1. A formális logika, a nulladrendű logika szintaxisa, logikai operátorok

Mi az a formális logika?

- A 19. században a logika a filozófiát elhagyva elkezdett a matematikával összefonódni. Így született meg a **formális logika**, ami **az egyes állításokat** formálisan, **logikai jelek segítségével írja le**.

Mire használják?

- A **processzorok** áramköreinek, "kapuinak" **működési elve**.
- Hardverek és szoftverek ellenőrzése, a lehetséges hibák megkeresése. (**Hardver- és szoftververifikáció**)
  - Pl.: Annak bizonyítása, hogy egy While ciklus sosem futhat a végtelenségig.

Milyen logikák léteznek?

- **Nulladrendű logika**
  - Minden objektum **igaz/hamis** típusú
  - **Logikai változók** összekapcsolása logikai műveletekkel
- **Elsőrendű logika**
  - **Logikai állítások** megfogalmazása **tetszőleges objektumok**kal.
  - A változók objektumokat vehetnek fel értékül, amiket logikai függvények vizsgálnak. Ezek vannak logikai műveletekkel összekötve.
- **Többértékű és Fuzzy logikák**
  - Létezik **háromrendű logika**, ami a 0 és 1 mellett **egy harmadik, bizonytalan érték**et is bevezet.
  - A **Fuzzy** logikák **0 és 1 között bármilyen értéket megengednek**. (A paksi atomerőmű bizonyos döntéshozó rendszerei is Fuzzy logikát használnak.)
- **Tempoláris logikák**
  - Kezelik az **időrendiség**et (fontosak a hardver- és szoftververifikációkban.)

A Nulladrendű logikai nyelv

- Nulladrendű logika, vagy **ítéletlogika**
- **Igaz és hamis kijelentések**kel foglalkozik, minden objektuma ilyen típusú.

A nyelv kiválasztása formalizáláskor

- A formalizálandó problémát **elemi állításokká (literálokká)** kell bontanunk, majd ezekhez ítéletváltozót kell rendelnünk.
- Az ítéletváltozókból felépítünk egy, az állításnak megfelelő logikai függvényt.

A formula definíciója

- Logikai kijelentés
- **Adott ítéletváltozóknak egy véges, nem üres V halmaza**
  - Ha A ítéletváltozó és **A**$\in$**V** , akkor A formula
  - Ha A formula, akkor ¬**A** is formula
  - Ha **A és B formulák**, akkor a következő kifejezések is formulák:
    - **Konjunkció**: (A$\wedge$B)
    - **Diszjunkció**: (A$\vee$B)
    - **Implikáció**: (A$\Rightarrow$B)
    - **Ekvivalencia**: (A$\Leftrightarrow$B)
    - **Kizáró vagy**: (A$\oplus$B)
  - Minden formula a fenti szabályok **véges sokszori** alkalmazásával áll elő.
  - **Rekurzív** definíció, a definiálandó fogalom (formula) definíciójában hivatkozunk magára a fogalomra. A definíció utolsó sora megtiltja a végtelen rekurziót.

Precedencia és asszociativitás

Precedencia	**Operátor**	**Név**	**Asszociatív?** (Több érték esetén a zárójelek számítanak?)
1	¬	Negáció	-
2	$\wedge$,$\vee$	Konjunkció, Diszjunkció	igen
3	$\Rightarrow$	Implikáció	nem
4	$\Leftrightarrow$,$\oplus$	Ekvivalencia, Antivalencia	igen

2. A Nulladrendű logika szemantikája, igazságtábla

A logikai operátorok szemantikája igazságtáblával

A	B	A∧B	A∨B	A⇒B	A⇔B	A⊕B
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	0	0	1
1	1	1	1	1	1	0

A	¬A
0	1
1	0

Interpretáció

- Egy interpretáció **minden ítéletváltozó értékét lerögzíti**, azaz mindegyikőjükhöz hozzárendel egy 0 vagy 1 értéket.
- **Ha egy** V ítéletváltozó-készlettel rendelkező **formulának adunk egy** olyan **interpretációt**, ami V összes elemének értéket ad, akkor **a formula értékelhetővé válik**.
- **F értékét az I interpretációban** így jelöljük: $\mathbf{[[F]]_I}$
- **Definíció:**
  - **Ítéletváltozóknak** egy véges, nem üres V halmaz interpretációjának egy $I:V \mapsto \{0,1\}$ függvényét nevezünk.

Formulák csoportosítása (törvény, ellentmondás, következmény)

Logikai törvény (Tautológia)	Logikai ellentmondás	Kielégíthető formula	Logikai következmény
Bármely interpretációja **igaz** végeredményt ad.	**Bármely** interpretációja **hamis** végeredményt ad.	**Létezik** olyan interpretáció, amelyben a formula **igaz** értéket ad.	**Egy formulából következik-e egy másiknak a bekövetkezése.** A Premisszák akkor következményei a Konklúziónak, ha nem létezik olyan interpretáció, ahol az összes Premissza igaz, a konklúzió pedig hamis.

Egy F formula logikai törvény, ha **minden I interpretáció esetén $[[F]]_I=1$.**	Egy F formula akkor logikai ellentmondás, ha **minden I interpretációban $[[F]]_I= 0$.**	Egy F formula kielégíthető, ha **létezik olyan I interpretáció, hogy $[[F]]_I=1$.**	A $P_1,\ldots,P_n$ formulák logikai következménye a C formula, ha **minden I interpretáció esetén ha $[[P_1]]_I=\cdots=[[P_n]]_I=1$, akkor $[[C]]_I=1$.**
Pl.: $A\vee\neg A\vee B$	Pl.: $A\wedge\neg A\wedge B$	Pl.: $A\wedge\neg A\vee B$	

