

Reprezentarea cunoștințelor cu reguli if-then

Regulile de tip if-then, numite și reguli de producție, constituie o formă naturală de exprimare a cunoștințelor și au următoarele caracteristici suplimentare:

- Modularitate: fiecare regulă definește o cantitate de cunoștințe relativ mică și independentă de celelalte.
- Incrementabilitate: noi reguli pot fi adăugate bazei de cunoștințe în mod relativ independent de celelalte reguli.
- Modificabilitate (ca o consecință a modularității): regulile vechi pot fi modificate relativ independent de celelalte reguli.
- Susțin transparența sistemului.

Această ultimă proprietate este o caracteristică importantă a sistemelor expert. Prin transparența sistemului se înțelege abilitatea sistemului de a explica deciziile și soluțiile sale. Regulile de producție facilitează generarea răspunsului pentru următoarele două tipuri de întrebări ale utilizatorului:

- întrebare de tipul “cum”: Cum ai ajuns la această concluzie?
- întrebare de tipul “de ce”: De ce te interesează această informație?

Regulile de tip if-then adesea definesc relații logice între conceptele aparținând domeniului problemei. Relațiile pur logice pot fi caracterizate ca aparținând așa-numitelor cunoștințe categorice, adică acelor cunoștințe care vor fi întotdeauna adevărate. În unele domenii însă, cum ar fi diagnosticarea în medicină, predomină cunoștințele “moi” sau probabiliste. În cazul acestui tip de cunoștințe, regularitățile empirice sunt valide numai până la un anumit punct (adesea, dar nu întotdeauna). În astfel de cazuri, regulile de producție pot fi modificate prin adăugarea la interpretarea lor logică a unei calificări de verosimilitate, obținându-se reguli de forma următoare:

if condiție A then concluzie B cu certitudinea F

Pentru a exemplifica folosirea regulilor de producție, vom lua în considerație baza de cunoștințe din Fig. 4.4, care își propune să trateze problema scurgerii de apă în apartamentul din aceeași figură. O scurgere de apă poate interveni fie în baie, fie în bucătărie. În ambele situații, scurgerea provoacă o problemă (inundație) și în hol. Această bază de cunoștințe simplistă nu presupune decât existența defectelor unice: problema poate fi la baie sau la bucătărie, dar nu în ambele locuri în același timp. Ea este reprezentată în Fig. 4.4 sub forma unei rețele de inferență:

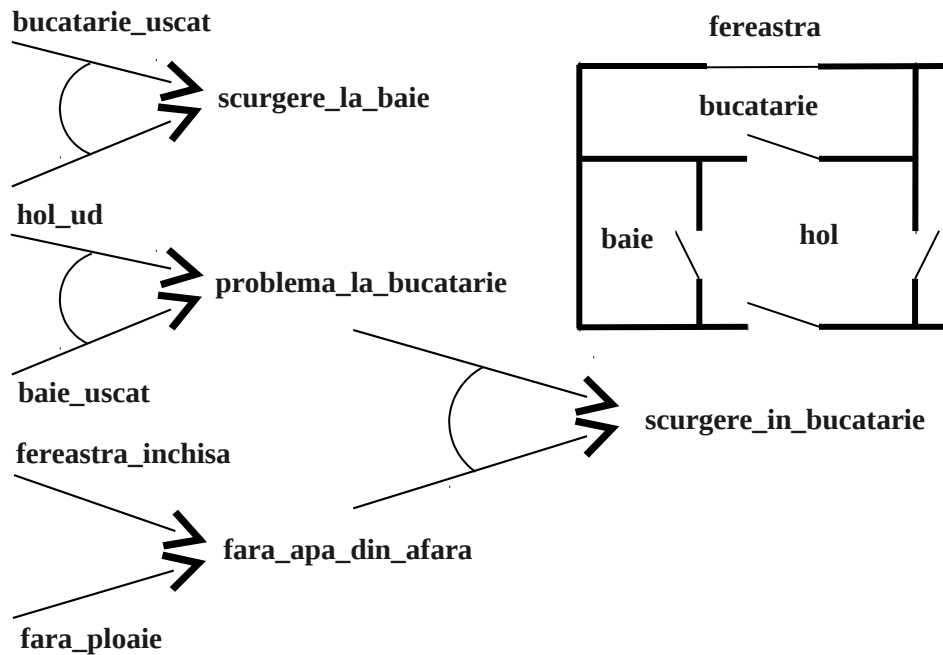


Fig. 4.4

Nodurile rețelei corespund propozițiilor, iar legăturile corespund regulilor din baza de cunoștințe. Arcele care conectează unele dintre legături indică conexiunea conjunctivă existentă între propozițiile corespunzătoare. În consecință, regula referitoare la existența unei probleme în bucătărie, în cadrul acestei rețele, este:

if hol_ud si baie_uscat then problema_la_bucatarie.

