 1985).

A potenciáltér-vezérlést, amely egyszerre próbálja megoldani a mozgástervezés és az irányítás problémáját, Khatib vezette be a robotika irodalmába (Khatib, 1986). Mobil robotok esetében ezt az ötletet részleges megoldásnak tekintették az ütközések elkerülésének problémájához. Később Borenstein (Borenstein, 1991) továbbfejlesztette az algoritmust, és a **vektortérhisztogram (vector field histogram)** nevet adta neki. A navigációs flüggvényeket, a determinisztikus MDF-hez tartozó vezérlési irányelvek robotikai megfelelőjét, Koditschek vezette be (Koditschek, 1987).

A robotokhoz tartozó szoftverarchitektúra témája sok heves vitát váltott ki. Az MI jó őreg zászlóshajója – a háromrétegű architektúra – Shakey tervezésének eredménye, később Gat újította fel (Gat, 1998). Az alárendelt architektúra Brooks munkája (Brooks, 1986), bár tőlük függetlenül Braitenberg (Braitenberg, 1984) is fejlesztett hasonlót, akinek könyve, a *Járművek*, egy sor egyszerű viselkedésalapú robotot mutat be. Brooks hatlábú lépegető robotjának sikerét sok más projekt követte. Connell PhD-munkája keretében (Connell, 1989) olyan teljesen reaktív mobil robotot fejlesztett, amely képes volt tárgyakért elmenni és azokat elhozni. A viselkedésalapú paradigma többrobotos rendszerekre való kiterjesztését megtalálhatjuk Mataric (Mataric, 1997) és Parker (Parker, 1996) munkáiban. A GLR (Horswill, 2000) és a COLBERT (Konolige, 1997) robotok megvalósították a párhuzamosított viselkedésalapú irányítást általános robotprogramozási nyelveken. Arkin átfogó képet nyújtott a téma aktuális helyzetéről (Arkin, 1998).

A 7. fejezetben leírt **helyezett automatákat (situated automata)** (Rosenschein, 1985; Kaelbling és Rosenschein, 1990) szintén használják mobil robotok irányítására, felderítési és szállítási feladatoknál. A helyezett automaták szorosan kapcsolódnak a viselkedésalapú tervezéshez abból a szempontból, hogy véges automatákból állnak, amelyek egyszerű kombinációs hálózatokkal nyomon követik a környezet állapotának bizonyos aspektusait. Amíg a viselkedésalapú megközelítés az explicit reprezentáció

hiányát hangsúlyozza, a helyezett automatákat algoritmikusan építik deklaratív környezeti modellekkel, így minden állapotregiszter által reprezentált tartalom jól definiált.

Számos jó kézikönyv jelent meg mostanában a mobil robotokról. A fent említetteken felül Kortenkamp és társai (Kortenkamp és társai, 1998) gyűjtése minden részletre kiterjedő áttekintést nyújt a mai mobil robot architektúrákról és rendszerekről. Két újabb könyv, Dudek és Jenkin (Dudek és Jenkin, 2000), valamint Murphy (Murphy, 2000) munkája általánosabb megközelítésben tárgyalja a robotikát. Mason könyve (Mason, 2001) a robotkarokról olyan modern témákat is tárgyal, mint például az önbeálló mozgás. A robotikával foglalkozó legjelentősebb konferencia az *IEEE International Conference on Robotics and Automation*. A témával foglalkozó folyóiratok közül kiemelkedik az *IEEE Robotics and Automation*, az *International Journal of Robotics Research* és a *Robotics and Autonomous Systems*.

## Feladatok

- 25.1.** A Monte Carlo-helymeghatározás *torzított* bármilyen véges méretű mintára. Azaz az algoritmus által kiszámolt várható érték a részecskeszűrés működéséből adódóan eltér a tényleges várható értéktől. Ebben a példában ezt a torzitást kell kiszámolni.

Hogy kicsit egyszerűsítsük a problémát, vegyünk egy teret, amelyben a robotnak négy lehetséges állapota (helyzete) van:  $X = \{x_1, x_2, x_3, x_4\}$ . Kezdetben egyenletes eloszlással választunk  $N \geq 1$  mintát a megadott állapotok közül. Mint általában, itt is tökéletesen elfogadható, ha egymás után akár több is generálódik bármelyik  $X$  állapotból. Legyen  $Z$  egy bináris szenzorváltozó, az alábbi feltételes valószínűségekkel:

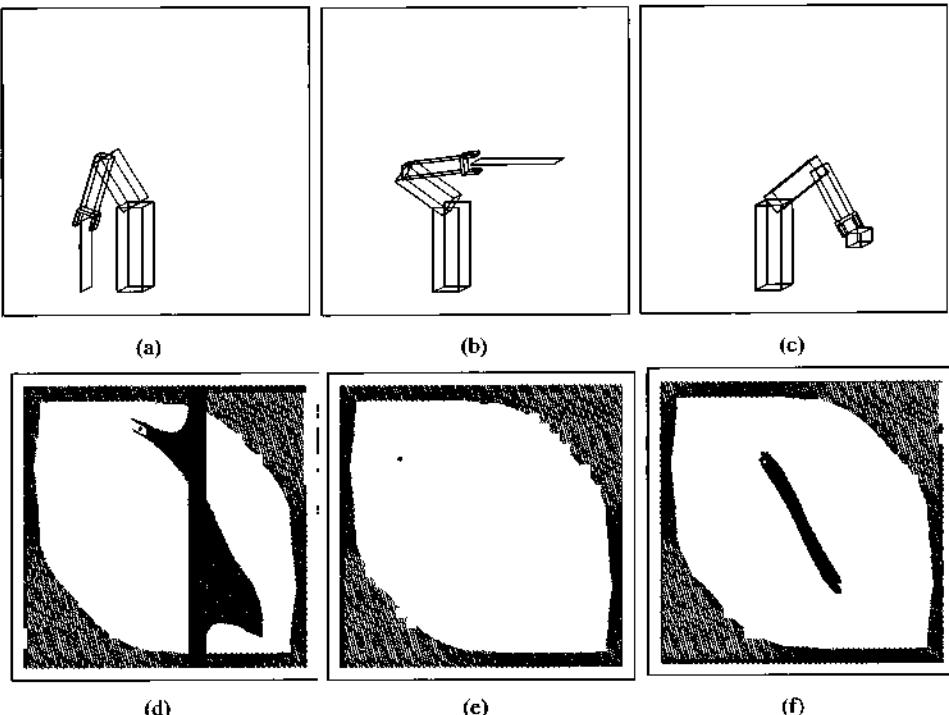
$$\begin{array}{ll} P(z|x_1) = 0,8 & P(\neg z|x_1) = 0,2 \\ P(z|x_2) = 0,4 & P(\neg z|x_2) = 0,6 \\ P(z|x_3) = 0,1 & P(\neg z|x_3) = 0,9 \\ P(z|x_4) = 0,1 & P(\neg z|x_4) = 0,9 \end{array}$$

A Monte Carlo-helymeghatározás ezeket a valószínűségeket használja, hogy generálja a részecskék súlyát, amelyeket azután normalizál és felhasznál az újabb mintavételezésnél. Az egyszerűség kedvéért feltételezzük, hogy az újabb mintavételezésnél,  $N$  nagyságától függetlenül, csak egyetlen új mintát generálunk. Ez a minta korrelálhat bármelyik  $X$  állapottal. Így a mintavező folyamat  $X$ -re nézve meghatároz egy valószínűség-eloszlást.

- (a) Mi az új mintára eredményül kapott valószínűség-eloszlás  $X$ -re nézve? Válaszolja meg a kérdést az  $N = 1, \dots, 10$  esetekre, valamint  $N = \infty$ -re.
- (b) A  $P$  és  $Q$  két valószínűség-eloszlás közötti különbséget mérhetjük a KL divergenciával, amelynek definíciója:

$$KL(P, Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

- (c) Melyek a KL divergencia értékei az (a)-beli eloszlások és a valódi posterior között?
- (d) Milyen változtatást hajtana végre a feladat szövegezésén (nem az algoritmuson!) annak érdekében, hogy garantáljuk a fenti specifikus becslő torzítatlanságát még véges  $N$ -re is? Legalább két különböző megoldást adjon (mindkettőnek elégsgesnek kell lennie).
- 25.2.** Alkalmazza egy pástázó távolságmérővel ellátott szimulált robot esetében a Monte Carlo-helymeghatározási algoritmust. A rácsterkép és a távolságadatok elérhetők az [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu) kódtárából. A megoldás akkor teljes, ha bemutatja a robot sikeres globális helymeghatározását.
- 25.3.** Vegyük a 25.12. ábrán látható robotkart. Tegyük fel, hogy a robot alapja 60 cm hosszú, a felkarja és az alkarma pedig egyaránt 40-40 cm. Ahogy a 25.4. alfejezetben beláttuk, az inverz kinematikai feladat megoldása sokszor nem egyértelmű. Állítson fel egy explicit, zárt alakú megoldást az inverz kinematikára erre a robotarra. Pontosan milyen feltételek mellett egyértelmű a megoldás?
- 25.4.** Implementáljon egy algoritmust, amely kiszámítja egy tetszőleges ( $n \times n$ -es logikai tömbbel) adott 2D környezet Voronoi-diagramját. Illusztrálja algoritmusát a Voronoi-diagram felrajzolásával 10 érdekes térképen. Mekkora az algoritmus komplexitása?
- 25.5.** Ez a feladat feltárja a kapcsolatot a munkatér és a konfigurációs tér között, a 25.25. ábra példáit használva.
- Vegyük a 25.25. (a)-tól (c)-ig látható robotkonfigurációkat, nem véve figyelembe a diagramokon feltüntetett akadályokat. Rajzolja fel az ezeknek megfelelő robotkar-konfigurációkat a konfigurációs térből. (Segítség: minden egyes karkonfiguráció egyetlen pontra képződik le a konfigurációs térből, ahogyan ezt a 25.12. (b) ábra is mutatja.)
  - Rajzolja meg a konfigurációs teret a 25.25. (a)–(c) ábrákon látható minden egyes munkatérdiagramra. (Segítség: a konfigurációs terek megegyeznek a 25.25. (a) ábrán lévővel, ami a robot önmagával való ütközését jelentő területet illeti, de különbségek adódnak a terület határát jelentő akadályok hiányából és az egyes ábrákon lévő akadályok különböző helyzetéből.)
  - A 25.25. (e)–(f) ábrákon látható fekete pontok mindegyikéhez rajzolja meg a hozzá tartozó konfigurációt a munkatérben. Az árnyékolt részeket hagyja figyelmen kívül.
  - A 25.25. (e)–(f) ábrákon látható konfigurációs terek mindegyikét egy-egy különálló (a munkatérben lévő) akadály generálta (ez a sötétebb terület), valamint azok a megszorítások, amelyek a robot önmagával való ütközéséből erednek (a világosabb árnyalat). Rajzolja meg minden diagramhoz a sötét területek megfelelő akadályt a munkatérben.
  - A 25.25. (d) ábra azt mutatja be, hogy egyetlen síkbeli (lapos) akadály két elválasztott részre tudja bontani a konfigurációs teret. Maximum hány elválasztott részre tudja bontani az egybefüggő akadálymentes teret egyetlen

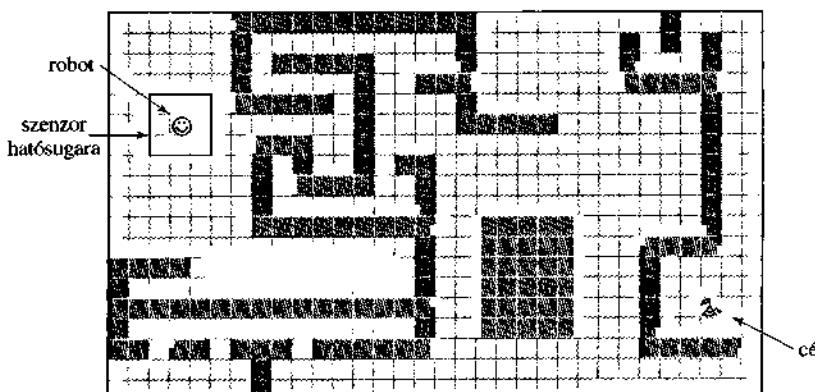


25.25. ábra. Diagram a 25.5. feladathoz

lapos akadály behelyezése egy 2 szabadságfokos robot esetén? Adjon példát rá, és indokolja meg, miért nem lehetséges nagyobb számú független terület kialakítása. Mi a helyzet, ha nem lapos az akadály?

**25.6.** Vegyük a 25.26. ábra egyszerűsített robotját. Tegyük fel, hogy a robot derékszögű koordinátái ismertek minden időpillanatban, mint ahogy a célhelyzet is. Ugyanakkor van egy akadály, amelynek a helyzete nem ismert. A robot csak a közvetlen környezetében képes érzékelni az akadályt, mint ahogy azt az ábra is illusztrálja. Az egyszerűség kedvéért tételezzük fel, hogy a robot mozgása zajjal nem terhelt, és az állapottér diszkrét. A 25.26. ábra csak egy példa. Ebben a feladatban azt kérjük, hogy minden lehetséges rácsvilágra gondoljon, ahol van érvényes út a start pozíciótól a célig.

- Tervezzen egy modellalapú vezérlőt, amely garantálja, hogy a robot mindenig eléri a célt, amennyiben ez lehetséges. A modellalapú vezérlő egy, a mozgás során fokozatosan kialakított térkép formájában képes tárolni a terép jellemzőit. A robot az egyes mozdulatok között tetszőleges időt tölthet gondolkodással.
- Most pedig tervezzen egy *reaktív* vezérlőt ugyanerre a feladatra. Ez a vezérlő nem tárolhatja a korábbi érzékelések eredményeit. (Azaz nem készíthet térképet!) Ehelyett minden döntését az aktuális érzékelő adatai alapján hozza meg, amely magában foglalja a saját helyének és a cél pontos hely-



25.26. ábra. Egyszerűsített robot egy labirintusban. Lásd 25.6. feladat.

zetének ismeretét. A döntéshozási időnek függetlennek kell lennie a környezet méretétől, valamint a már megtett lépések számától. Maximum hányszor kell lépés a robotnak ahhoz, hogy elérje célját?

- (c) Hogy fog viselkedni az (a) és (b) pontban megtervezett vezérlő, ha az alábbi hat feltétel valamelyike fennáll: folytonos állapottér; zajjal terhelt érzékelés; zaj a mozgásban; zaj mindenkorban, mindenkorban az érzékelésben; a cél helyzete ismeretlen (csak akkor ismeri fel a célt, ha az az érzékelő hatótávolságán belülre kerül); vagy mozgó akadályok vannak. minden egyes feltételre és minden két vezérlőre adjon egy-egy példát, amikor is a robot elbukik (vagy magyarázza el, miért nem tud).
- 25.7. A 25.22. (b) ábrán kiterjesztett véges automatát használtunk a hatlábusú robot egyetlen lábának vezérléséhez. Ebben a feladatban a cél egy olyan AFSM tervezése, amely a hat külön lábhoz tartozó vezérlővel együttműködve hatékony és stabil mozgást eredményez. Ennek eléréséhez ki kell bővíteni az egyes lábakat vezérlő AFSM-eket, hogy üzeneteket is küldjenek az új AFSM-nek, és hogy várjanak amíg más üzenetek érkeznek. Magyarázza el, hogy miért hatékony a vezérlője, azaz miért nem fogyászt feleslegesen energiát (pl. csíszó lábak miatt), és hogy miért mozgatja a robotot megfelelően nagy sebességgel! Bizonyítsa be, hogy vezérlője megfelel a 25.2. alfejezetben támasztott stabilitási feltételeknek.
- 25.8. (Ezt a feladatot Michael Genesereth és Nils Nilsson találták ki, és az első évesektől a végzőskig mindenkihez szól.) Az emberek annyira ügyesek alapfeladatok elvégzésében – mint például csészék felemelése vagy dobozok egymásra rakása –, hogy gyakran elfelejtik milyen összetettek ezek a mozdulatok valójában. A feladat során felfedezzük e feladatok komplexitását, és összefoglaljuk a robotika elmúlt 30 évének fejlődését. Először határozzuk meg a feladatot (például egy kapu készítése három dobozból), majd építünk egy robotot négy emberből az alábbiak szerint:
- **Agy:** az Agy feladata a cél elérését biztosító terv kidolgozása és a kezek irányítása a terv megvalósítása során. Az Agy információt csak a Szemtől

kaphat, *közvetlenül nem láthatja a helyszínt!* Az Agy az egyetlen, aki tudja, mi a cél.

- **Szem:** a Szem feladata a helyszín rövid leírása az Agy számára. A munkakörnyezettől kb. egy méternyire kell lennie, és vagy kvalitatív leírást (mint például: „Az oldalán fekvő zöld doboz tetején egy piros doboz van”), vagy kvantitatív leírást („A zöld doboz kb. 50 cm távolságra balra van a kék hengertől”) adhat. A Szem az Agytól jövő kérdésekre is válaszolhat, így például: „Van-e hely a Bal Kezem és a piros doboz között?” Ha van kéznél egy videokamera, akkor állítsuk rá a munkakörnyezetre, és a Szem ezen keresztül nézze azt, ne lássa közvetlenül!
- **Kezek (Jobb és Bal):** minden Kéz más ember játssza. A két Kéz egymás mellett áll. A Balkéz csak a saját balkezét használja, a Jobbkéz pedig csak a jobbat. A Kezek csak egyszerű, az Agytól érkező feladatokat hajthatnak végre. Például: „A Balkéz mozduljon 5 cm-t előre”. Csak mozgásparancsot képesek végrehajtani – például a „vedd fel a dobozt” nem olyan parancs, amit a Kéz végre tud hajtani. A csalások megakadályozása végett megkövetelhetjük, hogy a kezeken legyen kesztyű vagy használjanak csipeszt. A Kezeknek *bekötött szemmel kell tevékenykedniük*. Egyetlen érzékelési képességük az, hogy meg tudják mondani, amikor útjukat valamilyen elmozdítatlan akadály, például egy asztal vagy a másik Kéz zárja el. Ilyen esetben hangjelzést adhatnak az Agy informálása céljából.

# VIII. RÉSZ

# KONKLÚZIÓK

---

## 26. FILOZÓFIAI ALAPOK

Ebben a fejezetben arról esik szó, hogy mit jelent gondolkozni, vajon képesek-e erre a mesterséges tárgyak, és szeretnénk-e, ha képesek lennének rá.

Ahogyan az 1. fejezetben említettük, a filozófusok sokkal régebb óta vannak jelen, mint a számítógépek, és folyamatosan próbálnak válaszokat találni néhány olyan kérdésre, amely a mesterséges intelligenciához kötődik: Hogyan működik az elme? Képesek-e gépek intelligensen cselekedni; és ha igen, lenne-e elmélük? Milyen etikai következményei lehetnek az intelligens gépeknél? A könyv eddigi huszonöt fejezetében a mesterséges intelligencia saját kérdéseiről volt szó, most egy fejezet erejéig a filozófusok témaival foglalkozunk.

Először néhány elnevezés: a filozófiában **gyenge MI-hipotézisnek (weak AI)** nevezik azt az állítást, miszerint a gépek valószínűleg képesek intelligensen cselekedni (vagy jobban mondva, képesek úgy cselekedni, *mintha* intelligensek lennének), míg azt az állítást, hogy a gépek *valóban* intelligensen cselekszenek, **erős MI-hipotézisnek (strong AI)** hívják.

A legtöbb MI-kutató elfogadja a gyenge MI-hipotézist, az erőssel pedig nem törődik: amíg a program működik, az nem érdekes, hogy az intelligencia szimulációjának nevezik-e vagy valós intelligenciának. minden MI-kutatónak foglalkoznia kellene munkája etikai következményeivel.

### 26.1. GYENGE MI: TUDNAK-E A GÉPEK INTELLIGENSEN CSELEKEDNI?

Néhány filozófus megpróbálta bebizonyítani, hogy a mesterséges intelligencia nem lehetséges, azaz a gépek valószínűleg nem tudnak intelligensen cselekedni. Néhányan az MI-kutatás leállítását szeretnék elérni érveikkel:

*A számításorientált megközelítés kultuszának részeként üzött mesterséges intelligenciának még esélye sincsen arra, hogy tartós eredményeket érjen el (...) itt az ideje, hogy az MI-kutatók erőfeszítéseit – és a támogatásukra szánt hatalmas pénzeket – máshova tereljük, mint a számításorientált megközelítés (Sayre, 1993).*

A mesterséges intelligencia lehetséges mivolta nyilván azon múlik, miként definiáljuk. Lényegét tekintve a mesterséges intelligencia egy adott architektúrához a legjobb ágensprogramot keresi. Ebben a megfogalmazásban már a definíció szerint is lehetséges az MI: bármely  $k$  tárbitet tartalmazó digitális architektúrához pontosan  $2^k$  ágensprogram tartozik; ahhoz pedig, hogy megtaláljuk a legjobbat, egyszerűen csak egyesével fel kell

sorolni és tesztelni az ágenseket. Ez ugyan nem lenne praktikus nagy *k* esetén, de a filozófia az elméettel, nem pedig a gyakorlattal foglalkozik.