3. Logikai azonosságok, normálformák

Nevezetes azonosságok

$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$; $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	**Asszociativitás** (a zárójelek felcserélhetők)
$P \wedge Q \equiv Q \wedge P$; $P \vee Q \equiv Q \vee P$	**Kommutativitás** (az értékek felcserélhetőek)
$P \wedge P \equiv P$; $P \vee P \equiv P$	**Idempotencia**
$(P \wedge Q) \vee Q \equiv Q$; $(P \vee Q) \wedge Q \equiv Q$	**Abszorptivitás** (elnyelés)
$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$; $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	**Disztributivitás**
$P \wedge 1 \equiv P$, $P \wedge 0 \equiv 0$; $P \vee 1 \equiv 1$, $P \vee 0 \equiv P$	**Korlátosság**
$P \wedge \neg P \equiv 0$; $P \vee \neg P \equiv 1$	**Ellentmondástalanság, Kizárt harmadik elve**
$\neg\neg P \equiv P$	**Dupla tagadás**
$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$; $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$	**de Morgan** azonosságok
$P \Rightarrow Q \equiv \neg P \vee Q$; $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$	**Implikáció átírása**
$P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$	**Kontrapozíció**
$P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$; $P \oplus Q \equiv \neg(P \Leftrightarrow Q)$	**Ekvivalencia és XOR átírása**

Egyszerűsítő szabályok

- asszociativitás
- idempotencia
- abszorptivitás
- ellentmondástalanság
- kizárt harmadik elve
- korlátosság

Literál

- A literál egy **ítéletváltozó** (X) vagy annak negáltja (¬X).

Klóz (Elemi diszjunkció)

- **Elemi diszjunkciónak** vagy klóznak nevezünk egy $L_1 \vee L_2 \vee \cdots \vee L_k$ formulát, ahol $\forall L_i$ literál és $k \geq 0$.
- **Diszjunkcióval összekötött literálok**
- A 0 literálból álló klóz értéke 0

- **Cube** (Elemi konjunkció)
  - **Elemi konjunkciónak** vagy cube-nak nevezünk egy $L_1 \wedge L_2 \wedge \cdots \wedge L_k$ formulát, ahol $\forall L_i$ literál és $k \geq 0$.
  - **Konjunkcióval összekötött literálok.**

KNF (Konjunktív normálforma)

- Egy formula KNF-ben van, ha $E1 \wedge E2 \wedge \cdots \wedge En$ alakú, ahol $\forall Ei$ klóz és $n \geq 0$.
- **Kizárólag konjunkciót és diszjunkciót tartalmaz**
- **A klózokat konjunkciók kötik össze.**

DNF (Diszjunktív normálforma)

- Egy formula DNF-ben van, ha $E1 \vee E2 \vee \cdots \vee En$ alakú, ahol $\forall Ei$ elemi konjunkció és $n \geq 0$.
- **Kizárólag konjunkciót és diszjunkciót tartalmaz**
- **A cube-okat diszjunkciók kötik össze.**

A normálformára hozás algoritmusa

- **Igazságtáblával**

1.	Állítsuk elő a formula **igazságfüggvény**ét (igazságtábláját)!	
	TDNF	**TKNF**
2.	**Válasszuk** ki a függvény azon **interpretációit**, amiben **az eredménye igaz**!	**Válasszuk** ki a függvény azon **interpretációit**, amiben **az eredménye hamis**!
3.	**Minden egyes interpretációból előállítunk egy igaz klózt** (Úgy negáljuk az értékeket, hogy mindegyik 1 legyen).	**Minden egyes interpretációból előállítunk egy hamis cube-ot** (Úgy negáljuk az értékeket, hogy mindegyik 0 legyen).
4.	A kapott klózokat **összefűzzük diszjunkcióval**	A kapott cube-okat **összefűzzük konjunkciókkal**

- **Egyszerűsítési szabályokkal**

Az eljárás során végig ahol csak tudunk, **egyszerűsítsünk**!	
1.	**XOR** átírása
2.	**Ekvivalenciák** átírása
3.	**Implikációk** átírása
4.	**Negációk bevitele** az ítéletváltozókra de Morgan azonossággal
A Formula normálformában van, de hogy KNF-ben vagy DNF-ben, arra eddig nem volt ráhatásunk. Ha a másikat szeretnénk, **át kell alakítanunk**.	
Átírás	Konjunkciók és diszjunkciók felcserélése **asszociativitás**sal.

4. Tseitin transzformáció, Plaisted-Greenbaum kódolás

KNF/DNF problémái

- **A normálformára hozás** 3. tételben jegyzett **"naiv" formái exponenciális algoritmusok**, összetett formulák esetén **kezelhetetlen mennyiségű klózt vagy cube-ot hoznak létre**, ezért nem használják a gyakorlatban.

Tseitin transzformáció

- **Lineáris algoritmus** (nem növeli annyira a függvény hosszát, mint az előző eljárások).
- Kis formulák esetében ez a módszer is sok klózt állít elő, de **nagyobb formulák esetén kifizetődő a használata**.
- A formula összes nem literál **A részformuláját írjuk felül egy X ítéletváltozóval, majd írjuk le az X ⇔ A összefüggéseket konjunkciókkal elválasztva** (az 1. helyre mindig X1 kerül egyedül)!
- Példa:

$(R \Rightarrow P) \Rightarrow (\neg(Q \wedge R) \Rightarrow P)$

- $X1 \Leftrightarrow X2 \Rightarrow X3$
- $X2 \Leftrightarrow (R \Rightarrow P)$
- $X3 \Leftrightarrow (\neg X4 \Rightarrow P)$
- $X4 \Leftrightarrow (Q \wedge R)$

$X1 \wedge (X1 \Leftrightarrow X2 \Rightarrow X3) \wedge (X2 \Leftrightarrow (R \Rightarrow P)) \wedge (X3 \Leftrightarrow (\neg X4 \Rightarrow P)) \wedge (X4 \Leftrightarrow (Q \wedge R))$

- Ezt követően **klózokká kell alakítani a zárójelek közötti értékeket** és a formula máris **KNF**-ben van!