Înlănțuire înainte și înapoi în sistemele bazate pe reguli

Atunci când cunoștințele sunt reprezentate într-o anumită formă, este nevoie de o procedură de raționament care să tragă concluzii derivate din baza de cunoștințe. În cazul regulilor de tip if-then, există două modalități de a raționa, ambele extrem de ușor de implementat în Prolog, și anume:

- înlănțuire înapoi (“backward chaining”);
- înlănțuire înainte (“forward chaining”).

Înlănțuirea înapoi

Raționamentul de tip “înlănțuire înapoi” pleacă de la o ipoteză și apoi parcurge în sensul

“înapoi” rețeaua de inferență. În cazul bazei de cunoștințe din Fig. 4.4, spre exemplu, una dintre ipotezele de la care se poate pleca este `scurgere_in_bucatarie`. Pentru ca această ipoteză să fie confirmată, este nevoie ca `problema_la_bucatarie` și `fara_apa_din_afara` să fie adevărate. Prima dintre acestea este confirmată dacă se constată că holul este ud și baia este uscată. Cea de-a doua este confirmată dacă se constată, de pildă, că fereastra este închisă.

Acest tip de raționament a fost numit “înlănțuire înapoi” deoarece el urmează un lanț de reguli care pleacă de la ipoteză (`scurgere_in_bucatarie`) și se îndreaptă către faptele evidente (`hol_ud`).

Acest mod de a raționa este extrem de simplu de implementat în Prolog, întrucât reprezintă însuși mecanismul de raționament încorporat în acest limbaj. Cea mai simplă și mai directă modalitate de implementare este cea care enunță regulile din baza de cunoștințe sub forma unor reguli Prolog, ca în exemplele următoare:

```
problema_la_bucatarie:-  
    hol_ud,  
    baie_uscat.
```

```
fara_apa_din_afara:-  
    fereastra_inchisa  
    ;  
    fara_ploaie.
```

Faptele observate ca fiind evidente pot fi reprezentate sub forma unor fapte Prolog de tipul:

```
hol_ud.  
    baie_uscat.  
fereastra_inchisa.
```

Ipoteza de la care s-a plecat poate fi acum verificată printr-o interogare a Prologului de tipul următor:

```
?- scurgere_in_bucatarie.  
yes
```

În această implementare, în cazul regulilor, a fost folosită însăși sintaxa limbajului Prolog. Această abordare prezintă anumite dezavantaje, printre care faptul că expertul în domeniu care utilizează baza de cunoștințe trebuie să fie familiarizat cu limbajul Prolog, întrucât el trebuie să citească regulile, să le poată modifica și să poată specifica reguli noi. Un al doilea dezavantaj al acestei implementări este faptul că baza de cunoștințe nu se poate distinge, din punct de vedere sintactic, de restul programului.

Pentru a face distincția dintre baza de cunoștințe și restul programului mai clară, sintaxa regulilor expert poate fi modificată prin folosirea unor operatori definiți de utilizator. Spre exemplu, pot fi folosiți ca operatori if, then, and și or, declarați în felul următor:

```
:- op(800, fx, if).
:- op(700, xfx, then).
:- op(300, xfy, or).
:- op(200, xfy, and).
```

Regulile pot fi scrise atunci sub forma:

```
if
    hol_ud and bucatarie_uscat
then
    scurgere_la_baie.
if
    fereastra_inchisa or fara_ploaie
then
    fara_apa_din_afara.
```

Faptele observate pot fi enunțate sub forma unei proceduri pe care o vom numi `fapta`:

```
fapta (hol_ud).
fapta (baie_uscat).
fapta (fereastra_inchisa).
```

În această nouă sintaxă este nevoie de un nou interpretor pentru reguli. Un astfel de interpretor poate fi definit sub forma procedurii

`este_adevarat(P)`

unde propoziția `P` este fie dată prin intermediul procedurii `fapta`, fie poate fi derivată prin utilizarea regulilor. Noul interpretor este următorul:

```
:- op(800, fx, if).
:- op(700, xfx, then).
:- op(300, xfy, or).
:- op(200, xfy, and).
```

```
este_adevarat(P):-  
    fapta(P).
```

```
este_adevarat(P):-  
    if Conditie then P,  
    este_adevarat(Conditie).
```

```
este_adevarat(P1 and P2):-  
    este_adevarat(P1),  
    este_adevarat(P2).
```

```
este_adevarat(P1 or P2):-  
    este_adevarat(P1)  
    ;  
    este_adevarat(P2).
```

Se observă că acest interpretor pentru reguli if-then de tip “înlănțuire înapoi” continuă să lucreze înapoi în maniera depth-first.