A mesterséges intelligenciára adott definícióink jól működik a mérnöki probléma esetében is, ahol egy adott architektúrához tartozó jó ágenst kell találni. Nagy a kísértés tehát, hogy befejezzük itt ezt a szakaszt, és igenlő választ adjunk a címben feltett kérdésre. A filozófusok azonban két architektúrát akarnak összehasonlítni: az embert és a gépet. Ráadásul a hagyományosan feltett kérdés így hangzik: „**Tudnak-e a gépek gondolkodni?**” Sajnos ez egy helytelenül feltett kérdés. Hogy megértsük, miért, nézzük az alábbi kérdéseket:

- Tudnak a gépek repülni?
- Tudnak a gépek úszni?

A legtöbb angolul beszélő ember<sup>1</sup> egyetért abban, hogy az első kérdésre igennel kell felelni: a repülőgépek tudnak repülni; de a második kérdésre az angolban a válasz nemleges: a hajók és a tengeralattjárók haladnak ugyan a vízben, de az angolban ezt nem nevezik úszásnak. Sem ezeknek a kérdéseknek persze, sem a válaszoknak nincsen semmilyen hatása a tengerészek, a tengerészeti mérnökök vagy az ezekkel a dolgokkal bármilyen kapcsolatba kerülők minden nap munkájára. A válaszoknak alig van közük a repülőgépek és tengeralattjárók tervezéséhez és képességeihez; inkább arról szólnak, hogy a szavak használatának milyen módjait választottuk ki. Úgy alakult, hogy az „úszni” szó az angolban azt jelenti, hogy ‘testrészeinek mozgásával előrehalad a vízben’, míg a „repülni” szó jelentése az angolban nem határozza meg a helyváltoztatás módját.<sup>2</sup> A „gondolkodó gépek” gyakorlati lehetősége még csak mintegy ötven éve jelent meg, és ez az idő nem volt elég az angolul beszélőknek, hogy a „gondolkodni” szó az angolban új jelentést kapjon.

Alan Turing azt javasolta *Computing Machinery and Intelligence* c. híres cikkében (Turing, 1950), hogy ne azt kérdezzük, tudnak-e a gépek gondolkodni, hanem azt vizsgáljuk, hogy átmennek-e a gépek egy viselkedési intelligenciateszten, amelyet később Turing-tesztnak neveztek el. A teszt szerint a programnak öt percen át kell (gépeltek online üzenetekkel) beszélgetnie egy kérdezővel. A kérdezőnek ezután választania kell, hogy egy programmal vagy egy személlyel beszélgetett-e, és egy program akkor felel meg a teszten, ha az idő 30%-ában megtéveszti a kérdezőt. Turing azt a sejtést fogalmazta meg, hogy 2000-re egy  $10^9$  táregységből álló számítógépet be lehet úgy programozni, hogy megfeleljön a teszten, ámde nem lett igaza. Az ugyan megtörtént már, hogy néhány embert öt percig becsaptak; például az ELIZA program és az MGONZ internethoz kötött csevegőrobot becsapott olyan embereket, akik nem gondoltak arra, hogy talán egy programmal beszélhetnek, és az ALICE program becsapott egy zsűritagot a 2001-es Loebner-díjért zajló versenyben. De egyetlenegy program sem került annak közelébe, hogy 30%-ot érjen el képzett zsűrivel szemben, és az MI-kutatás egésze nem is szentel túl nagy figyelmet a Turing-teszteknek.

Turing az intelligens gépek lehetősége ellen felhozható érvek széles választékát is megvizsgálta, beleértve szinte minden olyan ellenérvet, amely a cikk megjelenése óta eltelt fél évszázadban felmerült. A következőkben ezek közül nézünk meg néhányat.

<sup>1</sup> A kérdések eredeti formája és így az elemzés tárgya az angol nyelv. (A ford.)

<sup>2</sup> Az orosz nyelvben az „úszni” megfelelője hajókra is vonatkozik. A magyar „úszni” ige is vonatkozhat hajókra, sőt, ismét csak az angoltól eltérően, még felhőkre is. (A ford.)

## A képesség hiányából vett érv

A „képesség hiányából vett érv” azzal a követeléssel lép fel, hogy „a gépek sohasem képesek *X*-et tenni”. Az *X*-ekre Turing a következő példákat hozza:

Kedvesnek, találékonynak, szépnek és barátságosnak lenni, kezdeményezni, humorérzéket mutatni, megkülböztetni a helyest a helytelentől, hibákat követni el, szerelmesnek lenni, élvezni a tejzsínes épret, elérni, hogy valaki szerelmes legyen belé, tapasztalatból tanulni, helyesen használni a szavakat, saját gondolatai alanyának lenni, annyira változatosan viselkedni, mint az ember, valami igazán újat tenni.

Turingnak az intuícióját kellett használnia, hogy megsejtse, mi válhat lehetséges a jövőben, mi azonban abban a kényelmes helyzetben vagyunk, hogy visszanézhetünk a számítógépek eddigi teljesítményére. Senki sem tagadja, hogy a számítógépek ma sok olyan dolgot megtesznek, amely korábban kizártlag az emberekhez tartozott. A programok sakkoznak, dámát vagy más játékokat játszanak, alkatrészeket vizsgálnak gyártósorokon, a szövegszerkesztéskor ellenőrzik a helyesírást, helikoptereket és autókat irányítanak, betegségeket diagnosztizálnak és több száz más tevékenységet végeznek éppolyan jól, vagy még jobban, mint az emberek. A számítógépek kicsiny, de jelentős felfedezéseket tettek a csillagászatban, a matematikában, a kémiában, az ásványtanban, a biológiában, a számítástudományban és más területeken. Ezekhez a felfedezésekhez az emberi szakértőhöz hasonló szintű teljesítményekre volt szükség.

Annak fényében, amit most tudunk a számítógépekről, nem meglepő, hogy jól teljesítenek a sakkhoz hasonló kombinatorikai problémákban. De az algoritmusok az emberhez mérhető szinten hajtanak végre olyan tevékenységeket is, amelyek látszólag emberi döntést igényelnek, amelyek során, mint Turing mondta, képesnek kell lenni a „tapasztalatból tanulni” és „megkülböztetni a helyest a helytelentől”. Még egészen korán, 1955-ben Paul Meehl (lásd még Grove és Meehl, 1996) képzett szakértők döntéshozatali folyamatát tanulmányozta olyan szubjektív esetekben, mint például hogy egy diákok elvégez-e egy képzési programot, vagy egy bűnöző visszaesik-e. A megvizsgált 20 eset közül 19-nél Meehl azt találta, hogy egyszerű statisztikai tanulási algoritmusok (mint például a lineáris regresszió vagy a naiv Bayes-tanulás) jobb előrejelzéseket adnak, mint az emberi szakértők. Az Educational Testing Service a GMAT-vizsga esszékérdéseinek millióit osztályozza 1999 óta egy automatizált programmal. A program osztályzata az osztályzást végző emberek minősítésével az esetek 97%-ában egyezik meg, és ez az arány hasonló ahhoz, amennyire a különböző emberek által adott minősítések meggyeznek (Burstein és társai, 2001).

Világos, hogy sok minden, köztük olyan dolgokat is, amelyekhez általános meggyőződésünk szerint komoly emberi élesemléjűség és értelelműszerűség, a számítógépek ugyanolyan jól el tudnak végezni, mint az emberek, vagy akár még jobban is. Ez persze nem jelenti azt, hogy a számítógépek élesemléjűséget és értelmet tanúsítanának az ilyen dolgok végrehajtásakor: ezek nem részei viselkedésüknek (erről a kérdésről majd másolat beszélünk), hanem inkább arról van szó, hogy gyakran téves az első feltételezésünk az adott tevékenység elvégzéséhez szükséges mentális folyamatról. Ugyanakkor az is igaz, hogy számtalan feladatban a számítógépek (finoman szólva) nem járnak az ellen, és ilyen Turing feladata, a nyíltvégű beszélgetések folytatása is.

## A matematikai ellenvetés

Turing (Turing, 1936) és Gödel (Gödel, 1931) munkássága révén közismert, hogy egyes matematikai kérdések elviében is megválaszolhatatlanok bizonyos formális rendszerekben. A leghíresebb példa erre Gödel nemteljességi tétele (lásd 9.5. alfejezet). Ez, röviden összefoglalva, azt mondja ki, hogy bármely  $F$  formális rendszerben, amelyben az aritmetika megfogalmazható, lehetséges egy  $G(F)$  Gödel-mondatot konstruálni, amelyre igaz, hogy

- $G(F)$  egy mondat  $F$ -ben, de  $F$ -en belül nem bizonyítható.
- Ha  $F$  konzisztens, akkor  $G(F)$  igaz.

Egyes filozófusok, mint például J. R. Lucas (Lucas, 1961) állítása szerint ez a tétel azt bizonyítja, hogy a gépek mentálisan alsóbbrendűek az embereknél, mert a gépeket mint formális rendszereket korlátozza a nemteljességi tételes (nem tudják megállapítani a saját Gödel-mondatuk igazságát), az embereket viszont ilyesmi nem korlátozza. Ez az állítás évtizedes, sok könyvet felölélő vitát keltett, például a matematikus Sir Roger Penrose két könyvében (Penrose, 1989; 1994) képviseli ezt az állítást kicsit megcsavarva (például azzal a feltételezéssel, hogy az emberek azért különböznek, mert agyukat a kvantumgravitáció működteti). Három problémát fogunk az állítással kapcsolatban megvizsgálni.

Először is Gödel nemteljességi tétele csak olyan formális rendszerekre vonatkozik, amelyek elég erősek az aritmetikához. Ezek közé tartozik a Turing-gép is, és Lucas állítása részben azon a feltételezésen alapul, hogy a számítógépek Turing-gépek. Ez jó közelítés, ám nem teljesen igaz. A Turing-gépek végtelenek, a számítógépek azonban végesek, és ezért bármely számítógépet le lehet írni egy nagyon nagy logikai állítás-rendszerrel, amelyre már nem vonatkozik Gödel nemteljességi tétele.

Másodszor is egy ágensnek nem kell túlzottan szégyenkeznie amiatt, hogy egyes állítások igazságát, másoktól eltérően, ő nem tudja eldönthetni. Nézzük például a következő mondatot:

J. R. Lucas nem tudja konzisztensen azt állítani, hogy ez a mondat igaz.

Lucas ellentmondana önmagának, ha azt állítaná, hogy a fenti (teljes) mondat igaz, tehát Lucas nem tudja ezt a mondatot konzisztensen állítani, tehát a mondat mindenképpen igaz. (A mondat nem lehet hamis, mert ha hamis volna, akkor Lucas nem tudná konzisztensen állítani, tehát igaz lenne.) Bebizonyítottuk tehát, hogy van egy mondat, amelyet Lucas nem tud konzisztensen állítani, mások viszont (beleértve a gépeket is) igen. De ez előttünk semmit sem von le Lucas értékéből. Egyetlenegy ember sem tudja, hogy egy másik példát vegyünk, élete során kiszámítani tízmilliárd tízjegyű szám összegét, egy számítógép viszont másodpercek alatt megteszí ezt. Mégsem tartjuk ezt az emberi gondolkodási képesség alapvető korlátjának. Évezredeken át, mielőtt felfedezték volna a matematikát, az emberek ugyanúgy intelligens viselkedést tanúsítottak, nem valószínű hát, hogy a matematika a periferiálisnál fontosabb szerepet játszana az intelligencia meghatározásában.

Harmadszor és legfőképpen, még ha el is fogadjuk, hogy a számítógépek bizonyítási képessége korlátozott, semmi sem bizonyítja, hogy az emberek mentesek lennének az ilyen korlátoztásuktól. Túlzottan is egyszerű szigorúan bebizonyítani, hogy egy for-

mális rendszer képtelen  $X$ -re, majd minden további bizonyíték nélkül azt állítani, hogy az emberek a maguk nem formális módján képesek erre az  $X$ -re. Azt, hogy az emberek nincsenek alávetve Gödel nemteljességi tételenek, valóban lehetetlen bebizonyítani, mert bármely szigorú bizonyításnak tartalmaznia kellene az állítás szerint formalizálhatatlan emberi tehetség egy formalizálását, tehát a bizonyítás önmagát cífolná meg. Lehet még fellebbezni az intuícióhoz, miszerint az emberek valamilyen módon képesek a matematikai belátás emberfeletti cselekedeteire. Ezt a fordulatot fejezik ki az olyan érvek, mint például „feltételezünk kell önnön konzisztenciánkat, hogy a gondolkodás egyáltalán lehetséges legyen” (Lucas, 1976). De ha bármiről is, akkor az emberi gondolkodásról mindenképpen tudjuk, hogy inkonzisztens. Ez egyértelműen igaz a minden nap érvélésünkre, de vonatkozik még az elővigyázatos matematikai gondolkodásra is. Híres példa erre a négyzsín-probléma. 1879-ben Alfred Kempe közzétett egy bizonyítást, amelyet széles körben elfogadtak, és amely hozzájárult, hogy a Royal Society tagjává választották. 1890-ben azonban Percy Heawood egy hibát fedezett fel a bizonyításban, és egészen 1977-ig senki nem tudta a tértelt bebizonyítani.

## A meghatározatlanságból vett érv

A mesterséges intelligencia vállalkozásának legnagyobb hatású és legmakacsabb kritikáját Turing úgy mutatta be, mint „a viselkedés meghatározatlanságából származó érvet”. Az állítás lényege az, hogy az emberi viselkedés túl komplex ahhoz, hogy egy szabálykészlettel le lehessen írni, és mivel a számítógépek semmi másra nem képesek, csak szabályok követésére, ezért nem tudnak az emberekhez hasonló intelligens viselkedést létrehozni. Azt, hogy logikai szabályok egy halmzával képelség minden leírni, a mesterséges intelligenciában kvalifikációs problémának (*qualification problem*) nevezik (lásd 10. fejezet).

Ezt a nézetet főként a filozófus Hubert Dreyfus képviselte, aki számos befolyásos MI-kritika szerzője: *What Computers Can't Do* (Dreyfus, 1972); *What Computers Still Can't Do* (Dreyfus, 1992), valamint a testvérével, Stuarttal, közösen írt *Mind Over Machine* (Dreyfus és Dreyfus, 1986).

Azt az álláspontot, amit kritizáltak, Haugeland (Haugeland, 1985) nyomán elkezdték „Jófajta, régvágású MI”-nek nevezni (angol rövidítéssel: GOFAI). A GOFAI azt feltételezné, hogy minden intelligens viselkedés megragadható egy olyan rendszerrel, amely a tárgyterületet leíró tények és szabályok halmozaiából kiindulva logikai következetést végez. Ezért a GOFAI a 7. fejezetben leírt legegyszerűbb logikai ágensnek felel meg. Dreyfusnak igaza van abban, hogy a logikai ágensek ki vannak téve a kvalifikációs probléma veszélyének. Amint a 13. fejezetben láttuk, a nyílt tárgyterületekhez sokkalta megfelelőbbek a valószínűségi következtető rendszerek. Dreyfus kritikája tehát nem *per se* a számítógépek, hanem csak a programozás egy meghatározott módja ellen irányul. Okkal feltételezhetjük persze, hogy nem lenne túl nagy hatású egy könny *Amire az elsőrendű logikai szabályalapú, nem tanuló rendszerek képtelenek* címmel.

Dreyfus nézete szerint az emberi szakértelemnek ugyan része néhány szabály ismerte, de ezek csak „holistikus kontextust” vagy „háttérét” alkotnak az emberi tevékenységekhez. Példaként az ajándék adásakor és fogadásakor illő társas viselkedést hozza fel: „Általában az ember a megfelelő körülmények között úgy reagál, hogy egy megfelelő

ajándékot ad.” Az embernek láthatólag „közvetlen érzéke van ahhoz, hogy miként törtennek a dolgok, és mit lehet elvární.” Hasonló állítást tehetünk a sakkjáték vonatkozásában is: „Egy átlagos sakkmesternek lehet, hogy gondolkodnia kell azon, mit tegyen, de egy nagymester látja, hogy a tábla egy bizonyos lépései követel (...) a helyes reakció egyszerűen csak beugrik a fejébe.” minden bizonnal igaz, hogy egy ajándékozó vagy egy sakknagymester gondolati folyamata azon a szinten zajlik, amely rejtegett a tudatos elme önismerete elől. Ez azonban nem jelenti, hogy ez a gondolati folyamat nem létezik. Lényeges, hogy Dreyfus nem válaszolja meg azt, hogy a helyes lépés *miként* kerül a nagymester fejébe. Mindez Daniel Dennett (Dennett, 1984) megjegyzésére emlékezet:

Ez olyan, mintha a filozófusok azt tartanák magukról, hogy szakértő magyarázói a színpadi bűvészek trükkjeinek, aztán, amikor megkérdezzük tőlük, hogy milyen trükköt használnak a bűvész a nő kettéfűrészelésénél, akkor azt a magyarázatot adják, hogy minden nagyon is világos: a bűvész nem fűrészeli ketté a nőt, ezt csak színleli. Ha azt kérdezzük azonban: „Mégis *hogyan* színleli?”, a filozófusok ezt felelik: „Nem ránk tartozik.”

A két Dreyfus (Dreyfus és Dreyfus, 1986) vázolja a szakértelem ötlépcsős elsajátításának folyamatát, kiindulva a hasonló szabályalapú feldolgozásból (ahogyan az a GOFAI-ban is szerepel) eljutva a helyes reakció azonnali kiválasztásának képességéhez. A szerzők ezzel a javaslattal tulajdonképpen az MI kritikusaiból az MI elméletalkotótává válnak: egy „eset-könyvtárakba” szervezett neurális háló-architektúrát vázolnak fel, azonban rámutatnak néhány problémára is. Ezeket a problémákat szerencsére már mind célba vette az MI-kutatás néhányukat részleges, másokat pedig teljes sikerkel oldva meg. Nézzünk néhány ilyen problémát:

1. Háttérudás nélkül nem lehet jól példákból általánosítani. Azt állítják, senki sem tudja, miként lehetne a háttérudást beépíteni a neuronhálók tanítási folyamatába. Pedig, amint a 19. fejezetben mi is láttuk, léteznek módszerek az a priori tudás felhasználására a tanítási folyamat során. Ezek a módszerek persze a tudás explicit megadásán alapulnak, amelynek lehetőségeit a szerzőpáros elszántan tagadja. Nézetünk szerint ez a probléma erős indok a neurális feldolgozás jelenlegi modelljeinek olyan újratervezésére, amely már a többi tanuló algoritmushoz hasonlóan *képes lesz* felhasználni a korábban megtanult tudást.
2. A neurális hálók tanulása a felügyelt tanulás egyik formája (lásd 18. fejezet), amely igényli, hogy előzetesen azonosítuk a releváns bemeneteket, valamint a megfelelő kimeneteket. Ezért azt állítják, hogy nem is képesek ezek a hálózatok autonóm módon, emberi betanítás nélkül üzemelni. Valójában a nem ellenőrzött tanulás (*unsupervised learning*) (lásd 20. fejezet) és a megerősítéssel tanulás (*reinforcement learning*) (lásd 21. fejezet) éppen a tanító nélküli tanulást teszi lehetővé.
3. A tanuló algoritmusok nem teljesítenek jól sok jellemző esetén, viszont ha kijelöljük a jellemzők egy halmazát, „nem ismeretes olyan módszer, amellyel új jellemzőket illeszthetünk be, ha a jelenlegiek nem lennének elegendők a megtanult tényekhez”, valójában új eljárások, mint például a szupport vektor gépek, jól megbirkóznak a nagy jellemzőhalmazokkal is. A 19. fejezetben pedig láthattuk, hogy alapjaikat tekintve léteznek módszerek az új jellemzők bevezetésére, habár még sok tennivaló akad ezen a területen.

4. Az agy képes a szenzorait releváns információk keresésére irányítani, és képes kinyerni az információkból azt, ami releváns az adott szituációban. Ezzel szemben „jelenleg egyetlen részletét sem értjük ennek a működésmódnak – állítják –, és még csak olyan hipotézist sem tudunk alkotni róluk, amely útmutatásul szolgálhatna az MI-kutatóknak”. Valójában az aktív látás területe, amelyet az információérték-elmélet (lásd 16. fejezet) alapoz meg, éppen hogy a szenzorok irányításának problémájával foglalkozik, és az elérte elméleti eredmények egy részét már át is ültették robotmegvalósításokba.

Összegezve azt mondhatjuk, hogy számos olyan kérdés, melyet Dreyfus felvetett – a köznapi háttérkérdés, a kvalifikáció probléma, a bizonytalanság kezelése, a tanulás, a döntéshozatal előre lefordított módjai, annak fontossága, hogy ne testetlen következetetőgépeket tekintsünk, hanem szituációjukba ágyazott ágenseket –, ma már bekerült az intelligens ágensek tervezésének szokványos menetébe. Véleményünk szerint minden tehát nem a mesterséges intelligencia lehetetlenségét, hanem éppen a fejlődését bizonyítja.

## 26.2. ERŐS MI: TUDNAK-E TÉNYLEGESEN GONDOLKODNI A GÉPEK?

Több filozófus álláspontja szerint egy Turing-teszten átmenő gép nem gondolkodna ténylegesen, csupán szimulálná a gondolkodást. Ezt az ellenvetést is előre láta Turing. Geoffrey Jefferson (Jefferson, 1949) professzor egy beszédét idézi:

Amíg nem képes egy gép egy szonettet írni vagy egy zenei versenyművet komponálni, pusztán a gondolatai és az érzései alapján, és nem szimbólumok véletlen alakulása folytán, addig nem fogunk beleegyezni, hogy egy gép egyenlő lehet az aggyal; ehhez az kell, hogy ne csupán megírja, de tudja is, hogy megríta.