Plaisted-Greenbaum kódolás

- A **Tseitin** transzformáció **továbbfejlesztése**
- **Lecsökkenti a** létrehozott **klózok számát**
- **A** behelyettesített **literálok polaritásától függ** a használata.
- A segítségével **csökkenteni lehet az ekvivalenciák számát**.
- Adott $X1 \wedge (X1 \Leftrightarrow X2 \Rightarrow X3)$ részformula, ahol el **szeretnénk tüntetni az ekvivalenciát**.
  - Ha **az ekvivalencia baloldalán lévő literál csak negálatlanul szerepel a formula többi részében** (Az a konkrét literál, ami az ekvivalencia mögött van, ilyenkor nem számít), akkor az ekvivalencia egy **jobbra** mutató **implikációra** cserélhető!
  - Ha **az ekvivalencia baloldalán lévő literál csak negáltan szerepel a formula többi részében** (Az a konkrét literál, ami az ekvivalencia mögött van, ilyenkor nem számít), akkor az ekvivalencia egy **balra** mutató **implikációra** cserélhető!
  - Ha **az ekvivalencia baloldalán lévő literál negáltan és negálatlanul is szerepel a formulában** (Az a konkrét literál, ami az ekvivalencia mögött van, ilyenkor nem számít), az ekvivalencia nem hagyható el!

- FONTOS: **A kódolás feltételezi, hogy a vizsgált kódrészlet előtti kód már KNF-ben van**, így a haladásunkkal párhuzamosan KNF-re kell alakítanunk a kódot!

5. SAT, Rezolúció, DPLL

SAT és k-SAT probléma

- **SAT**
  - Satisfiability, **kielégíthetőség**
  - Egy fontos **alapprobléma**
    - Útvonal-optimalizálás
    - Órarend tervezés
  - Kérdése: **Egy adott KNF formula igaz interpretációjának keresése.**
  - **NP-teljes** probléma, **nem létezik hatékony algoritmus a megoldására**. A gyakorlatban csak nagy futási idejű algoritmusok ismertek, ezek közül kell a leghatékonyabbat megtalálni.

- **k-KNF (Limitált KNF)**
  - **Egy formula k-KNF-ben van, ha** $C_1 \wedge \cdots \wedge C_n$ alakú, ahol $\forall \mathbf{C_i}$ **olyan klóz, amely maximum k db. literált tartalmaz.**
  - **Bármely formula 3-KNF-re alakítható**, de ez nem csökkent a SAT-probléma nehézségén.
  - A **2-KNF**-re alakítható formulák SAT-problémája azonban már **nem NP-teljes**, létezik hatékony módszer a megoldásukra.

Rezolúció

  - A leggyakrabban használt megoldás **a SAT-probléma vizsgálatára.**
  - **Cáfoló eljárás**, tehát a bizonyítandó állítás negáltjából kell kiindulnunk.
  - **Az üres klóz levezetése** ilyenkor azt bizonyítja, hogy **a formula** kielégíthetetlen, vagyis **UNSAT.**
  - **Írjuk fel a klózokat** külön, **egy felsorolásos listába**!
  - Válasszunk ki két **rezolválható klóz**t és alkossuk meg a **rezolvens**üket!
    - **Két klóz akkor rezolválható, ha szerepel bennük ugyanaz a P érték különböző polaritással.**
    - A **rezolvens** úgy keletkezik, hogy **eltávolítjuk a két P értéket, a klózok maradékát** pedig **összefűzzük.**
    - $A \vee \neg B$ és $\neg A \vee B$ klózok nem rezolválhatóak (**csak akkor rezolválható, ha egy érték rezolválható!**)
  - **Ezt addig ismételjük, amíg elő nem áll az üres klóz**, vagy ki nem próbáltunk minden lehetséges kombinációt
  - Ha van **üres klóz**, a formula **UNSAT.**
- **Logikai törvény vizsgálata**
  - **A formula akkor logikai törvény, ha ¬A kielégíthetetlen!**
    - Ha a KNF-re hozott **¬A** rezolúciójakor **üres klóz**t kapunk, akkor **A logikai törvény**!
- **Logikai következmény vizsgálata**
  - **A premisszáknak akkor következménye a konklúzió, ha** $\mathbf{P_1 \wedge \cdots \wedge P_k \Rightarrow C}$ **logikai törvény!**
    - **Negálnunk kell a formulát**, amiből az **egyszerűsítés** után $\mathbf{P_1 \wedge \cdots \wedge P_k \wedge \neg C}$ lesz.
    - Ezt a formulát KNF-re alakítjuk és rezolváljuk.
    - Ha kijön az **üres klóz**, akkor a konklúzió logikai **következmény**e a premisszáknak!

DPLL

- **A rezolúció** továbbfejlesztése **ötvözve a visszalépéses kereséssel.**
- Törekszik a **minél kevesebb rezolúciós lépés**re.
- A legtöbb mai **SAT-szolver** ezen az algoritmuson alapul.
- **Visszalépéses keresés** (Backtracking)
  - **Problémamegoldó** algoritmus
  - A kiindulóponttól indul, a **válaszutaknál random választ irányt**. **Ha zsákutcába jut**, visszamegy a legutolsó válaszúthoz és **egy másik irányba indul**.
  - **Ha egy válaszúton az összes lehetőséget bejárta**, akkor az egyel **előző válaszútig megy vissza**.
  - Mivel **minden lehetőséget megvizsgál**, ezért **ha** a problémának **van megoldás**a, akkor az algoritmus **meg fogja találni**.
- **Adott egy formula:**

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

- **Algoritmus:**
  - A DPLL algoritmus addig ismétlődik, míg nem jut eredményre
  - 2 paramétere van, S-az eredeti klózhalmaz és I-a jelenlegi interpretáció

DPLL(S, I)

1) Ha mindegyik klóz kepott értéket és az egyik hamis, akkor RETURN
2) Ha mindegyik klóz kepott értéket és mind igaz, akkor KILÉPÉS, eredmény = S SAT
3) DÖNTÉS: Válassz egy X változót, amihez még nem szerepel I-ben hozzárendelt érték! (ha az egyik nem vezet eredményre, a másikat is próbáld ki!)
   1. DPLL(S, I∪{X})
   2. DPLL(S , I∪{¬X})
4) Ha I üres, akkor KILÉPÉS, eredmény = S UNSAT