Interogarea Prologului se face acum în felul următor:

```
?- este_adevarat(scurgere_in_bucatarie).  
yes
```

Înlănțuirea înainte

Înlănțuirea înainte nu începe cu o ipoteză, ci face un raționament în direcție opusă, de la partea cu if la partea cu then. În cazul exemplului studiat, de pildă, după ce se observă că holul este ud iar baia este uscată, se trage concluzia că există o problemă la bucătărie.

Interpretorul pentru înlănțuirea înainte pe care îl vom prezenta aici presupune că regulile sunt, ca și înainte, de forma

if Conditie then Concluzie

unde Conditie poate fi o expresie de tipul AND/OR. Pentru simplitate vom continua să presupunem că regulile nu conțin variabile. Interpretorul începe cu ceea ce este deja cunoscut (și enunțat prin intermediul relației *fapta*), trage toate concluziile posibile și adaugă aceste concluzii (folosind *assert*) relației *fapta*:

inainte:-

```
fapta_noua_dedusa(P),  
!,  
write('Dedus: '),write(P),nl,  
assert(fapta (P)),  
inainte  
;
```

```
write('Nu mai exista fapte').
```

fapta_noua_dedusa(Concl):-

```
if Cond then Concl,  
not fapta(Concl),  
fapta_compusa(Cond).
```

fapta_compusa(Cond):-

```
fapta(Cond).
```

fapta_compusa(Cond1 and Cond2):-

```
fapta_compusa(Cond1),  
fapta_compusa(Cond2).
```

fapta_compusa(Cond1 or Cond2):-

```
fapta_compusa(Cond1)  
;  
fapta_compusa(Cond2).
```

Interogarea Prologului se face în felul următor:

?-inainte.

Dedus: problema_la_bucatarie

Dedus: fara_apa_din_afara

Dedus: scurgere_in_bucatarie

Nu mai exista fapte

Concluzii

Regulile if-then formează lanțuri de forma
informație input → ... → informație dedusă

Aceste două tipuri de informație sunt cunoscute sub diverse denumiri în literatura de specialitate, denumiri care depind, în mare măsură, de contextul în care ele sunt folosite. Informația de tip input mai poartă denumirea de date sau manifestări. Informația dedusă constituie ipotezele care trebuie demonstrate sau cauzele manifestărilor sau diagnostice sau explicații.

Atât înlănțuirea înainte, cât și cea înapoi presupun căutare, dar direcția de căutare este diferită pentru fiecare în parte. Înlănțuirea înapoi execută o căutare de la scopuri înspre date, din care cauză se spune despre ea că este orientată către scop. Prin contrast, înlănțuirea înainte caută pornind de la date înspre scopuri, fiind orientată către date.

- care tip de raționament este preferabil:

Răspunsul depinde în mare măsură de problema dată. În general, dacă se dorește verificarea unei anumite ipoteze, atunci înlănțuirea înapoi, pornindu-se de la respectiva ipoteză, pare mai naturală. Dacă însă există o multitudine de ipoteze și nu există o anumită motivație pentru testarea cu prioritate a uneia dintre ele, atunci înlănțuirea înainte va fi preferabilă. Această metodă se recomandă și în cazul sistemelor în care datele se achiziționează în mod continuu, iar sistemul trebuie să detecteze apariția oricărei situații reprezentând o anomalie. În acest caz, fiecare schimbare în datele de intrare poate fi propagată înainte, pentru a se constata dacă ea indică vreo eroare a procesului monitorizat sau o schimbare a nivelului de performanță.

În alegerea metodei de raționament poate fi utilă însăși forma rețelei în cauză. Astfel, un număr mic de noduri de date și unul ridicat al nodurilor scop pot sugera ca fiind mai adecvată înlănțuirea înainte. Un număr redus al nodurilor scop și multe noduri corespunzătoare datelor indică înlănțuirea înapoi ca fiind preferabilă.