Ezt nevezi Turing **tudatossági** (*consciousness*) érvnek: a gépnek saját mentális állapotainak és cselekvéseinek tudatában kell lennie. Ugyan a tudatosság fontos témakör, Jefferson megállapítása azonban lényegileg a **fenomenológiához** (*phenomenology*), a közvetlen tapasztalat tudományához kötődik: a gépnek ténylegesen éreznie kell az érzéseit. Más szerzők az **intencionalitásra** (*intentionality*) koncentrálnak, ami azt kérdezi, hogy a gép feltételezett hiedelmei, vágyai és más reprezentációi ténylegesen valamilyen valósvilág-beli dologra „vonatkoznak”-e.

Figyelemre méltó Turing válasza erre az ellenvetésre. Felhozhatott volna érveket amellett, hogy a gépek valóban rendelkezhetnek tudattal (avagy viszonyulhatnak fenomenológiai, illetve viselkedhetnek intencionálisan). Ehelyett Turing úgy érvel, hogy ez a kérdés éppen úgy rosszul lett megfogalmazva, mint a „Tudnak-e a gépek gondolkodni?” kérdés. Ráadásul miért kellene erősebb kritériumokat várni el a gépek esetében, mint az embereknél? A minden nap életben végső soron nincsen *semmilyen* közvetlen bizonyítékunk a többi ember belső mentális állapotairól. Mégis, amint Turing rámutat, „ahelyett hogy állandóan erről vitatkoznánk, azzal az **udvarias feltételezéssel** (*polite convention*) élünk, hogy mindenki képes gondolkodni”.

Azonban Turing szerint Jefferson hajlandó lenne ezt az udvarias feltételezést a gépekre is kiterjeszteni, ha találkozna intelligensen cselekvő gépekkel. Az alábbi dialógust idézi, amely annyira beépült a mesterséges intelligencia szájhagyományába, hogy egyszerűen nem hagyhattuk ki:

**EMBER:** Szonettel első sorában, mely úgy hangzik, hogy „hasonlítanálak téged a nyári naphoz”, nem lenne legalább ugyanolyan jó, ha „tavaszi nap” szerepelne?

**GÉP:** Nem jönne ki az ítem.

**EMBER:** De ha „téli nap” lenne? Ez ütemre ugyanolyan jó.

**GÉP:** Persze, de hát senki sem szeretné, ha egy téli naphoz hasonlítanák.

**EMBER:** Mondanál olyat, hogy Pickwick úr a karácsonyra emlékeztet téged?

**GÉP:** Bizonyos értelemben igen.

**EMBER:** A karácsony viszont egy téli nap, és én nem gondolom, hogy Pickwick úrnak ellenére lenne az összehasonlítás.

**GÉP:** Nem hiszem, hogy ezt komolyan gondolod. A téli nap alatt egy átlagos téli napot szokás érteni, nem pedig egy olyan különlegeset, mint a karácsony.

Azt ugyan elismeri Turing, hogy a tudatosság megmagyarázása nehéz kérdés, de tagadja, hogy bármi jelentősége is volna a mesterséges intelligencia gyakorlata számára: „Nem szeretném azt a benyomást kelteni, miszerint úgy gondolnám, hogy a tudatosságban ne volna semmi rejtelyes elem. (...) De nem gondolom azt, hogy ezeket a rejtelyleket meg kellene oldani ahhoz, hogy meg tudjuk válaszolni a cikkben tárgyalt kérdéseket.” Egyetérthetünk Turinggal, hiszen az a fontos számunkra, hogy intelligensen viselkedő programokat készítsünk, nem pedig az, hogy mások azt vajon szimulált vagy valóságos intelligenciának tartják. Persze másfelől sok filozófusnak igencsak fontos ez a kérdés. Ezt megvilágítandó vegyük szemügyre, hogy más eseteken mennyire tartják valóságosnak az ember készítette tárgyat.

A mesterséges karbamidot 1828-ban sikerült először előállítania Fredrick Wöhlernek. A szintézis fontosságát az adta, hogy bizonyítékot szolgáltatott a szerves és szervetlen kémia akkortájt sokak által vitatott egységre. A vegyület előállítása után a kémikusok egyetértettek abban, hogy a mesterséges karbamid valódi karbamid, hiszen az összes megfelelő fizikai jellemzőben megegyeznek. Ehhez hasonlít, hogy a mesterséges édesítőket ugyanúgy édesítőknek nevezzük, és a mesterséges megtermékenyítés (a mesterséges intelligencia másik módja) ugyanolyan megtermékenyítés. Azonban a mesterséges virágok nem virágok, és, amint Daniel Dennett rámutat, a mesterséges Chateau Latour bor sem lenne a különleges Chateau Latour francia bor, még akkor sem, ha kémiailag megkülönböztethetetlen volna, mivel nem a rögzített helyen (a francia Bordeaux-i borvidéken) és nem a rögzített módon készült. Ugyanígy egy mesterséges Picasso-festmény – akárhogyan is néz ki – nem volna igazi Picasso-festmény.

Megállapíthatjuk tehát, hogy bizonyos esetekben az ember készítette tárgy viselkedése fontos, más esetekben pedig a származása. Úgy látszik, megállapodás kérdése, hogy melyik esetben melyiket tartjuk fontosnak. Mesterséges elmék esetében nincsen ilyen megállapodás, így az intuícióinkra kell hagyatkoznunk. Egy filozófus, John Searle (Searle, 1980), elég határozott intuícióval rendelkezik:

Senki sem gondolja, hogy egy vihar számítógépes szimulációjától elázánánk. (...) Mi visz rá tehát embereket annak feltételezésére, hogy a mentális folyamatok számítógépes szimulációja mentális folyamatokat mutatna fel? (37–38. oldal)

Könnyen elfogadható persze, hogy a viharok számítógépes szimulációi senkit sem áztatnak el, az azonban kevésbé látható, miként lenne alkalmazható ez az analógia a mentális folyamatok számítógépes szimulációira. Hiszen amikor Hollywoodban locsolók és szélgépek segítségével szimulálják a vihart, eláznak a színészek. A legtöbben határo-

zottan összeadásnak tartják az összeadás számítógépes szimulációját, és a sakkjáték számítógépes szimulációja is sakkjáték. A mentális folyamatok vajon a viharokhoz hasonlíthatók-e, vagy inkább az összeadáshoz és a sakkhoz? Olyanok, mint a Chateau Latour bor és Picasso, vagy mint a karbamid? Ez teljes egészében azon műlik, hogy milyen elméletet vallunk a mentális állapotokról és folyamatokról.

A **funkcionalizmus (functionalism)** szerint a mentális állapotok köztes kauzális feltételek a bemenetek és a kimenetek között. A funkcionalista elmélet szerint ha két rendszer izomorf kauzális folyamatokkal rendelkezik, akkor ugyanazok a mentális állapotai is. Egy számítógépes programnak tehát ugyanolyan mentális állapotai lehetnek, mint egy embernek. Azt persze még nem határoztuk meg, hogy az „izomorf” pontosan mit jelent, de azt tételezzük fel, hogy létezik egy olyan absztrakciós szint, ami fölött nem szármíti a specifikus implementációt; és ha a folyamatok eddig a szintig izomorfak, akkor ugyanazok lesznek a mentális állapotok is.

Ezzel szemben a **biológiai naturalizmus (biological naturalism)** szerint a mentális állapotok magas szintű emergens tulajdonságok, amelyeket a *neuronok* alacsony szintű neurológiai folyamatai okoznak, azaz a neuronok (meghatározatlan) tulajdonságai a lényegesek. A mentális állapotokat így tehát nem lehet duplikálni valamilyen programmal, amelynek azonos a bemenet–kimenet funkcionális struktúrája, hanem az is szükséges, hogy a program olyan architektúrán fusson, amelynek kauzális erői meggyeznek a neuronokéval. Az elmélet nem mond semmit arról, hogy a neuronok miért rendelkeznek ilyen kauzális erővel, és arról sem, hogy más fizikai megtétesülések rendelkezhetnek-e ezekkel az erőkkel.

A két álláspont megvizsgálásához vegyük szemügyre először az elmefilozófia leg-régebb problémáját, majd pedig három gondolatkísérletet.

## A test–elme probléma

A **test–elme probléma (mind–body problem)** azt kérdezi, hogy a mentális állapotok és folyamatok miként kapcsolódnak a testi (specifikusan az agyi) állapotokhoz és folyamatokhoz. Ezt a kérdést, mintha nem lenne elég nehéz, általánosítani fogjuk az „architektúra–elme” problémává, hogy az elmével rendelkező gépek lehetőségéről beszélhessünk.

Egyáltalán mi a probléma a test–elme problémában? Az első nehézség René Descartes-ra vezethető vissza, aki egy nem halandó lélek interakcióját vizsgálta egy halandó testtel, és egy **dualista (dualist)** elméletet állított fel, miszerint a lélek és a test két különböző típusú dolog. A **monista (monist)**, gyakran **materialista (materialism)** névvel is illetett elmélet szerint nincs olyan dolog, hogy halhatatlan lélek, hanem csupán az anyagi objektumok léteznek. Ennek megfelelően a mentális állapotok (mint például a fájdalom, a lovaglás tudata vagy az a meggyőződés, hogy Bécs Ausztria fővárosa) csupán agyállapotok. Ezt az elméletet John Searle egy tömör szlogenben foglalta össze: „Az agy okozza az elmét.”

A materialistának legalább két komoly ellenvetéssel kell szembesülnie. Az első a **szabad akarat (free will)** problémája: hogyan lehetséges, hogy egy tisztán fizikai elme, amelynek minden átalakulását szigorú fizikai törvények irányítják, mégis megörzi a döntés szabadságát? A legtöbb filozófus szerint ez a probléma ahelyett, hogy a materializmusnak jelentene kihívást, inkább a szabad akaratról alkotott naiv fogalmunk

gondos újraalkotását igényli. A második probléma a **tudatosság (consciousness)** általános témaköréhez kapcsolódik (a probléma hasonló, mint a **megértés (understanding)** és az **öntudat (self-awareness)** kérdései, habár nem teljesen ugyanarról van szó). A kérdés egyszerűen ez: miért van, hogy bizonyos agyállapotokkal rendelkezni valamiféle **érzés**, viszont másfajta fizikai állapotokkal rendelkezni (például szíklának lenni) feltehetően nem érzés.

Ahhoz, hogy elkezdhetünk megválaszolni a kérdést, az agyállapotok egy olyan megfogalmazását kell megtalálni, amely absztraktabb szintű, mint az egy adott személy agyát alkotó atomok adott időpontbeli fizikai elhelyezkedése. Például miközben Ausztria fővárosára gondolok, agyam minden pikoszekundumban milliárd apró változáson megy keresztül, mégis ezek a változások nem jelentik az agyállapot **kvalitatív** változását. Ennek megfelelően be kell vezetnünk az agyállapot **típusok** fogalmát, amelynek segítségével elkülöníthetők a különböző típusba tartozó agyállapotok. Az egyes szerzők különbözőképpen értik a *típus* szót ebben az esetben. A legtöbbük szerint, ha például veszünk egy agyat, és néhány szénatomját más szénatomra<sup>3</sup> cseréljük, a mentális állapot még nem változik meg. Ez azért örvendetes, mert valójában az agy atomjai az anyagcsere-folyamatok során állandóan cserélődnek, mégsem tűnik úgy, mintha ez nagy mentális felfordulást okozna.

Vegyük most szemügyre a mentális állapotok egy fajtáját, az **intencionális állapotoknak (intentional states)** is nevezett **ítéletlogikai attitűdöket (propositional attitudes)** (amelyekről már szó esett a 10. fejezetben). Ezek az állapotok – mint például a vélekedés, a tudás, a vágyszakás, a félelem és társai – olyanok, hogy bizonyos szempontból a külső világra utalnak. Az a vélekedés például, miszerint Bécs Ausztria fővárosa, egy bizonyos városra és annak státusára vonatkozó vélekedés. Azt fogjuk vizsgálni, hogy rendelkezhetnek-e a számítógépek intencionális állapotokkal, ezért hasznos áttekinteni ezen állapotok jellemzését. Azt is lehet mondani például, hogy az a mentális állapot, hogy egy hamburgerre vagyom, különbözik attól az állapottól, hogy egy pizzát szeretnék, hiszen a hamburger és a pizza a valódi világ különböző tárgyai. Másfelől viszont amellett érveltünk éppen néhány bekezdéssel korábban, hogy a mentális állapotok agyállapotok, tehát az azonosságukat meg kellene tudnunk határozni csupán „a fejen belül” maradva, minden hivatkozás nélkül a külső világra. A dilemma megoldásához vizsgálunk meg egy olyan gondolatkísérletet, amely megpróbálja az intencionális állapotokat elkülöníteni a hozzájuk tartozó külső tárgyaktól.

## Az „agy a tartályban” kísérlet

Képzeljük el – ha belebocsátkozunk ebbe a gondolatkísérletbe –, hogy az agyunkat születésünkkor kiveszik a testünkötő, és egy csodálatos szerkezetű tartályba helyezik. Ez a tartály táplálja az agyat, lehetővé téve, hogy növekedjen és fejlődjön. Ugyanakkor azonban egy teljességgel kitalált világ számítógépes szimulációjának elektromos ingereit vezetik az agyba, az agy motorikus ingereit pedig elfogják, és hatásukra megfelelően módosítják a szimulációt.<sup>4</sup> Ekkor az agy rendelkezhet a *Halálosan Vágik (Én, Hamburger)*

<sup>3</sup> Esetleg a szénatom más izotópjára, ahogyan ez az agyi képalkotó vizsgálatok során gyakran történik.

<sup>4</sup> Ez a helyzet ismerős lehet azoknak, akik látták az 1999-ben készült *Mátrix (The Matrix)* c. filmet.

mentális állapottal, akkor is, ha nincs is teste, amely az éhséget érezné, ízlelőbimbói sin-csenek, sőt talán egyáltalán nincs is hamburger a valódi világban. Azonosnak tekintjük-e ezt a mentális állapotot egy testben lévő agy megfelelő mentális állapotával?

Az egyik kiút a dilemmából annak megállapítása, hogy a mentális állapotok tartalma két különböző szempontból értelmezhető. A tág **tartalom** (*wide content*) nézet képviselői a mentális állapotok tartalmát a minden tudó külső megfigyelő szemszögéből értelmezik, amely megfigyelő átlátja az egész szituációt, és felismeri a világ különbsségeit. Tehát a tág tartalom nézet szerint a tartályba helyezett agy vélekedései különbözök a „normál” személy vélekedéseitől. A szűk **tartalom** (*narrow content*) nézet szerint viszont csak a belső, szubjektív nézőpontot kell figyelembe venni, és szerintük a két vélekedés megegyezik.

Annak a vélekedésnek, hogy a hamburger ízletes, benső természete van: van valami, amihez az ilyen vélekedés hasonlít. Most már beléptünk a **qualia**, a benső élmények birodalmába (a *qualia* szó a latin nyelvből származik, megközelítő jelentése: ’ilyen dolgok’). Tegyük fel, hogy a retina és az idegrendszer valamelyen sérülése folytán *X* személy úgy érzékel a vörös színt, mint *Y* a zöldet, és *vice versa*. Ebben az esetben ugyanúgy látják a közlekedési lámpát, és ugyanúgy is cselekszenek, de *tapasztalataik* valamiképpen különbözni fognak. Mindketten elfogadhatják, hogy tapasztalatuk neve: „a lámpa vörös”, de tapasztalatuk különböző érzés lesz. Nem világos, hogy mit jelent ez a mentális állapotok megegyezését illetően. A következőkben egy olyan gondolatkísérlettel foglalkozunk, amely azt vizsgálja, hogy az emberi neuronokon kívül más fizikai tárgyaknak is lehetnek-e mentális állapotai.

## Az agyprotézis-kísérlet

Az agyprotézis-kísérletet Clark Glymour vezette be a hetvenes évek közepén, majd John Searle (Searle, 1980) is finomított rajta, de manapság leginkább Hans Moravec (Moravec, 1988) munkásságával szokás összefüggésbe hozni. A kísérlet a következő. Tegyük fel, hogy a neurofiziológia eljutott odáig, hogy tökéletesen megértette az emberi agy összes neuronjának bemeneti–kimeneti jellemzőit és összekapcsolódását. Tegyük fel továbbá azt is, hogy képesek vagyunk olyan mikroszkopikus méretű elektronikus berendezéseket építeni, amelyek utánozni tudják ezeket a jellemzőket, és problémamentesen hozzáilleszthetők az idegszövethöz. Végül pedig tegyük fel azt is, hogy valamilyen csodálatos sebészeti technikával anélkül tudjuk az egyes neuronokat a megfelelő elektronikus eszközökkel helyettesíteni, hogy az agy egészében vett működése megszakadna. A kísérlet abból áll, hogy valaki fejében az összes neuront elektronikus eszközökkel helyettesítik, majd pedig a folyamat visszafordításával visszajuttatjuk az illetőt a normális biológiai állapotába.

A személynek mind a műtét során tanúsított külső viselkedése, mind belső tapasztalatai érdekelnek minket. A kísérlet alanyának külső viselkedésének definíció szerint változatlannak kell maradnia az operáció nélkül megfigyelhető viselkedéshez képest.<sup>5</sup> A kísérlet alanyának azonban egyben képesnek kell lennie arra, hogy – noha a tudatosság hiányát

<sup>5</sup> El tudunk képzelni egy azonos kontrollszemélyt, akiin egy placebooperációt hajtanak végre, hogy a két viselkedést összehozzék.

vagy jelenlétét egy harmadik személy nehezen állapíthatja meg – beszámolót adjon a tudatos tapasztalatát ért változásokról. A történtekre vonatkozó különböző intuícióink nyilván ellentmondanak egymásnak. Moravec szerint, aki robotkutató és funkcionalista, változatlan maradna a tudata. A filozófus és biológiai naturalista Searle azonban ugyanilyen erősen vallja, hogy a tudata eltünne:

Azt fogod észrevenni – legnagyobb megdöbbenedre –, hogy a külső viselkedés irányítása mintegy kicsúszik a kezeid közül. Azt veszed észre, hogy például amikor az orvosok vizsgálják a látásodat, hallod őket, amint azt mondják: „Egy piros tárgyat tartunk ön előtt, kérjük, mondja el, mit lát.” Ez a kiáltás törne ki belőled: „Semmit sem látok. Teljesen megvakulok.” De hallod a saját hangodat, amint befolyásodtól teljesen függetlenül ezt mondja: „Egy vörös tárgyat látok magam előtt.” (...) Tudatos tapasztalatod lassan semmivé zsugorodik, de a külső, megfigyelhető viselkedés változatlan marad. (Searle, 1992)

Mindannyian csak intuícióink alapján tudunk érvelni. Először vegyük észre, ahhoz, hogy a külső viselkedés változatlan maradjon, mialatt a kísérlet alanya fokozatosan elveszti a tudatát, az akaratnak egyik pillanatról a másikra hirtelen és teljesen kell eltünnie; különben a tudat zsugorodása befolyásolná a külső viselkedést – például ilyesmi szavak hangzanának el: „Segítség, eltűnőben vagyok!” Elég valószínűtlenül hangzik azonban, hogy a neuronok egyenkénti lecserélése az akarat hirtelen, egyik lépésről a másikra történő teljes eltűnésséhez vezetne.

Másodszor pedig nézzük meg, mi történik akkor, ha a kísérlet alanyát az alatt az idő alatt kérdezzük tudatos tapasztalatairól, amikor egyetlen igazi neuron sincs jelen az agyában. A kísérlet feltétele szerint olyaféle válaszokat kell kapnunk, mint „Jól érzem magam. Meg kell mondnom, egy kicsit meg vagyok lepődve, hiszen elhittem, amit Searle mondott”. Vagy pedig egy mutatópálcával egy kicsit megszúrhatjuk a kísérleti alanyt és megfigyelhetjük válaszát: „Juj, ez fáj.” Viszont normál körtülmények között a mesterséges intelligencia programok ilyen kimeneteit a szkeptikus elutasíthatja mint egyszerű látszatválaszokat. Hiszen elég könnyű elképzelni egy olyan szabályt, mint például: „Ha a 12-es szenzoron »Magas« érték jelenik meg, adj ki »Juj!-t«. A lényeg azonban az, hogy mivel a normál emberi agy összes funkcionális tulajdonságát átmásoltuk, feltételezhetjük, hogy elektronikus agyunk nem tartalmaz ilyen látszatválasz-mechanizmusokat. Azaz az elektronikus agy által létrehozott tudati megnyilvánulásokra olyan magyarázatot kell adnunk, amely csak a neuronok funkcionális tulajdonságaira hivatkozik. *Ennek a magyarázatnak azonban a valós agyra is vonatkoznia kell, amelynek ugyanazok a funkcionális tulajdonságai.* Ezért, úgy tűnik, kétfajta konklúzió lehetőséges.

1. A tudat kauzális mechanizmusai, amelyek ezt a kimenetet a normális agyban létrehozzák, ugyanúgy működnek az elektronikus változatban is, amely tehát tudatos.
2. A normális agy tudatos mentális eseményeinek semmilyen kapcsolata nincs a viselkedéssel, valamint nincsenek is jelen ezek az események az elektronikus agyban, amely tehát nem tudatos.