Unit propagáció

- **Unit**: egy literált tartalmazó klóz
- Ha a rezolválandó **klózhalmazban vannak unitok**, **érdemes azokkal kezdeni a rezolválást,** hogy **leegyszerűsítsük a klózokat**. **Ez a unit propagáció**

DIMACS formátum, SAT szolverek (MiniSAT, CryptoMiniSAT)

- A SAT szolverek többsége egy **KNF klózhalmazt vár inputnak**.
- Használt **formátum: DIMACS**
  - **Szintaktika:**
    - **Az ítéletváltozókra a sorszámaikkal hivatkozunk (1-től kezde)**
    - A **negáció** jele a **negatív előjel**
    - A **diszjunkciók a szóközök**
    - A **klózokat a 0 végjellel zárjuk**
    - Szokás egy **fejléc** megadása `p cnf <változó_db> <klóz_db>` alakban.
  - Példa:
    - $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5)$

```
p cnf 5 5
1 -2 -3 0
-1 2 4 0
-2 3 -4 0
-1 -3 5 0
1 3 5 0
```

6. SMT

- **A DIMACS hátrányai:**
  - **Nehezen olvasható**
  - "Bőbeszédű"
    - Az összetettebb formulák leírása **nagy fájlméret**et eredményez.

Nagy rendszerek verifikációjára

- Fel kell írni **a rendszer**t, **mint logikai formulát**. (S)
- Meg kell fogalmazni egy **feltételt**, aminek a teljesülését vizsgáljuk (Pl.: nincs túlcsordulás, vagy 0-val való osztás) (C)
- Azt kell megvizsgálnunk, hogy **S rendszerben C feltétel mindig teljesül-e (**$S \wedge \neg C$).
  - A SAT szolver **kétféle válasz**t adhat:
    - **Kielégíthető**
      - Létezik olyan állapot, ahol ¬C nem teljesül, **a rendszer hibás**!
    - **Kielégíthetetlen**
      - A ¬C sosem teljesül, a **rendszer hibátlan**!
- **Bounded Model Checking**
  - Van, hogy S rendszer **több állapoton megy keresztül**.
  - Ennél a módszernél meg kell adnunk, hogy **hány állapotig gondolkodunk** előre (k>0)
  - A vizsgálandó formula: $\mathbf{S0 \wedge S1 \wedge \cdots \wedge Sk-1 \wedge \neg C}$
  - Ha nem az egész rendszert fedtük le, akkor előfordulhatnak hibák, mivel **a módszer csak k-1 iterációig vizsgál**t.

Az SMT

- A DIMACS-nál **magasabb szintű leíró nyelv**
- Kezeli az **összetett feladatok**at: számok, számtípusú változók, aritmetikák(+; -), összetett adatszerkezetek (tömb, lista).
- T**ámogatott elméletek**:
  - Egész számok
  - Valós számok
  - Bit-vetkorok
  - Tömbök
- **Példa:** Találjunk olyan x és y értéket, amire teljesül a feltétel: $x \geq 3 \wedge (x \leq 0 \vee y \geq 0)$

        - Ugyanúgy **alkalmazható a disztributivitás**: $(x \geq 3 \wedge x \leq 0) \vee (x \geq 3 \wedge y \geq 0)$
        - **Logikai úton nem úton egyszerűsíteni, de az egész aritmetika elméletével igen**: $(x \geq 3 \wedge x \leq 0)$ kiesik, mert nem teljesülhet.
        - $(x \geq 3 \wedge y \geq 0)$ lehetséges megoldása: x=3; y=0
- **Logikák**
    - **QF** - kvantormentes
        - Nincs értelmezve az egzisztenciális és univerzális operátor
    - **IA** - Egész aritmetika
        - Egész számokat értelmez és olyan operátorokat használ, amik eredménye mindig egész (nincs osztás)
    - **RA** - valós aritmetika
        - Valós számokat értelmez
    - **L** - lineáris
        - A szorzás csak (* c x) alakban szerepelhet, ahol c egy szám, x pedig egy változó. Ugyanez kell teljesüljön az osztásra is.
    - **N** - nemlineáris
        - Nincs a lineáris megkötése
- Pl.: **QF_LIA - Kvantormentes, lineáris, egész aritmetika**

Az SMT-LIB

- Az **SMT szolverek input formátuma**
- **Minden utasítás kerek zárójelben** van
- **Prefix formátum**, a műveletet közöljük először (+ x y)
- **A megjegyzéseket pontosvessző után** kell írni
- **Az** egyes **elemeket szóközzel választjuk el**
- **Fontosabb utasítások:**
    - **set-logic**
        - A háttér elmélet és logika megadása
    - **declare-const**
        - változó deklarálása (csak egyszer kaphat értéket és nem módosulhat, így **konstans**)
        - **assert**
            - állítás megadása
        - **check-sat**
            - A fölötte megadott assert-ek kielégíthetőségének vizsgálata
        - **get-model**
            - megadja az assert-eket kielégítő interpretációt
- **Fontosabb operátorok**

Jel	Leírás
+; -	Unárisként **előjel**; binárisként **összeadás/kivonás**
*; /; div; mod	szorzás; hagyományos, egész és maradékos osztás
=; distinct; <; >; <=; >=	relációk; **distinct**= nem egyenlő
and; or; not; =>(implikáció)	logikai operátorok
abs	Abszolútérték
ite	If-then-else - ?:

```
; Integer arithmetic
(set-logic QF_LIA)

(declare-const x Int)
(declare-const y Int)

(assert (= (+ x (* 2 y)) 20))
(assert (= (- x y) 2))

(check-sat)
(get-model)
```

SAT (SMT) szolverek

- **Z3**
  - AMicrosoft programja
- **CVC4**
  - Amerikai egyetemek közös projektje
- **Yices**

7. Elsőrendű logika szintaxisa

Az elsőrendű logika definíciója:

- Más néven **predikátumlogika**
- **Definíció:** Az elsőrendű logikai nyelv egy olyan $\langle Var,Pr,Fn\rangle$ formális 3-as, ahol
  - Var: változók (megszámlálható) halmaza
  - Pr: predikátumok halmaza, $Pr \neq \emptyset$
  - Fn: függvények halmaza
  - ahol minden $Pn \in Pr$ predikátumhoz és $fn \in Fn$ függvényhez adott egy $n \geq 0$ paraméterszám.
- **A nulladrendű logika kibővítése** paraméteres literálokkal (predikátumokkal), függvényekkel, konstansokkal.

Predikátumok

- **Paraméteres kijelentések logikai értékkel**
- **Definíció**: Egy P szimbólum akkor és csak akkor predikátum, ha P egy n>=0 paraméteres kijelentést jelöl, melynek alakja $P(t_1, t_2, \ldots, t_n)$, ahol t szimbólumok objektumot(termet) jelölnek
- **Példa:**
  - Szeretem a bort
  - Szeretem a sört
    - Nulladrendű logikában a két állítás teljesen független: B és S
    - Elsőrendű logikában felfedezhető kapcsolat a kettő között, az állítás ugyanaz, csak különböző tárgyakkal: Sz(B) Sz(S), ahol az Sz() predikátum, a B és S pedig term.
- Nem csak **konstansok**, hanem **változók** is lehetnek a **predikátumok paraméterei**.
- A predikátumoknak lehet **több paramétere** is.

Kvantorok

- A kvantorok **változót kötnek** (matematikában pl.: $\sum$)
- **Precedenciájuk a negációval egyenlő.**
- Elsőrendű logikában **két kvantor van:**
  - **Univerzális** kvantor ($\forall$)
    - Minden
  - **Egzisztenciális** kvantor($\exists$)
    - Létezik, van olyan
- A kvantorok szintaxisa a példákban látható.
- **1. Példa:** Ádám minden csokit szeret
  - $\forall x(Cs(x) \Rightarrow Sz(Á,x))$ - Minden x változóra igaz, hogyha x egy csoki, akkor Ádám szereti.
- **2. Példa**: Van olyan zöltség, amit Béla is megeszik.
  - $\exists x(Z(x) \wedge E(B,x))$ - Van olyan x változó, ami zöldség és Béla is megeszi.
- **A kvantorokat** a programozásba **függvényekként** lehet implementálni.

        - Az **univerzális** kvantor akkor ugrik ki és ad hamis értéket, hogy talál olyan esetet, amikor **a feltétel nem teljesül**
        - Az **egzisztenciális** kvantor akkor ugrik ki és ad igaz eredményt, ha talál olyan esetet, amire **a feltétel teljesül.**
    - A kvantorok **hatásköre** az utánuk **következő Predikátum**, vagy **az utánuk következő zárójel tartalma.**
- **A kötött változók** azok, amiknek **a kvantorok meghatározzák az értéküket**.
    - Egy változót **egyszerre csak egy kvantor köthet**. Bár szintaktikailag helyes, ha egy Kvantor hatáskörén belül egy másik kvantor ugyanazt a változót köti, igyekezzünk ezt elkerülni!
    - Lehetőség van **átnevezni a kötött változókat**. Ezt egyaránt a változónál és a kvantornál is meg kell tenni!
    - **Egy szabad változó bármilyen konstans értéket fölvehet (a Domainből),** legyen az konkrét érték, függvény, vagy másik változó.
- **Atom**
    - Az atomi formula az elsőrendű logikában a **paraméteres predikátum**
    - Definíció: Adott egy $\langle Var,Pr,Fn\rangle$ elsőrendű logikai nyelv. Az atom egy $P(t_1,\ldots,t_n)$ alakú kifejezés, ahol $Pn\in Pr$ és minden t_i term.
- **Term**
    - Lehet **változó, konstans, vagy függvényhívás** (a konstans egy 0 paraméteres függvényhívással egyenlő).
    - A termet **önmagával definiáljuk.**
    - Definíció: Adott egy $\langle Var,Pr,Fn\rangle$ elsőrendű logikai nyelv. A term egy kifejezés, mely a következőképpen képezhető:
        - $x\in Var$ változó
        - $f(t_1,\ldots,t_n)$, ahol $f_n\in F_n$ és minden t_i term
        - Minden term a fenti szabályok véges sokszori alkalmazásával áll elő.
- **Formula definíció**
    - **Nulladrendű**:
        - Ha $A\in V$, akkor A formula.
        - Ha A formula, akkor $\neg A$ is formula.
        - Ha A és B formulák, akkor a következő kifejezések is formulák:
            - Konjunkció: $(A\wedge B)$
            - Diszjunkció: $(A\vee B)$
            - Implikáció: $(A\Rightarrow B)$
            - Ekvivalencia: $(A\Leftrightarrow B)$
            - Kizáró vagy: $(A\oplus B)$
        - Minden formula a fenti szabályok véges sokszori alkalmazásával áll elő.
    - **Az elsőrendű logika** kis változtatásokat tesz a definíción
        - A logikai nyelvet már nem csak a V halmaz írja le, hanem egy $\langle Var,Pr,Fn\rangle$ elsőrendű logikai nyelv.(Változó, Predikátum, Függvény)
        - A legrövidebb formula már nem csak egy ítéletváltozó lehet, hanem egy atom (=predikátum paraméterekkel).
        - Megjelennek a kvantorok.
    - **Elsőrendű definíció:**
        - Adott egy $\langle Var,Pr,Fn\rangle$ elsőrendű logikai nyelv. Formulának nevezzük a következő kifejezéseket:
            - Atom: P(t1,…,tn), ahol $Pn\in Pr$ és minden ti term.
            - Negáció: Ha A formula, akkor $\neg A$ is formula.