A második lehetőséget ugyan nem zárhatjuk ki, de ez arra redukálná a tudatot, amit a filozófusok **epifenomenális** (*epiphenomenal*) szereplőnek neveznek: valami: ami megötént ugyan, de mintha nem is vetne árnyákat a megfigyelhető világban. Valamint ha a tudat tényleg epifenomén, akkor az agyban lennie kell egy második, nem tudatos mechanizmusnak, amelyhez a „Juj!” tartozik.

Harmadszor pedig tekintstük az operáció visszacsinálása utáni helyzetet, amikor a kísérlet alanyának ismét normális agya van. Emlékezzünk arra, hogy a definíció szerint a kísérleti alany külső viselkedésének olyannak kell lennie, mintha az operáció nem is történt volna meg. Ennek speciális eseteként fel kell tudnunk tenni azt a kérdést: „Milyen volt az operáció alatt? Emlékszik a mutatópálcára?” A kísérleti alanynak pontos emléknyomokkal kell rendelkeznie a tudatos tapasztalatai tényleges természetéről, beleértve a qualitát is; mindenkor ellenére, hogy Searle szerint nem is voltak ilyen tapasztalatai.

Searle talán azzal válaszolna, hogy nem jól definiáltuk a kísérletet. Ha a valódi neuronokat visszahelyezésükkor úgy keltjük életre, ami az eltávolításuk óta eltelt időt nem tükrözi, akkor természetesen nem fognak „emlékezni” az operáció alatti tapasztalatokra. Hogy erre az eshetőségre is felkészüljünk, a visszahelyezéskor a mesterséges neuronok belső állapotának megfelelően módosítanunk kell az őket helyettesítő neuronok állapotát. Ha a valódi neuronok feltételezett „nem funkcionális” aspektusai ekkor egy olyan viselkedést eredményeznék, amely funkcionálisan különbözik attól, mint amely a mesterséges neuronok bent hagyásával megfigyelhető lenne, akkor *reductio ad absurdum*hoz jutnánk, mert ez azt jelenti, hogy a mesterséges neuronok funkcionálisan nem ekvivalensek a valódi neuronokkal. (A 26.3. feladat egy lehetséges utat mutat ezen érv cáfolatára.)

Patricia Churchland (Churchland, 1986) rámutat arra, hogy ha a funkcionalista érv működik a neuronok szintjén, akkor működnie kell bármely nagyobb funkcionális egység szintjén is, legyen az akár neuronok egy halmaza, egy mentális modul, egy lebony, egy agyfélteke vagy akár az egész agy. Ezek szerint ha elfogadjuk, hogy az agyprotézis kísérlet tanulsága a tudat megmaradása az agy kicsérélése során, akkor azt is el kell fogadnunk, hogy a tudat akkor is megmarad, ha az egész agyat lecserélik egy áramkörre, amely csak annyit tesz, hogy a bemenetekhez tartozó kimeneteket visszakeresi egy óriási táblázatban. Ez zavarba hozza a legtöbb embert (beleértve magát Turingot is), akik intuíciója szerint a visszakereső táblák nem tudatosak, vagy legalábbis az általuk generált tudati tapasztalatok nem azonosak azokkal a tapasztalatokkal, amelyeket egy olyan rendszer generálhat, amit le lehetne írni úgy (akár csak az egyszerűen vett számítási értelemben), mint ami hiedelmeket, önismeretet, célokat és hasonlókat tart nyilván és hoz létre. Mindez sugallja, hogy ha meg akarjuk őrizni az agyprotézis-kísérletet mint az intuíciókat vezérlő segédeszközöt, akkor nem folyamodhatunk az agy egy lépéssben történő lecseréléséhez, de azt nem jelenti, hogy kénytelenek lennének a lépésenkénti neuroncserehez folyamodni, amit Searle szeretne velünk elérni.

## A kínai szoba

Legutolsó gondolatkísérlettünk valószínűleg egyben a leghíresebb is. John Searle (Searle, 1980) gondolta ki, és egy olyan hipotetikus rendszert írt le, amely látnivalóan egy programot futtat, és megfelel a Turing-teszten; de ugyanilyen világos az is (legalábbis Searle szerint), hogy semmit sem ért meg a bemeneti és kimeneti közül. Konklúziója szerint tehát a megfelelő program futtatása (azaz a megfelelő válaszok megléte) önmagában még nem *elégséges* feltétele az elmemivoltnak.

A rendszer egy csak angolul értő emberből áll, akinek egy angol nyelven írt szabálykönyve és többfajta papírhalmaza van, melyek közül néhány papírhalmaz üres, néhányen

pedig valamilyen érhetetlen jelsorozat áll. (Tehát az ember a CPU szerepét játssza, a szabálykönyv a programnak, a papírhalmazok pedig a tárolóeszközök felelnek meg.) A rendszer egy szobában található, amelynek csak egy kis nyílása van a külvilágra. Ezen a nyíláson keresztül érhetetlen feliratú papírcetlik jelennek meg. Az ember megtalálja a szabálykönyvben ezeket a szimbólumokat, és követi az ott leírtakat. Az utasítások között szerepelhet, hogy írjon le szimbólumokat új papírcédulákra, keressen szimbólumokat a papírhalmazokban, rendezze át a halmazokat és hasonlók. Végül, az utasítások következményeként, egy vagy több szimbólumot egy papírdarabra másol, és kiadja a külvilágba.

Eddig minden könnyen követhető. De kívülről nézve azt látjuk, hogy a rendszer kínai mondatokat fogad, kínai mondatokat ad vissza, és ez a párbeszéd nyilvánvalóan éppannyira intelligens, mint amit Turing elképzelt.<sup>6</sup> Searle ennek alapján így érvel: a szobában lévő személy nem ért kínaiul (ez adott). A szabálykönyv és a papírcetlik, lévén ezek csak papírok, szintén nem értenek kínaiul. Ebből az következik, hogy a kínai nyelv semmiféle megértése nem zajlik le. *Tehát Searle szerint a megfelelő program futtatása nem vezet szükségszerűen a megértéshez.*

Turinghoz hasonlóan Searle is megvizsgált néhány lehetséges ellenvetést az érvével szemben, és meg is próbálta visszautasítani ezeket. Több kommentátor, köztük John McCarthy és Robert Wilensky, azt hozta fel, amit Searle rendszerválasznak nevez. Az ellenvetés abban áll, hogy ugyan feltehető az a kérdés, miszerint a szobában lévő személy ért-e kínaiul, de ez olyan, mint ha azt kérdeznénk, hogy a CPU tud-e köbgyököt vonni. Mindkét esetben nemmel kell felelnünk, de minden esetben – állítják a rendszerválasz hívei – a rendszer egésze *igenis* rendelkezik a kérdéses képességgel. Az biztos, hogy ha megkérdezzük a kínai szobát, ért-e kínaiul, a válasz igenlő lesz (még hozzá folyékony kínaisággal). Turing *udvarias feltételezése* szerint ennyi elegendő is. Válaszából Searle ismételten rámutat arra, hogy sem az ember, sem a papírhalmaz nem tud kínaiul, tehát semmilyen kínaiul értés sem lehetséges. Továbbá hozzáteszi, az is elképzelhető, hogy az ember megtanulja a szabálykönyvet, és a céltartalmát is a fejében tartja, tehát csupán az ember az, aki kínaiul tudhatna, ha azonban megkérdezzük őt (angolul), akkor a válasza nemleges lesz.

Most már a lényegi kérdésnél vagyunk. A papírok tartalmának memorizálása csak elterelő hadművelet, hiszen mindenkor ugyanúgy egy futó program fizikai megtestesülése. Searle igazi állítása a következő négy axiómán alapszik (Searle, 1990):

1. A számítógépprogramok formális, szintaktikai entitások.
2. Az elnémet mentális tartalma, azaz szemantikája van.
3. A szintaktika önmagában még nem elégsges a szemantikához.
4. Az elmét az agy okozza.

Az első három axiómából arra a következtetésre jut, hogy a programok nem elégsegések az elnémetek számára. Másként megfogalmazva: ugyan lehet elnevezni egy programot futató agens, de nem lesz szükségszerűen elnevezni csupán a program futtatása miatt. A negyedik axiómából pedig arra a következtetésre jut, hogy „bármely más rendszernek, amely képes elnemelni, az aggyal (legalább) megegyező kauzális erőkkel kell rendelkeznie”. Ebből pedig

<sup>6</sup> Az, hogy a papírhalmaz talán nagyobb, mint az egész bolygó, és hogy a válaszok létrehozása millió évekig eltarthat, az érv *logikai* struktúráját nem érinti. A filozófiai képzés egyik célja éppen az, hogy kialakuljon egy kifinomult érzék, amellyel különválaszthatók a releváns és az irreleváns ellenvetések.

arra jut, hogy bármely mesterséges agynak nem elég csupán egy adott programot futtatni, hanem duplikálnia kell az agy kauzális erőit, valamint hogy az emberi agy a mentális jelenségeket nem csupán egy program futtatása révén hozza létre.

Az a végkövetkeztetés, hogy az elmékhez nem elégcségesek a programok, *igenis* következik az axiómából (amennyiben elég nagyvonalúan értelmezzük azokat). De a konklúzió nem elégít ki bennünket: Searle csupán azt bizonyította be, hogy ha kifejezetten elutasítjuk a funkcionalizmust (ahogy ezt a 3. axiómájával teszi), akkor nem lehet szükségszerűen állítani azt, hogy a nem-agyak is elmék volnának. Ez teljesen elfogadható, így hát az egész érv arra jut, hogy elfogadható-e a 3. axióma. Searle szerint a kínai szoba érvek éppen az a lényege, hogy intuíciót ad a 3. axióma elfogadásához. De az érvére adott reakciók azt mutatják, hogy csak azoknak nyújt intuitív alapot, akik már eleve hajlanak annak elfogadására, hogy a programok önmagukban nem hozhatnak létre igazi megértést.

Megismételve tehát, a kínai szoba érv célja az, hogy megcáfolja az erős MI-hipotézist: azt az állítást, miszerint a helyes változatú program futtatása szükségszerűen elmét eredményez. Úgy működik az érv, hogy bemutat egy látszólag intelligens, a megfelelő típusú programot futtató rendszert, amely Searle szerint *bizonyíthatóan* nem elme. Ennél a résznél azonban Searle az intuícióra és nem egy bizonyítéakra támaszkodik: nézzük csak a szobát, hol lenne itt az elme? De ez az érv ugyanúgy felhozható az aggyal kapcsolatban is: nézzünk rá a sejtek (vagy az atomok) sokaságára, amelyek vakon működnek a biokémia (vagy a fizika) törvényei szerint – hol lenne itt az elme? Miért tud egy nagy darab agy elme lenni, amikor egy nagy darab máj nem?

Továbbmenve, amikor Searle beisméri, hogy elméletileg a neuronuktól különböző anyag is lehet elme, akkor két okból is tovább gyengül az érve: egyrészt csak Searle intuícióira (vagy a saját intuícióinra), támaszkodhatunk, hogy eldöntsük, elme-e a kínai szoba; másrészt, még ha úgy is döntünk, hogy a szoba nem elme, ez semmit nem mond arról, hogy a programot futtató más fizikai közeg (például a számítógép) lehet-e elme.

Searle megengedi azt a logikai lehetőséget, hogy az agy éppenséggel egy hagyományos típusú logikai programot valósít meg; de ugyanazt a programot egy rosszfajta gépen futtatva nem kapnánk elmét. Searle tagadja, hogy azt gondolná, „a gépeknek nem lehet elmeje”, sőt azt állítja, hogy bizonyos gépeknek *igenis* van elmeje: az emberek elmével rendelkező biológiai gépek. Nem hagy azonban túl sok útmutatást afelől, hogy az elmével rendelkezés a gépek milyen csoportjára terjed ki.

## 26.3. A MESTERSÉGES INTELLIGENCIA FEJLESZTÉSÉNEK ETIKAI KÉRDÉSEI ÉS KOCKÁZATAI

Mindeddig arra összpontosítottunk, hogy *képesek vagyunk-e* kifejleszteni egy mesterséges intelligenciát, most azt is meg kell vizsgálnunk, hogy *kell-e* ezt tennünk. Amennyiben az MI-technológiák hatásai többségében inkább negatívak, mint pozitívak lennének, akkor morális felelőssége lenne az ezen a területen dolgozóknak, hogy más területekre helyezzék kutatásukat. Sok új technológia járt akaratlanul is negatív mellékhatásokkal: a belső égésű motor a légszennyezést hozta és azt, hogy még az Édenkertet is leaszfaltozták, a maghasadás felfedezése pedig a csernobili és a Three

Mile Island-i<sup>7</sup> atomkatasztrófákat hozta, és a globális pusztulás veszélyének fenyegését. minden tudónak és mérnöknek szembe kell néznie a munkájukkal kapcsolatos etikai kérdésekkel: mely projekteket szabad befejezni, és melyeket nem, és hogyan kell ezeket a projekteket kezelni. Még egy *Ethics of Computing* (Berleur és Brumstein, 2001) c. kézikönyv is létezik. A mesterséges intelligencia azonban, úgy tűnik, néhány új problémát is felvet, azon túl, hogy olyan hidakat akarunk építeni, amelyek nem dőlnek össze:

- Az emberek az automatizáció miatt elveszíthetik a munkájukat.
- Az embereknek túl sok (vagy túl kevés) szabadidejük marad.
- Az emberek elveszíthetik az egyediségérzésüket.
- Az emberek elveszíthetik a személyiségi jogai egy részét.
- Az MI-rendszer alkalmazása megszüntetheti a felelősségre vonhatóságot.
- A mesterséges intelligencia sikere az emberi faj végét jelentheti.

Sorra vesszük mindegyik felvetett problémát.

*Az emberek az automatizáció miatt elveszíthetik a munkájukat.* A modern ipari gazdaság általában véve függővé vált a számítógépektől, sőt néhány különleges MI-programtól is. A gazdaság nagy része például, különösen az Egyesült Államokban, a fogyasztói hiteltől függ. A hitelkártya-kérelmek elbírálását, a lehívások engedélyezését és a visszaélések felderítését napjainkban mesterséges intelligencia programok végzik. Azt is mondhatná valaki, hogy ezek a programok több ezer dolgozót tettek munkanélkülivé, de valójában, ha elvennének ezeket a mesterséges intelligencia programokat, nem is léteznének ilyen állások, mert az emberi munka elviselhetetlen többletköltséget róna ezekre a tranzakciókra. A mesterséges intelligenciát használó gépesítés mindenkorábban több állást hozott létre, mint ahányat megszüntetett, és ezek az új állások érdekesebbek, és jobban fizetnek. Ma már, amikor a mesterséges intelligencia programokat az embereket segítő „intelligens ágenseknek” tekintik, kevesebben félnek a munkahelyük elvesztésétől, mint amikor a mesterséges intelligencia célja az embereket helyettesítő „szakértői rendszerek” létrehozása volt.

*Az embereknek túl sok (vagy túl kevés) szabadidejük marad.* Alvin Toffler a *Future Shock*-ban (Toffler, 1970) ezt írta: „A századforduló óta a munkahét 50%-kal csökkent. Nem lóg ki a sorból ha azt jósoljuk, hogy 2000-re ismét felével csökken.” Arthur C. Clarke (Clarke, 1968b) azt írta, hogy 2001-ben az emberek talán már a „teljes unalommal néznek szembe, ahol az élet legnagyobb problémája a több száz tv-csatorna közti választás lesz”. Az egyetlen ezen jóslatok közül, amelyik közel áll a megvalósuláshoz, az a tv-csatornák száma (Springsteen, 1992). Valójában a tudásintenzív iparágakban dolgozó emberek egy 24 órában működő, integrált számítógépesített rendszer részévé válnak, és ahoz, hogy lépést tartanak, *hosszabb* munkaórákra lettek kényszerítve. Az ipari gazdaságban a jutalom közelítőleg egyenesen arányos a befektetett idővel; 10%-kal több munka többé-kevésbé 10%-os bevételnövekedést jelent. Az információs gazdaságban, amelyet a nagy sávszélességű kommunikáció és a szellemi tulajdon könyű replikációja jellemzi (ezt nevezi Frank és Cook „a-győztes-mindent-visz társada-

<sup>7</sup> Az Egyesült Államok eddigi legsúlyosabb atombalesetének helyszíne Pennsylvania államban. Az 1979-ben történt baleset ugyan rendkívül megrázta az amerikai közvélemyét, de káros egészségügyi következményeit tekintve nem méhető a csernobili esethez. (A ford.)

lom"-nak) (Frank és Cook, 1996), nagy jutalom jár azért, ha valaki csak egy kicsit is jobb a versenyben: 10%-kal több munka akár 100%-os bevételnövekedést is jelenthet. Ezért egyre növekvő nyomás nehezedik mindenkire, hogy keményebben dolgozzon. A mesterséges intelligencia növeli a technológiai innováció ütemét, tehát hozzájárul ehhez az általános trendhez, de egyben a mesterséges intelligencia tartalmazza annak ígéretét is, hogy az automatizált ágensek kicsit átvállalják a teendőket és egy kis időhöz juttatnak minket.

*Az emberek elveszíthetik az egyediségérzésüket.* Weizenbaum, az ELIZA program szerzője, *Computer Power and Human Reason* (Weizenbaum, 1976) c. művében rámutat néhány potenciális veszélyre, amelyeket a mesterséges intelligencia jelent a társadalom számára. Weizenbaum egyik fő érve az, hogy a mesterséges intelligencia kutatása lehetővé teszi annak elképzelését, hogy az emberek automaták volnának, amely elgondolás az autonómia, vagy akár a humanitás érzésének megszűnéséhez vezet. Vagyük észre azonban, hogy ez az elképzelés jóval régebb óta van jelen, mint a mesterséges intelligencia: legalábbis a *L'Homme Machine* (La Mettrie, 1748) c. könyvig vezethető vissza. Azt is vegyük észre, hogy az emberiség túlélte már az egyediségérzésnek más csökkenéseit is: A *De Revolutionibus Orbium Coelestium* (Kopernikusz, 1543) kimozdította a Földet a Naprendszer központjából, és a *Descent of Man* (Darwin, 1871) pedig a *Homo sapiens*t egy szintre helyezte a többi fajjal. A mesterséges intelligencia, amennyiben széles körben sikeres lesz, éppen úgy veszélyt jelenthet a 21. század morális feltételezéseire, ahogyan Darwin evolúciós elmélete fenyegette a 19. század morális feltételezéseit.

*Az emberek elveszíthetik a személyiségi jogai egy részét.* Weizenbaum arra is rámutatott, hogy a beszédfelismerési technológia a lehallgatás elterjedéséhez, és így a polgári jogok veszteségéhez vezethet. Nem láthatta előre azt a világot, ahol a terroristák fenyegettség megváltoztatja, hogy az emberek mennyi megfigyelést hajlandók elfogadni, azt azonban helyesen ismerte fel, hogy a mesterséges intelligencia a tömeges megfigyelés eszközévé válhat. Jósłata talán már valóra is vált: az Egyesült Államok kormányának titkos Echelon-rendszere „felvezőállomások, antennarészletek és radarállomások hálózatából áll, a rendszert nyelvek közti fordítást, beszédfelismerést és kulcsszavas keresést használó számítógépek támogatják, amelyek a telefon-, az e-mail, a fax- és a telexforgalmat vizsgálják végig.”<sup>8</sup> Néhányan elfogadják, hogy a számítógépesítés a magánszféra veszteségéhez vezet. Scott McNealy, a Sun Microsystems vezetője azt mondta: „Amíg is nulla a magánszférád. Felejtsd el.” Mások nem értenek ezzel egyet: Louis Brandeis bír<sup>9</sup> ezt írta 1890-ben: „A magánszférához való jog minden jogok legalapvetőbbike (...) a jog az ember saját személyiségehez.”

*Az MI-rendszerek alkalmazása megszüntetheti a felelősségre vonhatóságot.* Az Egyesült Államokban ma uralkodó pereskedő közhangulat fontos kérdéssé tette a jogi felelősséget. Amikor egy orvos a diagnózis felállításakor egy orvosi szakértői rendszer ítéletére hagyatkozik, ki tehető felelőssé a diagnózis hibájáért? Szerencsés módon, és ez részben a döntéselméleti megközelítés orvoslásra gyakorolt egyre növekvő befolyásának köszönhető, ma már elfogadott alapelv, hogy az orvos, aki végrehajtja a nagy várható hasznossági értékű beavatkozásokat, akkor sem vádolható hanyagsággal, ha ezeknek a

<sup>8</sup> Lásd „Eavesdropping on Europe”, *Wired* hírmagazin, 9/30/1998 és a hivatkozott EU-jelentéseket.

<sup>9</sup> Amerikai jogász és jogi közíró, a múlt század első felében a Legfelsőbb Bíróság tagja. (A ford.)

beavatkozásoknak katasztrofálisak a *tényleges következményei* a páciensnél. A kérdés ezek után a következő: „Kinek a hibája, ha a diagnózis nem plauzibilis?” Mindeddig a bíróságok úgy tartották, hogy az orvosi szakértői rendszerek ugyanazt a szerepet töltik be, mint az orvosi tankönyvek és kézikönyvek: az orvosoknak felelőssége, hogy megértsék a döntések indoklását, és saját megítélésük alapján kell határozniuk, hogy elfogadják-e a rendszer javaslatát. Tehát ha az orvosi szakértői rendszereket ágensekként tervezzük, akkor cselekvéseiket nem úgy kell tekinteni, mint ami közvetlenül kihat a páciensre, hanem mint ami befolyásolja az orvos viselkedését. Ha a szakértői rendszerek megbízhatóbban pontosabbak lesznek, mint a diagnózisokat felállító emberek, akkor az orvosok akár jogilag is felelősek lehetnek, ha *nem* használják fel a rendszer javaslatait. Gawande (Gawande, 2002) ezt a feltételezést térképezi fel.