        - Ha A és B formulák, akkor a következő kifejezések is formulák:
            - Konjunkció: $(A \wedge B)$
            - Diszjunkció: $(A \vee B)$
            - Implikáció: $(A \Rightarrow B)$
            - Ekvivalencia: $(A \Leftrightarrow B)$
            - Kizáró vagy: $(A \oplus B)$
        - Ha $x \in Var$ és A formula, akkor ezek is formulák:
            - Univerzálisan kvantált: $\forall xA$
            - Egzisztenciálisan kvantált: $\exists xA$
        - Minden formula a fenti szabályok véges sokszori alkalmazásával áll elő.
- A **formula paraméterei** a benne található szabad változók.
    - Paraméteres formulákat elsősorban **fogalmak definiálásához** használunk
    - Pl.: **Metszi(x,y)** definíciója:$\neg(x=y) \wedge \exists z(Pont(z) \wedge RajtaVan(z,x) \wedge RajtaVan(z,y))$
        - Két egyenes metszi egymást, ha nem egyenlőek és van közös pontjuk
    - **A definíciók írásának szabályai:**
        - Definíció **bal oldalán minden változónak szabad változónak kell lennie**. Ezek **a definíció paraméterei.**
        - Definíció **jobb oldalán** csak **azok a változók lehetnek szabadok, amelyek a definíció bal oldalán is szerepelnek**. Minden egyéb változónak kötöttnek kell lennie.
    - Hasonlít a programozásban használt metódusok formális és aktuális paramétereire.

8. Elsőrendű logika szemantikája

- **Elsőrendű logika interpretációja**
  - Legtöbbször **már nem elég a formula eredményének megállapításához**, paraméterkiértékelésre is szükség van.
  - Alapvető jelentése itt is **az értékek fixálása**
  - **Domain definiálása**
    - **Azon értékek** köre, amit a változók és konstansok **felvehetnek**.
    - Az I interpretáció által definiált domaint D_I-vel jelöljük.
    - D={1,2,3,4}
  - **Konstansok interpretálása**
    - **A Domainből minden konstans értéket kap.**
    - [[c]]=2
  - **Predikátumok interpretálása**
    - Meg kell határozni, hogy az adott predikátum milyen paraméterekre tér vissza igaz és hamis értékekkel. **Minden lehetséges opciót le kell fednünk!**
    - [P(x)]]=1 ha x páros, egyébként 0
  - **Függvények interpretálása**
    - Meg kell adni, hogy **milyen paraméterekre milyen értékkel tér vissza.**
    - [[f(x)]]=5−x
- **Paraméterkiértékelés**
  - **Ha a formulának vannak paraméterei** (szabad változói), akkor azokhoz **egy-egy domain-beli elemet kell rendelni**. ($\sigma=(x\to 1,y\to 2,z\to 2)$)
  - $\sigma=(x_1\to d_1,\ldots,x_n\to d_n)$, ahol minden x_i változó és $d_i\in D$.

- **Formula értéke**
  - **Ha van egy I interpretáció és egy σ paraméterkiértékelés**, akkor **meg tudjuk mondani az elsőrendű F formula eredményét**.
    - **Jelölés: $[[F]]_{I,\sigma}$**; vagy csak [[F]], ha a két elhagyott adat egyértelmű
- **Term értéke**
  - **Egy term** nem logikai értékkel bír, **bármilyen értéket felvehet a Domainből**.
  - **Jelölés: $[[t]]_{I,\sigma}$**; vagy csak [[t]], ha a két elhagyott adat egyértelmű

- **Az elsőrendű formulák szemantikai tulajdonságai (törvény, ellentmondás,...)**
  - Bizonyításuk **sokkal nehezebb, mint nulladrendben**
  - **Logikai törvény definíciója:**
    - Egy F formula logikai törvény, ha minden I interpretáció és σ paraméterkiértékelés esetén [[F]]I,σ=1.
  - **Logikai ellentmondás definíciója:**
    - Egy F formula logikai ellentmondás, ha minden I interpretáció és σ paraméterkiértékelés esetén [[F]]I,σ=0.
  - **Kielégíthetőség definíciója:**
    - Egy F formula kielégíthető, ha létezik I interpretáció és σ paraméterkiértékelés, hogy [[F]]I,σ= 1.
  - **Logikai következmény definíciója:**

- A $P_1,\ldots,P_n$ formuláknak logikai következménye a C formula, ha minden I interpretáció és σ paraméterkiértékelés esetén ha $[\![P_1]\!]=\cdots=[\![P_n]\!]=1$, akkor $[\![C]\!]=1$. A P_i-ket premisszáknak, a C-t pedig konklúziónak nevezzük.

9. Normálformák az elsőrendű logikában

- **KNF és DNF elsőrendben**
  - A megoldás alapja ugyanaz, mint nulladrendű logikában, kibővítve plusz lépésekkel

Prenexizálás:

- **A Kvantorok kivitele a formula elejére.**
- A Prenexizálás akkor kezdhető meg, **ha a formulában nincsenek a konjunkciónál és diszjunkciónál magasabb rendű operátorok**.
- **Definíció:** Egy formula prenex normálformában van, ha $\bigcirc x_1 \cdots \bigcirc x_k$ (A) alakú, ahol $\bigcirc \in \{ \forall, \exists \}$ és A kvantormentes.
- Prenexizálás előtt győződjünk meg róla, hogy minden kötött paraméternek egyedi neve van. **(Változótiszta alak)**
- A Prenexizáláskor az **összes kvantor hatásköre az egész formulára ki fog terjedni**.
- **Kvantoros de Morgan** azonosságok:
  - $\neg\forall xA \equiv \exists x\neg A$
  - $\neg\exists xA \equiv \forall x\neg A$
- **Kvantorkiemelés:**
  - $\bigcirc xA \circ B \equiv \bigcirc x(A \circ B)$, ahol $\bigcirc \in \{ \forall, \exists \}$, $\circ \in \{ \wedge, \vee \}$ és $x \notin Par(B)$.
- A negált kvantort át kell alakítani!

Skolemizálás

- **Eltávolítja az egzisztenciális kvantorokat.**
- Az egzisztenciális kvantort ki kell törölni a függvényből, az **általa kötött változó helyére pedig be kell vezetni egy új függvényt**, ami az eltüntetett kvantor előtti (balra) univerzális kvantorok által kötött értékeket kapja paraméterként. Ha Legelöl van az egzisztenciális kvantor, akkor a változója értelemszerűen egy 0 paraméteres függvény, egy konstans lesz.

- **Klózokká alakítás**
  - A rezolválásra való előkészítésként **az egyes klózok elé vigyük be azokat a kvantorokat**, amik kötik a bennük lévő változókat.

A KNF-re hozás algoritmusa

1. **XOR/ekvivalencia/implikáció** eltüntetése, negációk bevitele
2. **Változótiszta alak**ra hozás
3. **Prenexizálás:** kvantorok előre (sorrendet tartva!)
4. **Skolemizálás:** egzisztenciális kvantorok eliminálása, melynek az eredménye egy $\forall x1 \ldots \forall xnA$ alakú formula
5. **Az A KNF-re hozása**: disztributivitás alkalmazása
6. **Klózokra bontás**

10. Rezolúció az elsőrendű logikában

Rezolúció az elsőrendű logikában

- Az **alapjai megegyeznek a nulladrendű logika rezolúciójával**, de mint ahogy minden más, ezt is ki kell bővíteni.
- **Unifikáció (behelyettesítés)**
  - Adott két rezolválható **klóz két atomja**, amik bár megegyeznek, **a paramétereik különböznek.** Ekkor unifikálnunk kell!
  - **Unifikációs algoritmus:**
    1. Ha A és B nem ugyanazzal a predikátumszimbólummal kezdődnek, akkor nincs unifikátor, és vége.
    2. $\sigma := \emptyset$
    3. Ha nincs eltérő karakter A-ban és B-ben, akkor a keresett unifikátor σ, és vége.
    4. Jelölje t_1, illetve t_2 az első eltérő karakteren kezdődő termet A-ban, illetve B-ben!
    5. Ha t_1 és t_2 egyike sem változó (azaz függvényszimbólummal kezdődnek), akkor nincs unifikátor, és vége.
    6. Ha t_1 változó, akkor vezessük be a következő jelöléseket: $x := t_1$ és $s := t_2$. Egyébként legyen $x := t_2$ és $s := t_1$.
    7. Occur Check: Ha s nem változó és $x \in Par(s)$, akkor nincs unifikátor, és vége.
    8. $\sigma := \sigma\{x \rightarrow s\}$: A σ összes termjén elvégezzük az $x \rightarrow s$ helyettesítést, illetve kiegészítjük σ-át vele.
    9. $A := A_\sigma$ és $B := B_\sigma$
    10. Menj vissza a 3-ra!

- **Röviden**: **A változó termek megfeleltethetők egy másik értéknek, így egyenlővé hozható a két atom.** A behelyettesítési táblázatot **(Unifikátort) az eredményklóz egészére alkalmazni kell**! Ha a két itemet nem lehet unifikálni, akkor rezolválni sem lehet őket!
- A rezolúció folyamata pusztán az unifikálással lett kiegészítve

- **Logikai törvény bizonyítása:**
  - Az A formula logikai törvény, ha ¬A-ból levezethető az üres klóz.
- **Logikai következmény bizonyítása:**
  - A P1,…,Pk formuláknak logikai következménye a C formula, ha $P1 \wedge Pk \wedge \neg C$-ből levezethető az üres klóz.

Rezolúciós stratégiák

- Céljuk **a rezolúció célirányosabbá tétele**.
- **Lineáris rezolúció**
  - A két rezolvens közül az egyik **(centrális klóz) mindig a legutolsó létrejött új klóz** kell legyen! **A mellékklóz bármilyen lehet.** Mindig eredményre vezet.
- **SLD rezolúció**
  - Gyorsabb, mint a Lineáris rezolúció, de **nem mindig vezet eredményre.**
  - Biztosan eredményre vezet, ha az össze klóz Horn-klóz

- A két rezolvens közül **a centrális klóz a legutolsó létrejött klóz kell legyen, a mellékklóz pedig csak az eredeti klózhalmazból jöhet!**

Horn-klóz

- Olyan klóz, ami **legfeljebb egy nemnegált literált tartalmaz**.

11. Prolog

- **Mi az a Prolog?**
  - A Prolog egy **deklaratív MI programozási nyelv**, elsősorban **logikai problémák** megoldására.
  - Egy **Tudásbázist és egy Célt kell deklarálni**. A cél állhat több részcél konjunkciójából.
-

```
1likes(sam,Food) :- indian(Food), mild(Food).
2likes(sam,Food) :- italian(Food).
3likes(sam,chips).
4
5indian(curry).
6indian(dahl).
7
8mild(dahl).
9
10italian(pizza).
11italian(spaghetti).
```

Prolog szintaxisa

- A **változónevek nagybetűvel** kezdődnek.
- A **programnyelv konstansokkal** dolgozik, a változóknak csak egyszer kell értéket adni.
- **A sorokat pontokkal kell zárni.**
- **A Prologban** az egyenlőségjel az unifikáció operátora, s**zámok összeadásához az „is” kulcsszót kell használnunk** helyette!
- A deklaratív mivolta miatt **először egy tudásbázist kell építenünk** (fentebb), **ez alapján fogja a program kiértékelni a logikai problémákat**.
  - **Kétféle adat fog itt szerepelni**
    - **Tények:** Igaz állítások (Pl.: 3. sor)
    - **Szabályok:** Más állításokra visszavezethető állítások. (Pl.: 2. sor)
  - Ez azt jelenti, hogy kijelentéseket teszünk. (5. sor, a curry indiai étel, 6. és 8 sor, a dahl indiai étel és nem csípős)
- **A ":-" jelentése: akkor, ha.**
  - Szintaxisa: **fej :- feltétel 1., feltétel 2., ..., feltétel n.**
  - **Aláhúzásjel (_) → név nélküli (bármely) változó**. (Egy sorban) egyszer használatos változók helyettesítésére szolgál, amiknek **nem vagyunk kíváncsiak az értékére, csak a jelenléte fontos.**
  - **A feltételek között konjunkció van.**
  - Az 1. sor jelentése: Sam szereti az ételt, ha az indiai és nem csípős(mild).
- A lekérdezéseket egy "?-" után tehetjük meg.
  - ?- likes(sam, pizza)
    - Az első sor nem érvényes rá, de a második sor igaz választ ad vissza.