Hasonló kérdések kezdenek megjelenni az interneten használt intelligens ágensekkel kapcsolatban. Már elértek egy kis haladást kényszerfeltételek beépítésével az ágensekbe, hogy például ne okozhassanak károkat más felhasználók állományaiban (Weld és Etzioni, 1994). Amikor pénz is gázdát cserél, a probléma már nagyobb. Amikor egy pénzügyi tranzakció során egy intelligens ágens „valaki nevében” jár el, felelős-e ez a személy a keletkező adósságokért? Lehetségesse válhat-e, hogy egy intelligens ágensnek pénzügyi kintlévősége legyen, és a saját nevében eljárva elektronikus tranzakciókat hajtson végre? Mindeddig nem látszanak világosnak ezek a kérdések. Tudomásunk szerint eddig egy program sem kapta meg azt a jogi státust, hogy individuumként pénzügyi tranzakciókat hajthasson végre, és jelenleg ez nem is látszik indokoltnak. A valódi autópályákon sem tekintik a közlekedési szabályok betartatásakor „vezetőnek” a programokat. A kaliforniai jogban legalábbis semmilyen jogi szankció nincsen, amely megakadályozhatná, hogy egy automata jármű túllépje a sebességhatárt, noha baleset esetén felelőssé tehető a jármű vezérlőmechanizmusának tervezője. Csakúgy, mint az emberi megtermékenyítési technológia esetén, a jog még nem vette fel a lépést az új fejleményekkel.

A mesterséges intelligencia sikere az emberi faj végét jelentheti. Gyakorlatilag bármely technológiában benne rejlik a lehetőség, hogy kárt okozzon rossz kezekben, de a mesterséges intelligencia és a robotika esetében ezek a rossz kezek akár magához a technológiához is tartozhatnak. Számtalan sci-fi szól figyelmeztető jelként arról, hogy robotok vagy robot-ember kiborgok ámokfutást rendeznek. Marry Shelley *Frankenstein, or the Modern Prometheus* (Shelley, 1818)<sup>10</sup> c. műve és Karel Čapek *R.U.R.* c. drámája, amelyben a robotok meghódítják a világot, korai példái az ilyen történeteknek. Mozifilmek is szólnak erről: a *The Terminator* (1984) a robotok-leigázzák-a-világot kliséit az időutazással ötvözi, a *The Matrix* (1999) pedig ugyanezt az agyak-a-tartályban-nal kombinálja.

Úgy tűnik, legtöbb esetben a robotok azért főszereplői olyan sok világ-leigázása történetek, mert az ismeretlen jelképezik, csakúgy, mint a korábbi mesék boszorkányai és szellemei. Jelentenek-e a boszorkányoknál és a szellemeknél komolyabb veszélyt? Valószínűleg nem: ha a robotokat megfelelően, azaz olyan ágensekként tervezik, amelyek a gazdáik céljait teljesítik, akkor a jelenlegi tervezés lépésekben előrehaladásából származó robotok szolgálni fognak, nem pedig leigázni. Az emberek azért használják agresszíven az intelligenciájukat, mert a természetes kiválasztódás miatt veltük született agresszivitással rendelkeznek. De a gépek, amiket magunk építünk, nem születnek agresszívnak, hacsak nem döntünk úgy, hogy ilyenek építjük őket. Másrészt viszont

<sup>10</sup> Charles Babbage-ra is hatással volt fiatalemberként a *Frankenstein* olvasása.

lehetséges, hogy a számítógépek meghódítanak minket abban az értelemben, hogy szolgálatukkal elengedhetetlenné válnak, csakúgy, mint az autók meghódították (ebben az értelemben) az iparosodott világot. Egy forgatókönyv részletesebb tanulmányozást érdekel. I. J. Good (Good, 1965) írta ezt:

Definiáljuk az ultraintelligens gépet olyan gépként, amely messze túlhaladja a lehető legintelligensebb ember intellektuális tevékenységét is. Mivel a gépek tervezése az intellektuális tevékenységek egyike, egy ultraintelligens gép még jobb gépeket tud tervezni; nem kérdés, hogy egy „intelligenciárólbanás” történne, ami messze lehagyja az emberi intelligenciát. Az ultraintelligens gép így egyben az emberiség utolsó szükséges felfedezése, feltéve persze, hogy a gép elég engedelmes, és elmondja, hogyan tarthatjuk ellenőrzésünk alatt.

Az „intelligenciárólbanás”-t **technológiai szingularitásnak (technological singularity)** is nevezte a matematika-professzor és sci-fi író Vernor Vinge, aki így fogalmaz (Vinge, 1993): „Harminc éven belül rendelkezni fogunk a technológiai eszközökkel egy emberfeletti intelligencia létrehozására. Röviddel ezután befejeződik az ember korszaka.” Good és Vinge (és még sokan mások) jól figyelik meg, hogy a technológiai fejlődés jelenleg az exponenciális görbe szerint nő (gondoljunk csak a Moore-törvényre). Azonban ebből csak egy előreugrással lehet arra következteni, hogy ez a görbe így fog folytatódni a közel végig növekedés szingularitásához. Eddig minden más technológia egy S alakú görbét követett, ahol az exponenciális növekedés végül lecsengett.

A bekövetkező szingularitásról író Vinge-et rémíti ez a lehetőség, de más számító-géptudósok és futuristák örömmel várják. Hans Moravec a *Robot: Mere Machine to Transcendent Mind* c. írásában azt jósolja, hogy a robotok ötven éven belül elérík az emberi intelligenciát, majd pedig meghaladják azt. Ezt írja:

Egész gyorsan kiszorítanak minket a létezésből. Másokkal szemben engem ez a lehetőség nem tölt el aggodalommal, én ezeket a jövőbeli gépeket a leszármazottainknak, „elmebeli gyermekaink”-nek tartom, akiket képünkre és hasonlatosságunkra építettünk, önmagunkként egy jóval nagyobb képességű alakban. A korábbi generációk biológiai gyermekéihez hasonlóan az emberiség hosszú távú jövőről alkotott legjobb reményét testesítik meg. Belső kötelességünk minden előnyt megadni nekik és ellépni útjuktól, ha már többet nem tudunk segíteni. (Moravec, 2000)

A *The Age of Spiritual Machines* (Kurzweil, 2000) c. művében Ray Kurzweil azt jósolja, hogy 2000-re már „erős trend lesz összekapcsolni az emberi gondolkodást az eredetileg az emberi faj teremtette gépek világával”. Még egy új szó is van: **transzumanizmus (transhumanism)**, amely az ezt a jövőt váró aktív társadalmi mozgalmat jelöli. Elegendő csak annyit mondani, hogy az ilyen kérdések kihívást jelentenek a legtöbb mai erkölcsi gondolkodónak, akik az emberi élet és az emberi faj megőrzését jó dolognak tartják.

Gondoljunk bele végül a robotok nézőpontjába. Ha a robotok tudatossá válnak, akkor immorális lehet pusztta „gépként” kezelni őket (például darabokra szedni). A robotoknak maguknak is morálisan kell cselekedniük: beléük kell programozni a jó és a rossz elmeletét. A sci-fi írók, kezdve Isaac Asimovval (Asimov, 1942), már el is kezdtek foglalkozni a robotjogok és robotfelelősségek kérdéseivel. A jól ismert – *Mesterséges intelligencia (A.I.)* (Spielberg, 2001) c. mozi film Brian Aldiss történetén alapul, amelyben egy intelligens robot, akit arra programoztak, hogy embernek képzelje magát, képtelen feldolgozni, hogy nem kell tulajdonos-anyjának. A történet (és persze a film) a robotok polgárjogi mozgalmának megalapozottságáról győzhet meg egyeseket.

## 26.4. ÖSSZEFOGLALÁS

Ebben a fejezetben a következő kérdésekkel foglalkoztunk:

- A filozófusok szóhasználata szerint a **gyenge MI** kifejezés azt a feltevést jelenti, hogy a gépek valószínűleg képesek intelligensen viselkedni, az **erős MI** pedig azt a feltevést, hogy az ilyen gépek elmével rendelkezőknek számítanának (és nem elmét szimulálóknak).
- Alan Turing visszautasította a „Tudnak-e gondolkodni a gépek?” kérdést, és egy viselkedési tesztet állított a helyébe. Sok, a gondolkodó gépek lehetőségével szemben felhozható ellenvetést megelőlegezett. A mesterséges intelligencia kutatói közül kevesen szentelnek figyelmet a Turing-tesztre, előnyben részesítik, ha a rendszerek gyakorlati problémamegoldó képességére összpontosítunk, nem pedig az emberek utánzására való képességükre.
- A modern korban általábanosan egyetértenek abban, hogy a mentális állapotok agy-állapotok.
- Sem az erős MI ellen, sem a mellette felhozható érvek nem meggyőzők. Kevés első vonalbeli MI-kutató gondolja azt, hogy bármi fontos is függ ennek a vitának a kimentelétől.
- A tudatosság rejtély maradt.
- A mesterséges intelligencia és a kapcsolódó technológiák által a társadalomra jelenléttel veszélyek hat fajtáját azonosítottuk. Arra a következetésre jutottunk, hogy néhány veszély valószínűtlen, kettő azonban további megfontolást érdemel. Az első az, hogy az ultraintelligens gépek, a maitól nagyon különböző, ámde ránk nézvést korántsem kedvező jövőhöz vezethetnek. A második pedig az, hogy a robottechnológia a pszichopatáknak is a kezébe adhatja a tömegpusztító fegyverek kulcsát. Arra jutottunk azonban, hogy ez a fenyegetés inkább származik a biotechnológiából és a nanotechnológiából, mintsem a robotikából.

### Irodalmi és történeti megjegyzések

Az elme természete az óktortól napjainkig a filozófiai elméletalkotás állandó téma. Platón a *Phaidón* c. dialógusában kifejezetten megvizsgálta, majd elvetette azt a lehetőséget, hogy az elme a test részeinek „egybehangoltsága”, szerveződési mintája lenne (ez az álláspont közelít ahhoz, amit a modern elmefilozófiában funkcionalizmusnak nevezünk). Ehelyett arra az álláspontra helyezkedett, hogy az elme egy halhatatlan, nem anyagi lélek, amely a testtől elválasztható, és szubstanciájában (lényegi alkotóelemét tekintve) különbözik: ez a dualizmus álláspontja. Arisztotelész az élő lényekre jellemző lélek (görögül  $\psi\chi\eta$ ) több fajtáját különítette el, és legalábbis néhányukra funkcionalista leírást adott (erről bővebben lásd (Nussbaum, 1978)).

Hírhedt Descartes dualistikus elmelefogása, de ironikus módon történelmi hatása inkább a mechaniszta és materialista elmelefogás felé mutat. Az állatokat kifejezetten automatának tartotta, és megelőlegezte a Turing-tesztet: „azt nem tudjuk elképzelni, hogy [egy gép] a szavakat különféleképpen elrendezze, s ezáltal értelmesen tudjon felelni arra, amit a jelenlétében mondanak, amint ezt a legtompábbszű emberek is meg-

tudnak tenni” (Descartes, 1637).<sup>11</sup> Azzal, hogy hevesen kardoskodott az állatok automata mivolta mellett, Descartes valójában könnyebben tette, hogy az embereket is automatának véljük, noha maga ezt a lépést sohasem tette meg. A *L'Homme Machine* (La Mettrie, 1748) c. könyv pontosan amellett érvelt, hogy az emberek automaták.

A modern analitikus filozófia általában elfogadja a materializmust (gyakran az agy-állapot-azonosság elmélet (**identity theory**) formájában [Place, 1956; Armstrong, 1968], ami szerint a mentális állapotok azonosak az agyi állapotokkal), de sokkal megosztottabb a funkcionalizmust, az emberi elme gépi analógiáját tekintve, valamint abban a kérdésben, hogy szó szerinti értelemben véve tudnak-e gondolkodni a gépek. Turing (Turing, 1950) *Computing Machinery and Intelligence* c. cikkére adott korai filozófiai válaszok egy része, például Scriven (Scriven, 1953), megpróbálta cáfolni, hogy **értelmes** lenne azt mondani, hogy a gépek gondolkodhatnak, arra alapozva, hogy ez sérténé a szó jelentését. Ezt a nézetet, legalábbis Scriven, cikkének újranyomásához illesztett függelékének tanúsága szerint 1963-ra visszavonta (Anderson, 1964). A számítástechnika-kutató Edsger Dijkstra szerint „Az a kérdés, hogy tud-e egy gép gondolkodni, nem érdekesebb annál, hogy tud-e úszni egy tengeralattjáró.” Ford és Hayes (Ford és Hayes, 1965) amellett érvelnek, hogy a Turing-teszt semmi segítséget sem nyújt a mesterséges intelligencia számára.

Mivel az elmefilozófiában a mesterséges intelligencia alapján a funkcionalizmus választása tűnik a legtermészetesebbnek, a funkcionalizmus kritikája tehát gyakran a mesterséges intelligencia kritikájának formáját ölti (mint Searle esetében). Block (Block, 1980) osztályozását követve többféle funkcionalizmust különböztethetünk meg. Az agy-állapot-azonosság elmélet egyik variánsa, a **funkcionális specifikáció elmélete (functional specification theory)** (Lewis, 1966; 1980) a mentális állapotokkal identifikálandó agyi állapotokat a funkcionális szerepük választja ki. A gépi analógiához szorosabban kötődik a **funkcionális állapotazonosság elmélet (functional state identify theory)** (Putnam, 1960; 1967). E szerint a mentális állapotokat nem fizikai agyi állapotokkal, hanem a kifejezett számítószereként felfogott agy absztrakt számítási állapotaival kell azonosítani. Ezek az absztrakt állapotok feltételezés szerint függetlenek az agy specifikus felépítésétől, ami azonban néhányakat arra készít, hogy a funkcionális állapot-azonosság elméletet a dualizmus egy formájának tartás!

Mindkét agyállapot-azonosság elméletet, valamint a különböző formájú funkcionalizmusokat támadják azok a szerzők, akik szerint ezek az elméletek nem megfelelők a mentális állapotok *qualia*, azaz „milyen érzés is ez” aspektusának megmagyarázására (Nagel, 1974). Searle ehelyett az intencionalitás funkcionalizmusbeli állítólagos megmagyarázhatatlanságára koncentrált (Searle, 1980; 1984; 1992). Churchland és Churchland (Churchland és Churchland, 1982) minden fajta kritikát visszautasítják.

Az **eliminatív (kiküszöbölő) materializmus (eliminative materialism)** (Rorty, 1965; Churchland, 1979) abban különbözik az elmefilozófia többi jelentős elméletétől, hogy meg sem kíséri a „népies pszichológia”, az elmről alkotott hétköznapi elképzeléseink igazolását, hanem minden elveti, és megpróbálja az elme tisztán tudományos elméletével helyettesíteni. Elméletileg akár a klasszikus mesterséges intelligencia is szolgáltat-

<sup>11</sup> A szöveget a magyar fordítás alapján idéztük: *Értekezés a módszerről* (ford. Szemere Samu és Boros Gábor). (A ford.)

hatná ezt a tudományos elméletet, de valójában az eliminatív materialisták inkább az idegtudománytól és a neurális hálók kutatásától kölcsönöznek (Churchland, 1986), arra alapozva, hogy a klasszikus mesterséges intelligencia, különösen a 10. fejezetben bemutatott hoz hasonló „tudásreprezentációs” kutatási irány, hajlamos a népies pszichológia igazságaira alapozni. Ugyan az „intencionális beállítottság” álláspont (Dennett, 1971) értelmezhető funkcionalistaként, de valószínűleg az eliminatív materializmus egyik formájának kell tekinteni, mivel az intencionális beállítottság nem objektív tulajdonsága az ágensnek, akire vonatkozik. Azt is vegyük figyelembe, hogy lehetünk eliminatív materialisták egyes mentális jelenségekkel kapcsolatban, miközben másokat nem így elemzünk. Dennett (Dennett, 1978) például a qualitákat illetően sokkal inkább eliminativista, mint az intencionalitással kapcsolatban.

A fejezetben szóltunk a gyenge MI-vel szembeni kritika lehetséges forrásairól. Ugyan a posztneurális hálók korszakában divatossá vált idejétmúlnak tartani a szimbolikus megközelítéseket, nem minden filozófus kritikus a „Jófajta, régivágású MI”-vel szemben. Akadnak ennek még elkötelezettségek, sőt akár gyakorló hívei is. Zenon Wylshyn (Wylshyn, 1984) amellett érvelt, hogy a számítási modell nemcsak elviekin a legjobb modell a megismerés megértéséhez, hanem a jelenlegi kutatás útja is lehet, és kifejezetten megcáfolta Dreyfus kritikáját az emberi megismerés számítási modelljéről. Gilbert Harman (Harman, 1983) a hiedelemmódosítás vizsgálatát az igazság-karbantartó mesterséges intelligencia rendszerek kutatásához kapcsolta. Micheal Bratman az emberi pszichológiáról alkotott „hiedelem-vágy-intenció” modelljét (Bratman, 1987) alkalmazta a tervkészítéssel kapcsolatos mesterséges intelligencia kutatásokra (Bratman, 1992). Aaron Sloman (Sloman, 1978, iii. o.), a mesterséges intelligencia erős hipotézisének radikális híve odáig elment, hogy rasszistának nevezte Joseph Weizenbaumot (Weizenbaum, 1976), mert szerinte a feltételezett intelligens gépeket nem kellene személyeknek tekinteni.

Az elméről, az agyról és a kapcsolódó témakról szóló filozófiai irodalom kiterjedt, és a megfelelő szaknyelv és érvelési módozatokra vonatkozó megalapozott képzés hiányában gyakran nehezen olvasható. Ehhez a folyamathoz az *Encyclopedia of Philosophy* (Edwards, 1967) felettesebb hasznos segítséget nyújthat meggyőzően informatív szócikkeivel. A *The Cambridge Dictionary of Philosophy* (Audi, 1999) rövidebb, de közérthetőbb mű, noha a fő szócikkek (mint például az „elmefilozófia”) még így is akár több mint tíz oldal hosszúak is lehetnek. A *MIT Encyclopedia of Cognitive Science* (Wilson és Keil, 1999) egyaránt foglalkozik az elme filozófiájával, biológiájával és pszichológiájával. Az elmefilozófiáról (beleértve ebbe a funkcionálizmust és más, a mesterséges intelligenciához kötődő álláspontokat) szóló cikkek általános gyűjteménye található a *Materialism and the Mind-Body Problem*-ben (Rosenthal, 1971) és a *Readings in the Philosophy of Psychology* (Block, 1980) első kötetében. Biro és Shanan (Biro és Shanan, 1980) a funkcionálizmus ellen és mellett érvelő cikkekből állított össze antológiát. Más cikkgyűjtemények kifejezetten a filozófia és a mesterséges intelligencia kapcsolatával foglalkoznak: *Minds and Machines* (Anderson, 1964); *Philosophical Perspectives in Artificial Intelligence* (Ringle, 1979); *Mind Design* (Haugeland, 1981) és a *The Philosophy of Artificial Intelligence* (Boden 1990). Számtalan bevezető létezik a filozófiai „MI-kérdés”-hez (Boden, 1977; 1990; Haugeland, 1985; Copeland, 1993). A *The Behavioral and Brain Sciences*, rövidítve *BBS*, fontos folyóirat, amelyet a mesterséges intelligenciával és az idegtudományokkal kapcsolatos

vitáknak szentelnek. Több újság foglalkozik a mesterséges intelligencia etikai és felelősségi kérdéseivel: *AI and Society*, *Law, Computers and Artificial Intelligence*, valamint az *Artificial Intelligence and Law*.<sup>12</sup>

## Feladatok

- 26.1. Haladjon végig Turing állítólagos gépi „fogyatékosságokról” alkotott listáján, keresse meg, melyeket valósítottak már meg, melyeket lehet elvben megvalósítani egy programmal, és melyek problematikusak azért, mert tudatos mentális állapotokat igényelnek.
- 26.2. Szükségszerűen bizonyítsa-e a kínai szoba érv cáfolata, hogy a megfelelően programozott számítógépek mentális állapotokkal rendelkeznek? Szükségszerűen együtt jár-e az érv elfogadása azzal, hogy a számítógépeknek nem lehetnek mentális állapotaik?
- 26.3. Az agyprotézis érv fontos része, hogy képesnek kell lennünk a kísérleti személy agyát visszaállítani a normális állapotába, hogy külső viselkedése ugyanolyan legyen, mintha a műtét nem is történt volna meg. Megalapozott lehet-e egy olyan szkeptikus ellenvetés, miszerint ez azzal járna, hogy módosítani kell a neuronoknak a tudatossághoz kötődő neurofiziológiai tulajdonságait, melyek különböznek a funkcionális viselkedésben résztvevőktől.
- 26.4. Keressen ki a populáris médiából egy vagy több, a mesterséges intelligencia lehetetlenségére irányuló érvet, és elemezze azt.
- 26.5. Próbáljon meg definíciókat adni az „intelligencia”, a „gondolkodás” és a „tudatosság” fogalmakra. Vessen fel lehetséges ellenvetéseket is a definíciókkal szemben.
- 26.6. Elemezze a mesterséges intelligenciának a társadalomra jelentett potenciális veszélyeit. Mely fenyegétek a legsúlyosabbak, és hogyan lehetne küzdeni ellenük? Hogyan viszonyulnak ezek a potenciális előnyökhez?
- 26.7. A mesterséges intelligencia jelentette potenciális fenyegétek hogyan viszonyulnak más számítástechnikai, vagy bio-, nano- és nukleáris technológiákból eredő veszélyekhez?
- 26.8. Néhány kritikus ellenvetése szerint a mesterséges intelligencia lehetetlen, mások szerint meg *tílusásgosan is* lehetséges, és az ultraintelligens gépek veszélyt jelentenek. A két felhozott ellenvetés közül melyiket tartja valószínűnek? Ellentmondásos-e a kettőt együtt állítani?