Példa:

```
1fia(géza, kálmán).
2fia(géza, álmos).
3fia(kálmán, istván).
4fia(álmos, béla).
5apja(X,Y) :- fia(Y,X).
6nagyapja(X,Z) :- apja(X,Y), apja(Y,Z).
```

5. sor: X akkor apja Y-nak, ha Y a fia X-nek (csak az lett előre megadva, hogy ki kinek a fia)
6. sor: X akkor nagyapja Z-nek, ha van olyan Y, akinek X az apja és Z a fia.

A prolog és a logika kapcsolata

- **A Prolog kódok mindig átalakíthatók egy KNF, Horn-klózokból álló függvénnyé.**
- A szabályok **":-"** jele egy **balra mutató implikáció.**
  - fej <= felt1 ∧ felt2 --> fej ∨¬(felt1∧felt2) --> (fej ∨¬felt1 ∨ ¬felt2) --> mindig Horn-klóz!
- **A** szabályok közti **vessző** megfelel az **"and"**-nek
- A szabályok **fejében** lévő változókat **univerzális kvantor**, míg a **törzsben** lévő, **a fejben nem létező** változókat **egzisztenciális kvantor** köti.

Függvény írása Prologban

```
1összege(A,B,C) :- C is A+B.
2kivonás(A,B,C) :- C is A-B.
```

- Nem szó szerint vett függvény, hanem **egy háromparaméteres szabály**
- Az utolsó paraméter az eredmény.
- Ha C-nek konkrét számot adunk meg, akkor nem összeadja és kivonja az értékeket, hanem megnézi az egyenlőséget C és A+B között, majd egy bool értéket ad vissza.
- **Ha C-nek egy változót adunk meg, akkor egyszerűen megmondja az értékét.**

Farokrekurzió:

- A rekurzió egy **nagyon költséges eszköz** a programozásban, **a farokrekurzió a legoptimálisabb** futású rekurziófajta, mivel a háttérben ciklusként lehet kezelni
- A farokrekurzióban **a rekurzív hívás mindig az utolsó parancs az utasítástörzsben**.

Listakezelés Prologban

- A Prologban egy listát úgy definiálunk, hogy kapcsos zárójelek között megadjuk az értékeit.
- A Lista első elemét Fejnek, az összes többi elemét faroknak nevezzük
  - Ha szükséges, a fejre egy szűrőjellel (|) tudunk hivatkozni.
  - **Lineáris keresés implementálása (Egy adott szám eleme-e a listának?)**
    - **search(X, [X|_])** → Paraméterként kér egy értéket és egy listát, majd megvizsgálja, hogy az érték a lista első eleme-e
    - **search(X, [_|T]) :- search([X|T])**→ Ha az érték nem egyezik meg, akkor a függvény rekurzívan meghívja saját magát, paraméterként megadva a régi értéket és a régi lista farokrészét. A régi lista farokrésze szintén egy lista, melynek az első eleme, vagyis feje, a régi lista második eleme. Így tudunk végigmenni a lista összes elemén. Ez a farokrekurzió prezentálása is!
    - „True” vagy „False” értékkel fog visszatérni.
    - Meghívás példa: **search(3, [1,4,100,3,5]).**

- **Lista elemeinek szummázása**
  - A függvény paraméterül kap egy listát és egy változót, amibe a számértékkel visszatérhet. A predikátum önmagában csak logikai értékkel tud visszatérni!
  - **sum([X|T], SumIn, SumOut) :-** → Paraméterként kérünk egy listát, egy olyan változók, amit lehet mindig növelni és egy olyan változót, amivel majd kivisszük a végeredményt. (A kimenő és a bemenő változó külön szedése legtöbbször a farokrekurzió velejárója).
  - **SumIn2 is SumIn + X,** → Mivel a változók konstansok, az érték eltárolására egy új változót kell bevezetnünk.
  - **sum(T, SumIn2, SumOut).** → Farokrekurzióval újra meghívjuk a függvényt ezúttal paraméterként a régi lista farkát és a megnövelt belső változót átadva.
  - **sum([], S, S).** → Alapeset felvétele, üres lista elemeinek ez összege mindig 0. Mivel a bemenő változónak (középső) kezdéskor 0 értéket adunk, így az értéke kimenőnek is megfelel. Viszont ha egy nem üres listán futtatjuk le a függvényt, akkor ez a sor felel azért, hogy a belső változó értéke kiürített lista esetén a kimenő változóba kerüljön.
  - **sum(L,S) :- sum(L, 0, S).** → A 0 értéket lefixálja, így a felhasználónak nincs lehetősége átírni azt.
  - Meghívás példa: **sum([1,5,100,3], S)** → Az S változó fogja kihozni az értéket.
- **Lista elemeinek megszámolása** (A működési elve ugyanaz, mint a sum függvényé!)
  - **count([], C, C).**
  - **count([_|T], Cin, Cout) :- CIn2 is CIn+1, count(T, CIn2, Cout).**
- **Átlag kiszámítása** (A sum és a count függvény felhasználásával)
  - **average(L, A)** :- → Paraméterként vár egy listát és egy változót, amibe az eredményt kiadhatja.
  - **sum(L, S),** → A sum függvény végrehajtódik az L listán, az eredmény az S változóban tárolódik.
  - **count(L, C),** → A Count függvény végrehajtódik az L listán, az eredmény a C változóban tárolódik.
  - **A is S / C.** → A szumma elosztása a mennyiséggel, a végeredmény az A változóban tárolódik, ami átadja a megadott kimenő változónak.

A Prologban is léteznek **beépített modulok** többek között a fentebb vázolt problémák megoldására.

Tartalomjegyzék

¤