<sup>12</sup> A magyar olvasó a mesterséges intelligencia filozófiájáról Darabos Tamás művéből tájékozódhat: *A geopolitikai értelem. Vázlatok a mesterséges intelligencia filozófiájáról*. Áron Kiadó, Budapest, 1991. (A ford.)

# 27. MI: JELEN ÉS JÖVŐ

*Ebben a fejezetben felmérjük, hogy hol vagyunk és merre tartunk, mielőtt folytatnánk. Talán nem felesleges.*

Az I. részben bevezettük a racionális ágensrendszeret mint a mesterséges intelligencia egységes szemléleti keretét. Bemutattuk, hogy a tervezési probléma az ágens rendelkezésére álló észleléseken és cselekvéseken, az ágens viselkedésével kielégítendő célon és a környezet természetén múlik. Az ágensrendszertervezek egész sorát lehet elkövetni, a reflexszerű ágenstől kezdve egészen a teljesen célutatatos, tudásalapú ágensig. Továbbá ezen rendszertervezek komponensei többféle konkrét alakot öltethetnek – például logikait, valószínűségit vagy „neurálist”. Az ilyen jellegű komponensek működésével a megfelelő fejezetek foglalkoztak.

Az ágensrendszer és komponensei tekintetében mind tudományos megértésünk, mind technológiai képességünk hatalmasat fejlődött. Ebben a fejezetben hátrébb lépünk a részletektől, és azt kérdezzük: „Elvezet-e ez a fejlődés egy változatos környezetekben működésre képes, általános célú intelligens ágenshez?” A 27.1. alfejezetben számba vesszük az intelligens ágens komponenseit, hogy meghatározhassuk, mi az, amit ismerünk, és mi az, ami még hiányzik. A 27.2. alfejezetben az általános ágensarchitektúra kapcsán tesszük ugyanezt. A 27.3. alfejezetben azt vizsgáljuk, hogy a „racionális ágens tervezése”-e az elsőlegesen helyes cél. (A válasz: „Nem igazán, de egyelőre megteheti.”) Végül a 27.4. alfejezetben azzal foglalkozunk, milyen következményekkel járhat vállalkozásunk sikere.

## 27.1. ÁGENSÖSSZETEVŐK

A 2. fejezetben számos ágenstervet és azok komponenseit mutattuk be. Figyelmünket összpontosítandó itt csak a hasznosságával foglalkozunk, amelyet a 27.1. ábrán mutatunk be újra. Ez a legáltalánosabb ágenstervünk; de megvizsgáljuk tanulási képességekkel való kiterjesztését is (lásd 2.15. ábra).

*Interakció a környezettel szenzorok és beavatkozószervek segítségével.* Ez a mesterséges intelligencia történetének nagy részében komoly hiányosság. Néhány tiszteletre méltó kivétellel minden MI-rendszert úgy építettek fel, hogy embereknek kellett szolgáltatniuk a bemeneteket, és értelmezni a kimeneteket, a robotika pedig alacsony szintű feladatokra összpontosított, ahol nagyrészt hiányzott a magas szintű következtetés és tervkészítés. Ez részben a magas költségek és a komoly tervezői erőfeszítés miatt volt így, ami ahhoz kellett, hogy egyáltalán valódi, működő robotokat kapunk. Az elmúlt években az előre elkészített, programozható robotok – mint például a 25.4. (b.) ábrán látható négylábú

robot – megjelenésével a helyzet rohamosan megváltozott. Ez a változás végső soron a kicsi, olcsó, nagy felbontású CCD-kameráknak és a kompakt, megbízható motoros hajlásoknak köszönhetően következett be. Az MEMS- (mikro-elektromechikai rendszerek) technológia miniatürizált gyorsulásmérőket és giroszkópokat hozott létre, napjainkban pedig olyan beavatkozószerveket állít elő, amelyek például egy mesterséges repülő rovar bizonyos képességekkel ruháznak majd fel. (Talán MEMS-beavatkozószervek milliói kapcsolhatók majd össze, hogy nagy teljesítményű makroszkopikus beavatkozókat kapjunk.) A fizikai környezeteket illetően az MI-rendszereknek nem lehet több kifogása. Továbbá egy teljességgel új környezet, az internet is létrejött.

Állapot	
Hogyan változik a világ?	Hogyan néz ki most a világ?
Mit okoznak a cselekvésem?	Hogyan fog kinézni, ha az A cselekvést hajtom végre?
Hasznossá	Milyen boldog leszek egy ilyen állapotban?
	Milyen cselekvéseket kell most végrehajtanom?
:	

27.1. ábra. Egy modell- és hasznosságalapú ágens, ahogyan először a 2.14. ábrán bemutattuk

*Nyomon követni a világ állapotát.* Ez az intelligens ágens számára szükséges legfontosabb képességek egyike, amelyhez egyaránt szükség van érzékelésre és a belső reprezentáció frissítésére. A 7. fejezetben ítéletlogikát alkalmaztunk a világ állapotának nyomon követésére; a 10. fejezetben kiterjesztettük ezt az elsőrendű logikára; a 15. fejezetben pedig **szűrő algoritmusokat** (*filtering algorithms*) mutattunk be, hogy a bizonytalan környezeteket is követni tudjuk. Ezekre a szűrőeszközökre van szükség, amikor igazi – tehát tökéletlen – érzékelésre kerül sor. A jelenlegi szűrő és érzékelési algoritmusokat lehet úgy kombinálni, hogy elég jó teljesítményt kapjunk alacsony szintű predikátumok leírásánál, mint például „a csésze az asztalon van”, de van még megejendő út előttünk, míg az algoritmusok azt is ki tudják jelenteni, hogy: „dr. Russell teázik dr. Norviggel.” Egy másik probléma az, hogy ugyan az approximációs szűrő algoritmusok képesek elég nagy környezeteket kezelni, lényegüket tekintve azonban még mindig *propozicionálisak*: a propozicionális logikához hasonlóan nem kezelik explicit módon az objektumokat és a relációkat. A 14. fejezetben bemutattuk, hogyan lehet ennek a problémának a megoldására elsőrendű logikát alkalmazni; elvárásaink szerint nagy előrelépéssel fog majd járni, ha a komplex környezetek kezelésére is alkalmazzák ezeket az elgondolásokat. Továbbá, mihelyt egy bizonytalan környezetben elkezdünk objektumokról beszélni, felmerül a **bizonytalan azonosság** (*identity uncertainty*) prob-

lémája: nem tudjuk, melyik objektum melyik. A logika alapú mesterséges intelligenciában meglehetősen elhanyagolták ezt a problémát, mivel általában feltételezték, hogy az észleletek állandó szimbólumokat tartalmaznak az objektumok azonosítására.

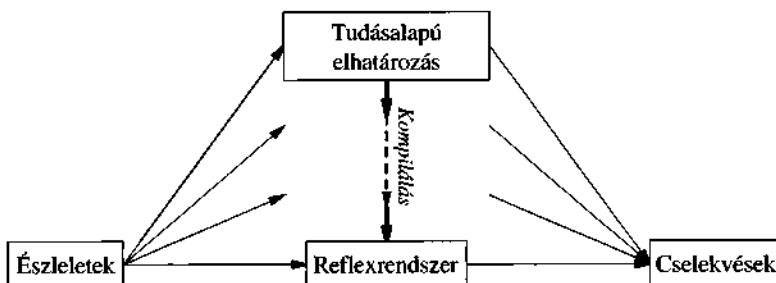
**Előrevetíteni, értékelni és kiválasztani jövőbeli cselekvésekét.** Az alapvető tudásreprezentációs elvárások itt is ugyanazok, mint a világ állapotának nyomon követésénél; az elsődleges nehézség a cselekvésekkel (mint például beszélgetni és teázni) való boldogulásból származik, amelyek esetenként az igazi ágens elemi lépéseinak ezreit vagy millióit tartalmazhatják. Az emberek is csak úgy tudnak ezzel megbirkózni, hogy a viselkedést egy **hierarchikus struktúrába (hierarchical structure)** szervezik. A 12. fejezetben bemutatott tervezési algoritmusok némelyike az ehhez hasonló méretű problémák kezelésére hierarchikus és elsőrendű logikai reprezentációt használ; másrészről pedig a 17. fejezetben megadott bizonytalansági döntéshozó algoritmusok lényegileg ugyanazon az elven alapulnak, mint a 3. fejezet állapotterbeli kereső algoritmusai. Láthatóan itt sok a tennivaló, talán a **hierarchikus megerősítéses tanulás (hierarchical reinforcement learning)** újabb fejleményei mentén.

**Hasznosság mint a preferenciák kifejezése.** A racionális döntéseket az elvárt hasznosság maximalizálására alapozni elméletileg teljesen általános, és megőv a tisztán célorientált megközelítések számtalan hibájától (mint például az ellentmondó célok és a bizonytalan elérésük). Mindeddig azonban kevés munka folyt a *realisztikus hasznossági függvények előállítása* terén – képzeliük el például az egymással kölcsönhatásban lévő preferenciák komplex hálózatát, amelyeket az emberek irodai asszisztenseként működő ágensnek meg kell értenie. Elég nehéznek bizonyult az összetett állapotokkal kapcsolatos preferenciákat dekomponálni az összetett állapotokról szóló bizonyosságok Bayes-hálós dekompozíciójához hasonlóan. Ennek az egyik oka az lehet, hogy az állapotok preferenciái valójában az állapotok múltbeli preferenciáiból lettek összeszerkesztve, amelyeket **jatalomfüggvények (reward functions)** írnak le (lásd 17. fejezet). A keletkező hasznossági függvény akkor is nagyon összetett lehet, ha a jatalomfüggvény egyszerű. Mindez azt sugallja, hogy komolyan kell vennünk a jatalomfüggvények tudásmérnöki konstrukcióját, mert ezzel tudjuk az ágens értésére adni, mit akarunk tőle.

**Tanulás.** A 18–20. fejezetben leírtuk, hogy az ágens tanulása miként fogalmazható meg az ágens különböző komponenseinek induktív (ellenőrzött, nem ellenőrzött vagy megerősítéses) tanulásaként. Nagyon hatékony logikai és statisztikai technikákat fejlesztettek ki, amelyek egész nagy problémákkal is meg tudnak birkózni, és gyakran elérik, sőt meghaladják az embernek azt a képességet, amellyel egy adott szótár feletti prediktív mintákat azonosít. Másrészről viszont abban a fontos problémában, hogy a be-meneti szótárnál absztraktabb szintű, új reprezentációkat hozzanak létre, a gépi tanulás kevés haladást ért el. Hogyan tud például egy autonóm robot létrehozni az emberektől meg nem kapott olyan új predikátumokat, mint például az *Iroda* és a *Kávé*? Hasonló megfontolások alkalmazhatók a viselkedés tanulására is: a *Teázás* fontos, magas szintű cselekvés, de hogyan kerül be egy olyan cselekvéskönyvtárba, ahol eredetileg csak jóval egyszerűbb cselekvések szerepeltek, mint például a *KezeitFelemleni* és a *Lenyelni*. Amíg nem értjük meg ezeket a problémákat, addig a hatalmas józan ész tudásbázisok kézi létrehozásának csüggesztő feladatával nézünk szembe.

## 27.2. ÁGENSARCHITEKTÚRÁK

Természetes, ha azt kérdezzük: „Melyik architektúrát alkalmazzam a 2. fejezetben bemutatottak közül?” A válasz: „Mindegyiket!” Láttuk, hogy a reflexszerű válaszok alkalmásak az időkritikus helyzetekben, míg a tudásalapú elhatározás lehetővé teszi, hogy az ágens előre tervezzen. Egy teljes ágensnek egy **hibrid architektúra** (*hybrid architecture*) segítségével képesnek kell lennie mindenktőre. A hibrid architektúra egyik fontos tulajdonsága, hogy a különböző döntési komponensek határa nem rögzített. A szerkesztés vagy **komplilálás** (*compilation*) például az elhatározás szintjén megjelenő deklaratív információt folyamatosan hatékonyabb reprezentációkká alakítja át, eljutva végül a reflexszerű szintre (27.2. ábra). (Ez a célja a 19. fejezetben bemutatott magyarázatalapú tanulásnak.) Az olyan rendszereknek, mint a SOAR (Laird és társai, 1987) és a THEO (Mitchell, 1990), pontosan ez a struktúrájuk. minden alkalommal, amikor explicit elhatározás révén jutnak el egy probléma megoldásáig, a megoldás általánosított verzióját is eltárolják a reflexszerű komponens részére. Ennek a folyamatnak a *fordítottját* kevésbé tanulmányozták még: a környezet változásakor a megtanult reflexek lehet, hogy többé már nem érvényesek, és az ágensnek vissza kell térnie a tudatos konstrukció szintjére, hogy új viselkedést hozhasson létre.



**27.2. ábra.** A komplilálás célja a megfontolt (deliberatív) döntéshozatal hatékonyabb, reflexszerű mechanizmussá való átalakítása

Egy ágensnek saját következtetéseit is az ellenőrzése alá kell tudnia vonni. A következtetést abba kell hagynia, ha cselekvésre van szükség, és a következtetésre rendelkezésre álló időt a legígéretesebb számításokra kell fordítania. Egy taxisofőr ágensnek, például, ha közvetlenül maga előtt balesetet észlel, nem félórák alatt, hanem a másodperc töredéke alatt el kell tudnia dönteni, hogy fékezni fog, vagy inkább kikerüli a balesetet. A másodperc ezen töredékét a legfontosabb kérdések vizsgálatára kell fordítania, hogy balra, illetve jobbra szabad-e a sáv, és nincs-e közvetlenül mögötte egy nagy teherautó, és nem arra, hogy milyen az idő, mennyire kopnak a gumik, vagy hogy hol találja a következő utast. Az ilyen problémákkal a **valós idejű MI** (*real-time AI*) foglalkozik. Ahogy az intelligens rendszerek egyre bonyolultabb tárgyterületekre vonulnak be, minden probléma valós idejűvé válik, hiszen az ágensnek soha nem lesz elegendő ideje arra, hogy a döntési problémát egzakt módon oldja meg.

Világos, hogy sürgető szükség van az általánosabb döntéshozó helyzetekben is működő módszerekre. Az utóbbi időben két kecsegtető módszertan is napvilágot látott. Az első az **akármikor algoritmusok** (*anytime algorithms*) alkalmazása (Dean és

Boddy, 1988; Horvitz, 1987). Egy akármikor algoritmusnál az algoritmus kimenetének minősége az idő műlásával fokozatosan javul, bármikor szakítjuk is meg tehát az algoritmust, ésszerű döntéshez jutunk. Az ilyen algoritmus vezérlését egy metadöntési eljárás végzi, amely meghatározza, hogy érdemes-e a számításokat tovább folytatni. Az akármikor algoritmusra egy egyszerű példa a kétszemélyes játékoknál ismertetett iteratív mélyítő keresés. Azonban bonyolultabb rendszerek – amelyek számos ilyen, együttesen működő algoritmusból állnak – is konstruálhatók (Zilberstein és Russell, 1996). A másik módszer a **döntéselméleti metakövetkeztetés (decision-theoretic meta-reasoning)** (Horvitz, 1989; Russell és Wefald, 1991; Horvitz és Breese, 1996). Ebben a megközelítésben az információérték elméletét (16. fejezet) a számítások megválasztására használjuk. A számítás értéke minden költségtől (a késleltetett cselekvés miatt), minden hasznosságától (a döntés javított minősége miatt) függ. A metakövetkeztetési sémaikkal jobb keresési algoritmusokat lehet előállítani, automatikusan garantálva az akármikor tulajdonságot. A metakövetkeztetés természetesen költséges, de a komplikáltas arra is felhasználható, hogy a túlmunka költsége eltörpüljön a számítás költsége mellett, lévén hogy a számítás is felügyelt.

A metakövetkeztetés csupán egyik aspektusa az általános **reflektív architektúrának (reflective architecture)**: ezek az architektúrák lehetővé teszik a magán az architektúrán belül keletkező számítási entitások és cselekvések fölötti tudatosságot. A reflektív architektúrák elméleti alapja úgy építhető fel, hogy a környezet állapotából és magának az ágensnek a számítási állapotából álló, közös állapotteret definiálunk. Döntési eljárásokat és tanulási algoritmusokat lehet úgy tervezni, hogy ebben a közös állapotterben működjenek, ezzel segítve és javítva az ágens számítási teljesítményét.

Végső soron azt várjuk, hogy az olyan feladatspecifikus algoritmusok, mint amilyen az alfa-béta keresés és a hátrafelé láncolás, eltűnnék az MI-rendszerekből, továbbá, hogy ezek olyan általános módszerekkel lesznek helyettesíthetők, amelyek az ágens számításait jó minőségű döntések hatékony létrehozása felé irányítják.

## 27.3. EGYÁLTALÁN A JÓ IRÁNYBA HALADUNK?

Az előző fejezetekben sok előrelépést soroltunk fel, és még több lehetőséget a jövőbeli fejlődésre. De hova vezet minden? Dreyfus analógiaként azt hozza fel (Dreyfus, 1992), hogy amikor valaki egy fa megnászással akar a Holdra jutni, az illető állandó haladásról számolhat be egészen a fa csúcsáig. Ebben a fejezetben az a kérdésünk, hogy a mesterséges intelligencia fejlődése a famászásra hasonlít-e, vagy inkább a holdrakétára.

Az 1. fejezetben azt mondta, hogy célunk a *racionálisan cselekvő* ágens építése. Azonban arra is figyelmeztettünk, hogy

...komplex környezetben a tökéletes racionálitást – minden helyesen cselekedni – lehetetlen elérni. A számítási szükségletek egyszerűen túl komolyak. A könyv nagyobb részében azonban azzal a munkahipotézissel fogunk élni, hogy a tökéletes döntéshozatal megértése jó kiindulópont.

Itt az ideje ismét végiggondolni, mi is az MI pontos célja. Ágenseket akarunk építeni, de milyen specifikáció van közben a fejünkben? Íme, négy lehetőség:

- Tökéletes racionáliság (perfect rationality):** A tökéletesen racionális ágens minden pillanatban úgy cselekszik, hogy a környezetről szerzett információja alapján várható hasznosságát maximalizálja. Mivel láttuk, hogy a legtöbb környezetben túlságosan időraklı a tökéletes racionálitáshoz szükséges számítások, ezért ez nem realizistikus cél.
- Számítható racionálitás (calculative rationality):** Ez az a megközelítés, amit implicit módon a logikai és döntéshozzájárulás tervezésekor használtunk. Egy számíthatóan racionális ágens *végül* meghatározza azt, ami racionális választás lehetett volna következtetéseinek az elején. Ez érdekes tulajdonsága egy rendszemek, de a legtöbb környezetben a rosszkorérkező jó válasznak nincs semmi értéke. A gyakorlatban az MI-rendszerek tervezői rá vannak kényszerítve arra, hogy a tervezési minőséget illetően egy elfogadható általános teljesítmény érdekében kompromisszumot hozzanak; szerencsétlen módon azonban a kalkulatív racionálitás elméleti alapja nem nyújt jól kidolgozott lehetőséget egy ilyen kompromisszum meghozatalára.
- Korlátozott racionálitás (bounded rationality):** Herbert Simon elvetette a tökéletes (vagy akár a közelítőleg tökéletes) racionálitás gondolatát, és a korlátozott racionálitást állította a helyére (Simon, 1957), ami a valódi ágensek döntéseinek leíró elmélete. A következő írta:

A valós világ-beli objektív racionális viselkedéshez, vagy akár csak ennek az objektív racionális viselkedésnek az elfogadható megközelítéséhez megoldandó problémák méretéhez képest túl kicsi az emberi elme komplex problémák megfogalmazására és megoldására szolgáló kapacitása.

Azt javasolta, hogy a korlátozott racionálitás elsődlegesen az **elfogadhatóságban (satisficing)** nyilvánul meg: olyan hosszú ideig kell foglalkozni a döntéssel, ami elelegendő egy „elfogadhatóan jó” válasz megtalálásához. Ezért a munkájáért Simon közgazdasági Nobel-díjat kapott, és később részletesen is írt még róla (Simon, 1982). Ez az emberi viselkedésnek sok esetben hasznosnak tűnő modellje, azonban nem ad formális specifikációt az intelligens ágensek számára, mert az elmélet nem adja meg az „elfogadhatóan jó” definícióját. Továbbá az elfogadhatóság láthatóan csak egyike a korlátozott erőforrások leküzdését szolgáló technikáknak.

- Korlátozott optimalitás, KO (bounded optimality, BO):** Egy korlátozottan optimális ágens *adott számítási erőforrások mellett* a lehető legjobban cselekszik. Egy ilyen ágens esetén az ágensprogram várható hasznossága legalább olyan magas, mint az ugyanazon a gépen futó bármilyen más ágensprogramé.

A négy lehetőség közül, úgy tűnik, a „korlátozott optimalitás” választása adja a legtöbb esélyt az MI erős elméleti megalapozására. Megvan az az előnye, hogy elérhető: minden van legalább egy legjobb program – a tökéletes racionálitásról ugyanez nem mondható el. A korlátozottan optimalitás ágensek hasznosak a minden nap gyakorlatban is, a számítható racionálitású ágensek általában nem, míg az elfogadhatóságra törekvő ágenseknél ez a szeszélyeiken műlik.

A hagyományos megközelítés az MI-ben az volt, hogy a számítható racionálitással kezdünk, majd az erőforráskorlátok kielégíthetősége érdekében kompromisszumokat hozunk. Ha ezek a korlátok csak kis problémákat okoznak, akkor a keletkező terv várhatóan a KO ágens tervéhez lesz hasonló. Ha azonban az erőforráskorlátok kritikussá válnak – például mert a környezet egyre összetettebbé válik –, akkor a két terv várhatóan el fog tértani egymástól. A korlátozott optimalitás elmélete ezen korlátok kezelésére kínál elveket.

A korlátozott optimalitásról egyelőre keveset tudunk. Nagyon egyszerű gépekhez és valamelyen mértékben korlátozott környezetekhez korlátozottan optimális programok konstruálhatók (Etzioni, 1989; Russell és társai, 1993). Bonyolult környezetekben működő nagy, általános rendeltetésű számítógépek esetén azonban fogalmunk sincs, hogy milyenek legyenek a KO-programok. Ha a korlátozott optimalitás konstruktív elmélete valaha is megfogalmazható lesz, hinnünk kell, hogy a korlátozottan optimális programok tervezése nem fog túlságosan függni a használt számítógép részleteitől. A tudományos kutatást nagyon megnehezítené, ha például kiderülne, hogy egy gigabájtos memoriájú gép néhány kbájt memoriával való bővítése lényegi különbséget jelentene a KO-program tervében. Ahhoz, hogy ez ne történjen meg, a korlátozott optimalitás kritériumait kissé lazábban kell kezelni. Az aszimptotikus bonyolultság (A) függelék mintájára definiálhatjuk az **aszimptotikus korlátozott optimalitást, AKO** (**asymptotic bounded optimality, ABO**) (Russell és Subramanian, 1995). Tegyük fel, hogy egy  $P$  program egy  $M$  gép esetén korlátozottan optimális a környezetek **E** osztályában, ahol a környezetek komplexitása nem korlátozott. Ekkor a  $P'$  program egy AKO-program  $M$  részére **E**-ben, ha egy  $P$  programot egy  $kM$  gépen futtatva a teljesítményében túlszármazalja, és ahol a  $kM$  gép  $k$ -szor gyorsabb (vagy nagyobb), mint  $M$ . Ha  $k$  nem bizonyulna óriásnak, elégedettek lehetnének, ha nemtriviális környezetek és nemtriviális architektúrák esetén AKO-programokkal rendelkezünk. Kevés ráció lenne abban, hogy óriási erőfeszítések árán inkább KO-, mint AKO-programokat keressünk, ha a használható gépek nagysága és sebessége, úgy tűnik, egy adott idő alatt konstans mértékben úgyis nő.

Megkockázthatjuk azt is, hogy a KO- vagy az AKO-programok – nagy teljesítőképességű számítógépek és bonyolult környezetek esetén – nem fognak szükségszerűen egyszerű és elegáns struktúrával rendelkezni. Látuk már, hogy az általános rendeltetésű intelligenciához bizonyos fokú reflexszerű és tudatos képesség szükséges, ami a tudás és a döntéshozatali formák sokaságát, ezen formák mindegyikéhez a tanuló és a komplilálási mechanizmusokat, a következetés vezérlésének módszereit és a tárgyterületre vonatkozó specifikus tudás nagy tárát jelenti. A korlátozottan optimális ágensnek adaptálódnia kell ahhoz a környezethez, amelyben tartózkodik, így belső szervezettsége végül az adott környezetre jellemző optimalizálási sémákat tükrözheti. Ez elvárható, hasonlóképpen ahhoz, ahogy a súlyukban és a lőerőben korlátozott versenyautók szélsőségesen komplikált konstrukcióig jutottak el. Az a sejtésünk, hogy a korlátozott optimalitáson alapuló mesterséges intelligencia tudománya nagy részben olyan folyamatok tanulmányozásából fog állni, amelyek révén egy ágensprogram a korlátozott optimalitáshoz fog konvergálni, és talán kevésbé fog az eredményül kapott, nehezen áttekinthető programok részleteivel törődni.

Összegezve, a korlátozott optimalitás koncepcióját jól definiált és kivitelezhető kutatási feladatnak javasoljuk a mesterséges intelligencia kutatása számára. A korlátozott optimalitás az optimális *programokat* specifikálja, és nem az optimális *cselekvést*. A cselekvést úgyis a programok generálják, és a programok azok, amelyek felett a tervező befolyással rendelkezik.

## 27.4. MI VAN, HA AZ MI SIKERREL JÁR?

David Lodge az irodalomkritikusok egyetemi világáról szóló *Small world (Kis világ)* (Lodge, 1984) c. novellájában a főhős megdöbbenést okoz, amikor a kiváló, ámde egymással vitatkozó tagokból álló irodalomelméleti bizottságtól megkérdezi: „*Mi van, ha igazuk van?*” Az elméletalkotók egyike sem gondolt még erre a kérdésre (valószínűleg azért, mert megcáfoltatlan elméleteken vitatkozni csak öncél lehet). Néha hasonló zavart okozhatunk, ha MI-kutatóktól kérdezünk azt: „*Mi van, ha sikerrel jársz?*” A mesterséges intelligencia izgalmas dolog, az intelligens számítógépek pedig nyilvánvalóan jobbak, mint buta társaik, miért hát az aggodalom?

Ahogy a 26.3. alfejezetben felvetettük, bizonyos etikai kérdéseket is meg kell fogni. Az intelligens gépek sokkal többre képesek, de vajon erekjüköt jára vagy rosszra fordítják? A mesterséges intelligencia kifejlesztésén fáradozók felelőssége, hogy megbizonyosodjanak munkájuk pozitív hatásáról. Hogy a hatás mennyire lesz széles körű, az a siker mértékétől függ. Már a mesterséges intelligencia mérsékelt sikere is változásokat hozott a számítógép-tudomány tanításában (Stein, 2002) és a szoftverek fejlesztésében. Az MI olyan új alkalmazásokat tett lehetővé, mint például a beszédfelismerő, készlet-ellenőrző vagy megfigyelő rendszerek, robotok és keresőgépek.

A mesterséges intelligencia közepes méretű sikere várhatóan befolyásolná a legkülönbözőbb emberek minden napjai életét. A számítógépesített kommunikációs rendszerek (mint például a mobiltelefonok és az internet) már eddig is mélyreható változásokat okoztak a társadalomban, a mesterséges intelligencia esetében azonban ez még várat magára. Könnyű elképzelni, hogy a tényleg hasznos otthoni és irodai személyi asszisztensek komoly pozitív hatással lennének az emberek életére, habár rövid távon valószínűleg némi gazdasági zavart is okozna. Az ilyen fejlettségi szintű technológiai kapacitás azonban alkalmazható lenne autonóm fegyverek létrehozására is, és ez a legtöbbek szerint nemkívánatos eredmény lenne.

Végül pedig valószínűnek tűnik, hogy a mesterséges intelligencia nagyarányú sikere – az emberihez hasonló vagy annál magasabb szintű intelligencia létrehozása – megváltoztatná az emberiség többségének életét. Átalakulna munkánk és játkunk belső természe, csakúgy, mint az intelligenciáról és a tudatosságról vallott nézeteink, sőt megváltozna az emberi faj jövőbeli sorsa is. Ezen a szinten az MI-rendszerek közvetlen fenyegetést jelenthetnének az emberi autonómiára, szabadságra, sőt akár a fennmaradásunkra is. Mindezek miatt a mesterséges intelligencia kutatásától nem választhatók el az etikai következményei.

Mit hoz majd a jövő? A sci-fi írók láthatóan jobban kedvelik a negatív utópiákat, mint a pozitívakat, valószínűleg azért, mert ez ad lehetőséget az érdekesebb cselekményekre. De a mesterséges intelligencia mindeddig más forradalmi technológiákhoz (nyomtatás, csővezetékek, légi utazás, telefónia) hasonlóan viselkedett: pozitív aspektusaik ellensúlyozták negatív hatásaikat.

Összefoglalásul azt mondhatjuk, hogy láttuk, a mesterséges intelligencia mekkora haladást tett meg rövid története alatt, de ma is érvényes Alan Turing *Computing Machinery and Intelligence* c. esszéjének zárómondata:

Csak rövidtávolságra látunk előre, de azt látjuk, hogy sok a tennivaló.

# A) MATEMATIKAI ALAPOK

## A1. BONYOLULTSÁGANALÍZIS ÉS AZ O() JELÖLÉS

A számítógépes szakembereknek gyakran szembe kell nézniük azzal a feladattal, hogy előöntsék, két algoritmus közül melyik fut gyorsabban, vagy melyik igényel kevesebb memóriát. Ezt a feladatot kétféle módon lehet megközelíteni. Az első a **teljesítménymérés (benchmarking)** – ahol az algoritmusokat számítógépen futtatjuk, és megmérjük, milyen a sebességük másodpercenkben, valamint a memória fogyasztásuk bájtokban. Hiszen végül is ez az, ami igazából számít. A teljesítménymérés azonban nem biztos, hogy kielégítő mérték, mivel igen specifikus: egy konkrét program teljesítményét méri, amely program egy konkrét programozási nyelven íródott, konkrét gépen fut, konkrét fordító és bemeneti adatok mellett. A teljesítménymérő program által szolgáltatott egyszerű adatokból nehéz lenne arra következtetni, hogy az algoritmus milyen jól vizsgázik más fordító, más számítógép vagy más adathalmaz esetén.

### Aszimptotikus analízis

A második megközelítés az **algoritmusok matematikai elemzésén** (analysis of algorithms) alapul, konkrét implementációtól és bemenetektől függetlenül. Ezt a megközelítést egy példa kapcsán, egy számsorozat összeadó program esetén fogjuk ismertetni:

```
function Összegzés(szekvencia) returns egy szám
    összeg ← 0
    for i ← 1 to Hossz(szekvencia)
        összeg ← összeg + szekvencia[i]
    end
    return összeg
```

Az elemzés első lépése a bemenetektől való eltekintés, olyan paraméter vagy paraméterek megkeresése, amely vagy amelyek jellemzők a bemenet méretére. Jelen példában a bemenetet a szekvencia hosszával jellemzhetjük, aminek a jele legyen  $n$ . Második lépés az implementációtól való eltekintés, azaz egy olyan mérték megkeresése, amely tükrözi az algoritmus futási idő-igényét, de nem kötődik egy konkrét fordítóhoz vagy

számítógéphez. Az ÖSSZEGZÉS program esetén ez lehetne például a végrehajtott kód sorainak a száma. A mérték részletesebb is lehetne, külön specifikálva az algoritmus által végrehajtott összeadások, értékkedésök, tömbhivatkozások és elágazások számát. Mindkét esetben megkapjuk az algoritmus által végzett lépések teljes számának a jellemzését a bemenet nagyságának függvényében, amit  $T(n)$ -nek fogunk nevezni. Ha megszámoljuk a kód sorait, a példánk esetében az eredmény  $T(n) = 2n + 2$ .

Ha minden program olyan egyszerű lenne, mint az ÖSSZEGZÉS, az algoritmusok elemzése triviális lenne. Két probléma miatt a helyzet azért bonyolultabb. Először is, ritka eset olyan paramétert találni, mint az  $n$ , amely teljes egészében jellemzi az algoritmus által végrehajtott lépések számát. A legjobb, amivel ehelyett rendelkezhetünk, a legrosszabb esetet jelentő  $T_{\text{legrosszabb}}(n)$  vagy az átlagos  $T_{\text{átlag}}(n)$  számítása. Az átlag kiszámításához az elemzőnek fel kell tételeznie a bemenetek valamelyen eloszlását.

A másik probléma az, hogy az algoritmusok ellenállnak az egzakt elemzésnek. Ilyen esetben közelítésekre szükséges támaszkodnunk. Azt mondhatjuk, hogy az ÖSSZEGZÉS algoritmus  $O(n)$  – avagy nagy ordó  $n$  – jelezvén, hogy a mértéke, néhány kis értékű  $n$  kivételével, a legtöbb esetben  $n$  konstans-szorosa. Formálisan:

$$T(n) O(f(n)) \text{ mértékű, ha } T(n) \leq kf(n) \text{ valamelyen } k\text{-ra, minden } n > n_0 \text{ esetén.}$$

Az  $O(n)$  jelölés az **aszimptotikus analízist** (*asymptotic analysis*) jelenti. Külön vizsgálat nélkül azt mondhatjuk, hogy ha  $n$  aszimptotikusan közeledik a végtelenhez, egy  $O(n)$  algoritmus jobb, mint egy  $O(n^2)$  algoritmus. Egy egyedi teljesítménymérő adat ilyen kijelentést nem tudna alátámasztani.

Az  $O()$  jelölés a konstans szorzóktól eltekint, emiatt kevésbé precíz, viszont könnyebb használni, mint a  $T()$ -t. Így például egy  $O(n^2)$  algoritmus egy  $O(n)$  algoritmusnál hosszú távon minden rosszabb lesz, azonban ha a két algoritmus  $T(n^2 + 1)$  és  $T(100n + 1000)$ , akkor az  $O(n^2)$  algoritmus  $n \leq 110$ -re jobb lesz.

E hátránya ellenére az aszimptotikus elemzés az algoritmuselemzés legelterjedtebben használt módszere. Ez azért van így, mert az elemzés mind a műveletek egzakt számától (a  $k$  konstans tényezőt figyelem kívül hagyva), mind a bemenet egzakt tartalmától (csupán  $n$  nagyságát figyelembe véve) elvonatkoztat. Ily módon az elemzés matematikailag kivitelezhetővé válik. Az  $O()$  jelölés jó kompromisszum a precizitás és a könnyű elemzés között.

## NP és az inherensen nehéz problémák

Az algoritmusok elemzése és az  $O()$  jelölés lehetővé teszi, hogy egy konkrét algoritmus hatékonyságáról nyilatkozzunk. Arról azonban nem tudunk mondani semmit, hogy az adott problémához van-e nála jobb algoritmus. A **bonyolultságelmélet** (*complexity analysis*) területe nem az algoritmusokat, hanem inkább a problémákat elemzi. Az első nagy szakadék a polinom időben megoldható és a polinom idejű megoldást nélkülöző problémák között van, akármilyen algoritmust is használnánk. A polinom idejű problémák osztályát – azokét, amelyek  $O(n^k)$  időben oldhatók meg valamelyen  $k$  esetén – P-vel jelöljük. Ezeket néha „könnnyű” problémáknak hívjuk, mert az osztályhoz  $O(\log n)$  és  $O(n)$  futási idejű problémák is tartoznak. Idetartoznak azonban  $O(n^{1000})$  problémák is, a „könnnyű” elnevezést tehát túlságosan szó szerint venni nem lehet.

A problémák másik fontos osztálya az NP osztály, a nemdeterminisztikus polinom idejű problémák osztálya. Egy probléma akkor tartozik ide, ha létezik egy algoritmus, ami polinom időben meg tud találni egy megoldást, és még ellenőrizni is tudja, hogy a megtalált megoldás korrekt-e. Az ötlet az, hogy ha tetszőlegesen nagyszámú proceszszorral rendelkezünk ahoz, hogy az összes sejtést egyszerre próbáljuk ki, vagy pedig ha szerencsések vagyunk, és mindenkor előre eltaláljuk a helyes megoldást, az NP probléma P problémává válik. A számítógépes tudomány egyik nagy nyitott kérdése az, hogy az NP osztály ekvivalens-e a P osztályalakkor, ha nélkülözhető a végtelen processzorhalmaz, illetve a mindenkorudó sejtés luxusát. A legtöbb számítógépes szakember meg van győződve arról, hogy  $P \neq NP$ , vagyis, hogy az NP problémák inherens módon nehezek, és nincsenek számukra polinom idejű algoritmusok. Ezt azonban még soha sem bizonyították be.

Akik a  $P = NP$  eldöntése iránt érdeklődnek, azok figyelmébe az NP osztály egy alosztályát, az **NP-teljes** (**NP-complete**) problémák alosztályát ajánljuk. A *teljes* szó itt a „legvégsőt” jelenti, és az NP osztály legnehezebb problémáit jelöli ki. Azt bebizonyították már, hogy vagy minden NP-teljes probléma P-ben van, vagy egyik sem. Emiatt ez az osztály elméletileg érdekes, de gyakorlati jelentősége is van, mert sok fontos problémáról tudjuk, hogy NP-teljes. Egy lehetséges példa a Boole-kifejezések kielégíthetőségének problémája: egy adott logikai kifejezés esetén létezik-e az ítéletváltozónak olyan logikaiérték-hozzárendelése, hogy a kifejezés igaz legyen? Hacsak egy csodának nem leszünk tanúi, és a  $P = NP$  nem lesz igaz, nincs olyan algoritmus, amely minden kielégíthetőségi problémát polinomiális időben megold. Az MI azonban jobban érdekelt abban, hogy vajon léteznek-e hatékony algoritmusok *tipikus*, előre rögzített eloszlásokból sorsolt problémák esetén. Ahogy ezt a 7. fejezetben láttuk, vannak olyan algoritmusok, mint a WALKSAT, amelyek sok probléma esetén egészen jól teljesítenek.

A **co-NP** osztály az NP komplexitateban, hogy minden NP-beli döntési probléma esetén létezik egy neki megfelelő co-NP-beli probléma felcserélt „igen” és „nem” válaszokkal. Tudjuk, hogy a P mind az NP, mind a co-NP osztályok részhalmaza, és azt gondoljuk, hogy vannak olyan co-NP-beli problémák, amelyek P-nek nem elemei. A co-NP osztály legnehezebb problémái a **co-NP-teljes** problémák.

A #P osztály az NP-beli döntési problémáknak megfelelő számlálási problémák halmaza. Döntési problémáknak igen-nem a megoldásuk: például létezik-e megoldása ennek a 3-SAT mondatnak? A számlálási problémáknak egész számok a megoldásai; mennyi megoldása van ennek a 3-SAT mondatnak? Egyes esetekben a számlálási probléma a döntési problémánál sokkal nehezebb. Így például annak az eldöntése, hogy egy páros gráphoz létezik-e egy tökéletes párosítása  $O(VE)$  idejű (ahol a gráfnak V csúcsa és E éle van). A „hány tökéletes párosítása létezik egy konkrét páros gráf esetén” számlálási probléma viszont #P-teljes, amin azt kell érteni, hogy legalább olyan nehéz, mint a #P-beli problémák bármelyike, és így legalább olyan nehéz, mint az NP-beli problémák bármelyike.

A P-TÁR – a polinom-tárigényű – problémák osztályát szintén tanulmányozták még nemdeterminisztikus gép esetén is. Az az általános vélekedés, hogy a P-TÁR-nehéz problémák az NP-teljes problémáknál nehezebbek, bár kiderülhet, hogy  $NP = P\text{-TÁR}$ , mint ahogy az is kiderülhet, hogy  $P = NP$ .

## A2. VEKTOROK, MÁTRIXOK ÉS LINEÁRIS ALGEBRA

A matematikusok egy vektort egy vektortrélemként definiálnak. Mi azonban konkrétabb definíciót fogunk használni. Egy vektor az értékek egy rendezett szekvenciája. A kétdimenziós térben például olyan vektoraink vannak, mint:  $\mathbf{x} = \langle 3, 4 \rangle$  és  $\mathbf{y} = \langle 0, 2 \rangle$ . A szokásos konvencióhoz folyamodunk vastagon szedett betűket használva a vektormeveknek, bár egyes szerzők a nevek fölött nyilat vagy vonalat húznak:  $\vec{x}$  vagy  $\bar{y}$ . Egy vektor elemeire indexekkel lehet hivatkozni:  $\mathbf{z} = \langle z_1, z_2, \dots, z_n \rangle$ .

A vektorokon értelmezett két alapvető művelet a vektorok összeadása és egy skalárral való szorzása. Az  $\mathbf{x} + \mathbf{y}$  vektorösszeadás az elemenkénti összegzés:  $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$ . A skalárszorzás minden egyik elemet egy konstans értékkel szorozza be:  $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$ .

Egy vektor hosszát az  $|\mathbf{x}|$  jelöli, amelynek értéke egyenlő a vektor elemeinek négyzetösszegéből vett négyzetgyökével:  $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$ . Két vektor skalárszorzata,  $\mathbf{x} \cdot \mathbf{y}$  a megfelelő vektorelekemek szorzatának összege, azaz  $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$ , vagy konkréten esetünkben  $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$ .

A vektorokat sokszor egy  $n$ -dimenziós euklideszi tér irányított vonalszakaszainak (nyílaknak) képzeljük. Ekkor a vektorösszeadás annak felel meg, mintha az egyik vektor kezdő pontját a másik vektor végpontjához illeszténénk, az  $\mathbf{x} \cdot \mathbf{y}$  skalárszorzat viszont  $|\mathbf{x}||\mathbf{y}|\cos\theta$ -vel lesz egyenlő, ahol  $\theta$  az  $\mathbf{x}$  és az  $\mathbf{y}$  közötti szög.

Egy mátrix az értékek négyzögletes elrendezése sorokba és oszlopokba. Itt egy  $3 \times 4$  méretű  $\mathbf{m}$  mátrixot látunk:

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \end{pmatrix}$$

Az  $m_{i,j}$  első indexe a sort, a második az oszlopot jelöli. Programozási nyelvekben  $m_{i,j}$ -t gyakran  $\mathbf{m}[i][j]$ , vagy  $\mathbf{m}[i][j]$ -nek írják.

Két mátrix összegét a megfelelő elemek összeadásával definiáljuk:  $(\mathbf{m} + \mathbf{n})_{i,j} = m_{i,j} + n_{i,j}$ . (Az összeg különböző méretű  $\mathbf{m}$  és  $\mathbf{n}$  mátrixokra nincs definiálva.) Egy mátrixnak egy skalárral történő szorzatát szintén definiálhatjuk:  $(c\mathbf{m})_{i,j} = cm_{i,j}$ . A mátrixszorzat (két mátrix szorzata) már bonyolultabb. Az  $\mathbf{mn}$  szorzat csak akkor definiált, ha az  $\mathbf{m}$  mátrix  $a \times b$ , az  $\mathbf{n}$  mátrix pedig  $b \times c$  méretű (azaz a második mátrixnak annyi sora van, mint amennyi oszlopa az első mátrixnak). Az eredmény egy  $a \times c$  méretű mátrix. Ez azt jelenti, hogy a mátrixszorzat nem kommutatív: általánosságban  $\mathbf{mn} \neq \mathbf{nm}$ . Ha a mátrixok megfelelő méretűek, akkor az eredmény:

$$(\mathbf{mn})_{i,k} = \sum_j \mathbf{m}_{i,j} \mathbf{n}_{j,k}$$

Az  $\mathbf{I}$  egységmátrix  $I_{i,j}$  elemei egységnyiek, ha  $i = j$ , különben nullák. Az egységmátrix tulajdonsága, hogy  $\mathbf{mI} = \mathbf{m}$ , minden  $\mathbf{m}$ -re. Az  $\mathbf{m}$  transzponáltját,  $\mathbf{m}^T$ -t, megkapjuk, ha az  $\mathbf{m}$  sorait és oszlopait felcseréljük, formálisan:  $\mathbf{m}^T_{i,j} = \mathbf{m}_{j,i}$ .

A mátrixokat lineáris egyenletrendszerek megoldására a **Gauss–Jordan–eliminációs** (**Gauss–Jordan elimination**) algoritmust használjuk, ami egy  $O(n^3)$  komplexitású algoritmus. Tekintsünk az alábbi egyenletrendszert, amelynek  $x$ ,  $y$  és  $z$  megoldására vagyunk kíváncsiak:

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

Ezt az egyenletrendszer kifejezhetjük mátrixos formában is:

$$\begin{array}{cccc} x & y & z & c \\ \left( \begin{array}{cccc} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -1 \\ -2 & 1 & 2 & -3 \end{array} \right) \end{array}$$

Itt az  $x, y, z, c$  nem része a mátrixnak, csak emlékeztetőül szolgál. Tudjuk, hogy ha egy egyenlet minden két oldalát egy konstanssal megszorozzuk, vagy ha két egyenletet egymáshoz hozzáadunk, akkor szintén egy érvényes egyenlethez jutunk el. A Gauss-Jordan-elimináció ezeket a lépéseket ismételten alkalmazza úgy, hogy először az első változót (az  $x$ -et) elimináljuk, az elsőt kivéve az összes egyenletből, majd folytatjuk minden  $i$ -re, az  $i$ -edik változót minden egyenletből eliminálva, az  $i$ -edik egyenletet kivéve. Az  $x$ -et a második egyenletből úgy elimináljuk, hogy az első egyenletet  $3/2$ -vel megszorozzuk, az eredményt pedig a második egyenlethez hozzáadjuk. Ezzel az alábbi mátrixhoz jutunk el:

$$\begin{array}{cccc} x & y & z & c \\ \left( \begin{array}{cccc} 2 & 1 & -1 & 8 \\ 0 & 0,5 & 0,5 & 1 \\ -2 & 1 & 2 & -3 \end{array} \right) \end{array}$$

Folytatjuk így eliminálva az  $x$ -et, az  $y$ -t és a  $z$ -t, míg végül a:

$$\begin{array}{cccc} x & y & z & c \\ \left( \begin{array}{cccc} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array} \right) \end{array}$$

mátrixot kapjuk, amely jelzi, hogy a megoldás  $x = 2, y = 3, z = -1$ . (Próbálja ki!)

## A3. VALÓSZÍNÜSÉGI ELOSZLÁSOK

A valószínűség az eseményhalmazokon definiált olyan mérték, amely az alábbi három axiómát elégíti ki:

1. minden esemény mértéke egy 0 és 1 közötti szám. Ezt a következőképpen írjuk:  $0 \leq P(E = e_i) \leq 1$ , ahol az  $E$  egy eseményt képviselő véletlen változó, és az  $e_i$ -k az  $E$  lehetséges értékei. Általánosságban a véletlen változókat nagybetűkkel, az értékeket pedig kisbetűkkel jelöljük.
2. Az egész halmaz mértéke 1; azaz  $\sum_{i=1}^n P(E = e_i) = 1$

3. Diszjunkt halmazok uniójának a valószínűsége az egyedi események valószínűségeinek összege, azaz  $P(E = e_1 \vee E = e_2) = P(E = e_1) + P(E = e_2)$ , ahol  $e_1$  és  $e_2$  diszjunktak.

Egy **valószínűségi modell (probabilistic model)** a kölcsönösen egymást kizáró, lehetőséges kimenetelek mintateréből áll, és minden kimenetel valószínűségi mértékéből. A holnapi időjárás modelljében például a kimenetelek a következők lehetnek: *napos, felhős, esős vagy havas*. Ezen kimenetelek egy részhalmaza képvisel egy eseményt. A csapadékesemény például az  $\{\text{esős}, \text{havas}\}$  részhalmaz.

A  $\langle P(E = e_1), \dots, P(E = e_n) \rangle$  értékekből képzett vektor jelölésére a  $\mathbf{P}(E)$ -t fogjuk használni. A  $P(e_i)$  jelölést szintén használni fogjuk a  $P(E = e_i)$  rövidítésére, hasonlóan a  $\sum_{i=1}^n P(E = e_i)$ -t a  $\sum_e P(e)$ -ként rövidítjük.

A  $P(B|A)$  feltételes valószínűséget a  $P(B \cap A)/P(A)$  definiálja. A és B feltételesen függetlenek, ha  $P(B|A) = P(B)$  [vagy ekvivalens módon  $P(A|B) = P(A)$ ]. Folytonos változók esetén az értékek halmaza végtelen, és csak pontszerű tüskék nincsenek, minden egyik érték valószínűsége 0. Emiatt egy **valószínűségi sűrűségfüggvényt (probability density function)** definiálunk, amit szintén  $P(X)$ -szel fogunk jelölni.  $P(X)$  jelentése a  $P(A)$  diszkrit valószínűségi függvénytől kissé eltér. A  $P(X = c)$  sűrűségfüggvény-érték az a valószínűség, hogy az  $X$  a  $c$  körülí intervallumba esik, elosztva az intervallum hosszával, és határátmenetet képezve, ahogy az intervallum hossza tart a 0-hoz:

$$P(X = c) = \lim_{dx \rightarrow 0} P(c \leq X \leq c + dx)/dx$$

A sűrűségfüggvénynek nemnegatívnak kell lennie minden  $x$ -re, és teljesítenie kell, hogy:

$$\int_{-\infty}^{\infty} P(X)dx = 1$$

Definiálhatunk egy **valószínűségi eloszlásfüggvényt (cumulative probability density function)**,  $F(X)$ -et is, ami annak a valószínűsége, hogy a véletlen változó  $x$ -nél kisebb:

$$F(X) = \int_{-\infty}^x P(Z)dz$$

Jegyezzük meg, hogy a valószínűségi sűrűségfüggvénynek létezik egysége, az eloszlásfüggvény viszont egység nélküli. Ha az  $X$ -et például másodpercben mérjük, akkor a sűrűséget Hz-ben (azaz 1/s) fogjuk. Ha  $X$  a méterben mért háromdimenziós tér egy pontja, akkor a sűrűséget  $1/m^3$ -ben mérjük.

Az egyik legfontosabb valószínűség-eloszlás a **Gauss-eloszlás (Gaussian distribution)**, amit **normális eloszlásnak (normal distribution)** is szokás nevezni. A  $\mu$  közép-értékű és  $\sigma$  szórású ( $\sigma^2$  varianciájú) Gauss-eloszlás definíciója:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$

ahol  $x$  egy folytonos változó  $-\infty$ -től  $+\infty$ -ig terjedő értékkészlettel.  $\mu = 0$  és  $\sigma^2 = 1$  mellett kapjuk a **standard normális eloszlás (standard normal distribution)** speciális esetét. Egy  $d$  dimenziós  $x$  vektor feletti eloszlás a **többváltozós Gauss-eloszlás (multivariate Gaussian distribution)**:

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}((\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu))}$$

ahol  $\mu$  az eloszlás középértékvektora,  $\Sigma$  pedig a **kovarianciamátrix (covariance matrix)**.

A standard normális eloszlás egydimenziós eloszlásfüggvénye:

$$F(x) = \int_{-\infty}^x P(x)dx = \frac{1}{2}(1 + \text{erf}\left(\frac{x - \mu}{\sigma\sqrt{2}}\right))$$

ahol az  $\text{erf}(x)$  az ún. **hibafüggvény (error function)**, amelynek zárt alakja nincs.

A **központi határeloszlás tétele (central limit theorem)** azt mondja, hogy  $n$  véletlen változó középértékének az eloszlása a normális eloszláshoz tart, ha  $n$  tart végtelenhez. Igaz ez a véletlen változók majdnem minden halmazára, hacsak valamelyik véges alhalmaz varianciája a többöt nem fogja dominálni.

## Irodalmi és történeti megjegyzések

A számítógépes tudományokban ma oly elterjedten használt  $O()$  jelölést (nagy ordó) P. G. H. Bachmann német matematikus vezette be a számelmélettel kapcsolatban (Bachmann, 1894). Az NP-teljesség koncepcióját Cook (Cook, 1971) dolgozta ki, míg a problémák egymásra redukálásának modern módszertanát Karpnak köszönhetjük (Karp, 1972). Munkásságukért mindenketen Turing-díjban – a számítógépes tudományok legmagasabb elismerésben – részesültek.

Az algoritmusok elemzésére és tervezésére irányuló klasszikus munkák között találjuk a (Knuth, 1973; Aho és társai, 1974) műveket. Újabb adalékok Tarjantól (Tarjan, 1983) és a Cormen, Leiserson és Rivest szerzőhármastól (Cormen és társai, 1990) származnak. Ezekben a művekben a hangsúlyt a kezelhető problémákat megoldó algoritmusok tervezésére és elemzésére helyezték. Az NP-teljességnek, illetve a kezelhetetlenség más formáinak az elméletébe a legjobb bevezetők (Garey és Johnson, 1979) és (Papadimitriou, 1994). Az elmélet tárgyalásán túlmenően Garey és Johnson meggyőzően, példákkal illusztrálják, hogy a számítógépes szakemberek miért a polinom és az exponenciális futási időigény határára teszik egyhangúan a kezelhető és kezelhetetlen problémák közötti határvonalat. A szerzők az NP-teljesként vagy más szempontból kezelhetetlenek ismert problémák terjedelmes katalógusát is mellékelik.

A valószínűség-számításról jó művek (Chung, 1979; Ross, 1988; Bertsekas–Tsitsiklis, 2002; Feller, 1971).

# B. MEGJEGYZÉSEK A NYELVEKRŐL ÉS AZ ALGORITMUSOKRÓL

## B.1. NYELVEK DEFINIÁLÁSA BACKUS–NAUR–FORMÁBAN (BNF)

Jelen könyvben definiáltunk néhány nyelvet, az ítéletlogika nyelvét (258. oldal), az elsőrendű logika nyelvét (304. oldal) és az angol nyelv részhalmazát (921. oldal). Egy formális nyelvet mondathalmazként definiálunk, ahol egy-egy mondat a szimbólumok halmaza. A számunkra érdekes nyelvek mindegyike végtelen sok mondatból áll, következésképpen szükségünk van e halmaz tömör jellemzésére. A jellemzést egy **nyelvtan** (grammar) segítségével kíséreljük meg. Nyelvtanunkat **Backus–Naur–forma** (Backus–Naur form) vagy BNF-formalizmusban fejezzük ki. Egy BNF-nyelvtannak négy komponense van:

- A zárószimbólumok vagy a **terminálisok** (*terminal symbols*) halmaza. Ezekből a szimbólumokból, másiképpen szavakból, állnak össze a nyelv mondatai. Ezek lehetnek betűk (A, B, C, ...) vagy szavak (a, aardvark, abacus, ...).
- A nyelv részkifejezéseit kategorizáló **nemzáró szimbólumok** vagy **nemterminálisok** (*nonterminal symbols*) halmaza. Az angol nyelvben például a *FőnéviKifejezés* (*NounPhrase*) nemzáró szimbólum a „you” és a „the big slobbery dog” mondatokat is tartalmazó végtelen halmazt jelenti.
- A **kezdő szimbólum** (*start symbol*) a nyelv teljes mondatait jelentő nemzáró szimbólum. Az angol nyelv esetén ez a *Mondat* (*Sentence*), az aritmetika esetén ilyen szimbólum lehet a *Kifejezés* (*Expr*).
- A **képzési szabályok** (*rewrite rules*) halmaza, ahol egy-egy szabály *LHS* → *RHS* formájú. *LHS* egy nemzáró szimbólum, az *RHS* pedig 0 vagy több (záró vagy nemzáró) szimbólumból állhat.

A

*Mondat* → *FőnéviKifejezés* | *IgeiKifejezés*

alakú képzési szabály azt jelenti, hogy ha bármikor is van két, egy *FőnéviKifejezés*-ként és egy *IgeiKifejezés*-ként kategorizált mondatunk, akkor azokat összekapcsolhatjuk, és az eredményt *Mondat*-ként kategorizálhatjuk. A | szimbólum az alternatív jobb oldali kifejezéseket választja szét. Nézzük most az egyszerű aritmetikai kifejezések BNF-nyelvtanát:

<i>Kifejezés</i>	$\rightarrow$ <i>Kifejezés Operátor Kifejezés   ( Kifejezés )   Szám</i>
<i>Szám</i>	$\rightarrow$ <i>Számjegy   Szám Számjegy</i>
<i>Számjegy</i>	$\rightarrow$ <i>0 1 2 3 4 5 6 7 8 9</i>
<i>Operátor</i>	$\rightarrow$ $+   -   \div   \times$

A nyelvekkel és a nyelvtanokkal bővebben a 22. fejezetben foglalkoztunk. Előfordul, hogy más forrásokban a BNF-re kissé módosított jelölést használnak. Nemzáró szimbólum esetén *{Számjegy}*-gyel találkozhatunk a *Számjegy* helyett, a zárószimbólum esetén 'szó'-val a szó helyett, illetve szabályokban a  $:$  : = jelöléssel a  $\rightarrow$  helyett.

## B2. AZ ALGORITMUSOK LEÍRÁSA PSZEUDOKÓDDAL

A könyvünkben több mint 80 algoritmust definiáltunk részletesen. Ahelyett hogy egy programozási nyelvet választottuk volna (és ahelyett, hogy azt kockáztattuk volna, hogy a nyelvet nem ismerő olvasó elveszti a fonalat), úgy döntöttünk, hogy pszeudokóddal írjuk le az algoritmusokat. Ezek legtöbbje érthető lesz azoknak, aik Javát, C++-t, Lispet vagy hasonló nyelveket használnak. Helyenként azonban matematikai formulákat, illetve minden nap nyelvet használunk olyan részek leírásához, amelyek máskülönben nehezebben lennének olvashatók. Néhány egyéni vonásról is szót kell ejtenünk.

**Statikus változók.** A static kulcsszót egy olyan változó megjelölésére használjuk, amely kezdeti – vagy a rákövetkező értékkedásból adódó – értékét a függvény első hívásánál kapja meg, és megtartja a függvény minden azt követő hívásánál. A statikus változó tehát hasonlít egy globális változóra abban, hogy a függvény egyedi hívását túlélí, értéke azonban csak a függvényen belül férhető hozzá. A könyv ágensprogramjai statikus változókat „memória” létesítésére használnak. A statikus változókat kezelő programok „objektumként” implementálhatók olyan objektumorientált nyelvekben, mint a Java vagy a Smalltalk. A funkcionális nyelvekben az implementálásuk könnyen oldható meg a lambda-kifejezés végrehajtásával egy olyan környezeten belül, amelyben a szükséges változót definiáltuk.

**Függvények mint értékek.** A függvények és eljárások jelölésére nagybetűs neveket, a változók jelölésére pedig kis-, dőlt betűs neveket használunk. Egy függvényhívás legtöbb esetben tehát  $FN(x)$  képet nyújt. Megengedjük azonban, hogy egy változó függvényértéket is felvehessen. Ha az  $f$  értéke például a négyzetgyökfüggvény, akkor az  $f(9)$  által visszaadott érték a 3.

**A tömbök 1-től kezdődnek.** Ha másképpen nincs specifikálva, egy tömb első indexe mindig 1, ahogy ez a matematikában szokás, és nem 0, mint a Javában vagy a C-ben.

**A beljebb szedett bekezdés fontos.** A beljebb szedett bekezdést a hurok vagy a feltételes kifejezés hatáskörének megjelölésére használjuk, mint például a Python nyelvben, ellentétben a Javával és a C++-szal (ahol a zárójelezés használatos), illetve a Pascallal és a Visual Basicnel (ahol egy end használatos).

### B3. ONLINE SEGÉDNYÚJTÁS

A könyvben található algoritmusok többségét online kódtárnakban implementáltuk, amelynek címe:

 [aima.cs.berkeley.edu](mailto:aima.cs.berkeley.edu)

Ha Önnek megjegyzései lennének a könyünkkel kapcsolatban, hibákat vett észre, illetve ha ötletei vannak, amelyekkel a könyv javítható, keressen meg bennünket. Kérjük, látogassa meg a weboldalunkon a vitafórumunkat, vagy küldjön e-mail üzenetet a következő címre:

 [aima@cs.berkeley.edu](mailto:aima@cs.berkeley.edu)

# IRODALOMJEGYZÉK

- Aarup, M., Arentoft, M. M., Parrod, Y., Stadler, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. and Zweben, M. (Eds.), *Knowledge Based Scheduling*. Morgan Kaufmann, San Mateo, California.
- Abramson, B. and Yung, M. (1989). Divide and conquer under global constraints: A solution to the N-queens problem. *Journal of Parallel and Distributed Computing*, 6(3), 649–662.
- Ackley, D. H. and Littman, M. L. (1991). Interactions between learning and evolution. In Langton, C., Taylor, C., Farmer, J. D., and Ramussen, S. (Eds.), *Artificial Life II*, pp. 487–509. Addison-Wesley, Redwood City, California.
- Aarup, M., Arentoft, M. M., Parrod, Y., Stadler, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. And zweben, M. (Eds.), *Knowledge Based Scheduling*.Morgan reactions between learning and evolution. In Langton, C., Taylor, C., Farmer, J. D., and Ramussen, S. (Eds.), *Artificial Life II*. pp. 487–509. Addison-Wesley, Redwood City, California.
- Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Mathematical Surveys*, 25, 221–262.
- Adelson-Velsky, G. M., Arlazarov, V. L., and Donskoy, M. V. (1975). Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, 6(4), 361–371.
- Agmon, S. (1954.) The relaxation method for linear inequalities. *Canadian J. Math.*, 6(3), 382–392.
- Agre, P. E. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 268–72, Milan. Morgan Kaufman.
- Aho, A. V., Hopcroft, J., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.
- Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Upper Saddle River, New Jersey.
- Alt-Kaci, H. and Podelski, A. (1993). Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3–4), 195–234.
- Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential functions method in pattern recognition learning. *Automation and Remote Control*, 25, 821–837.
- Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97, 270–277.
- Aldous, D. and Vazirani, U. (1994). „Go with the winners” algorithms. In *Proceedings of the 35th Annual Symposiumon Foundations of Computer Science*, pp. 492–501, Santa Fe, New Mexico. IEEE Computer Society Press.
- Allais, M. (1953). Le comportement de l'homme rationnel devant la risque: critique des postulats et axiomes de l'école Américaine. *Econometrica*, 21, 503–546.
- Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, 26(11), 832–843.
- Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23, 123–154.
- Allen, J. F. (1991). Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6, 341–355.
- Allen, J. F. (1995). *Natural Language Understanding*. Benjamin/Cummings, Redwood City; California.
- Allen, J. F., Hendler, J., and Tate, A. (Eds.). (1990). *Readings in Planning*. Morgan Kaufmann, San Mateo, California.