- majoritatea sistemelor expert sunt infinit mai complexe și necesită o combinație a celor două tipuri de raționament, adică o combinație a înlănțuirii în ambele direcții.

Generarea explicațiilor

Una dintre caracteristicile regulilor de producție care fac din acestea o modalitate naturală de exprimare a cunoștințelor în cadrul sistemelor expert este faptul că ele susțin transparența sistemului. Prin transparența sistemului se înțelege abilitatea acestuia de a explica deciziile și soluțiile sale. Regulile de producție facilitează generarea răspunsului pentru următoarele două tipuri de întrebări ale

utilizatorului:

- Întrebare de tipul “cum”: Cum ai ajuns la această concluzie?
- Întrebare de tipul “de ce”: De ce te interesează această informație?

În cele ce urmează, ne vom ocupa de primul tip de întrebare. O tratare a întrebărilor de tipul “de ce”, care necesită interacțiunea utilizatorului cu procesul de raționament, poate fi consultată în: I. Bratko, „Prolog Programming for Artificial Intelligence”.

În cazul întrebărilor de tipul “cum”, explicația pe care sistemul o furnizează cu privire la modul în care a fost dedus răspunsul său constituie un arbore de demonstrație a modului în care concluzia finală decurge din regulile și faptele aflate în baza de cunoștințe.

Fie “ \leq ” un operator infixat. Atunci arborele de demonstrație al unei propoziții poate fi reprezentat în una dintre următoarele forme, în funcție de necesități:

- Dacă P este o faptă, atunci arborele de demonstrație este P.
- Dacă P a fost dedus folosind o regulă de forma

if Cond then P

atunci arborele de demonstrație este

P \leq DemCond

unde DemCond este un arbore de demonstrație a lui Cond.

- Fie P1 și P2 propoziții ale căror arbori de demonstrație sunt Dem1 și Dem2. Dacă P este de forma P1 and P2, atunci arborele de demonstrație corespunzător este Dem1 and Dem2. Dacă P este de forma P1 or P2, atunci arborele de demonstrație este fie Dem1, fie Dem2.

Construcția arborilor de demonstrație în Prolog este directă și se poate realiza prin modificarea predicatului este_adevarat, introdus anterior, în conformitate cu cele trei cazuri enunțate mai sus. Un astfel de predicat este_adevarat modificat poate fi următorul:

```
%este_adevarat(P, Dem) daca Dem %constituie o demonstratie a faptului  
%ca P este adevarat
```

```
:-op(800,xfx,<=).
```

```
este_adevarat(P,P):-  
    fapta(P).
```



```

este_adevarat(P, P <= DemCond) :-
    if Cond then P,
    este_adevarat(Cond, DemCond) .
este_adevarat(P1 and P2, Dem1 and Dem2) :-
    este_adevarat(P1, Dem1),
    este_adevarat(P2, Dem2) .
este_adevarat(P1 or P2, Dem) :-
    este_adevarat(P1, Dem)
    ;
    este_adevarat(P2, Dem) .

```

Introducerea incertitudinii

Reprezentarea cunoștințelor luată în discuție până acum pleacă de la presupunerea că domeniile problemelor sunt categorice. Aceasta înseamnă că răspunsul la orice întrebare este fie adevărat, fie fals. Regulile care interveneau erau de aceeași natură, reprezentând așa-numite implicații categorice. Totuși, majoritatea domeniilor expert nu sunt categorice. Atât datele referitoare la o anumită problemă, cât și regulile generale pot să nu fie certe. Incertitudinea poate fi modelată prin atribuirea unei calificări, alta decât adevărat sau fals, majorității aserțiunilor. Gradul de adevăr poate fi exprimat prin intermediul unui număr real aflat într-un anumit interval - spre exemplu, un număr între 0 și 1 sau între -5 și +5. Astfel de numere cunosc, în literatura de specialitate, o întreagă varietate de denumiri, cum ar fi factor de certitudine, măsură a încrederii sau certitudine subiectivă.

În cele ce urmează, vom exemplifica prin extinderea reprezentării bazate pe reguli de până acum cu o schemă simplă de incertitudine. Fiecărei propoziții i se va adăuga un număr între 0 și 1 ca factor de certitudine. Reprezentarea folosită va consta dintr-o pereche de forma:

Propoziție: FactorCertitudine

Această notație va fi aplicată și regulilor. Astfel, următoarea formă va defini o regulă și gradul de certitudine până la care acea regulă este validă:

If Condiție then Concluzie: Certitudine.

În cazul oricărei reprezentări cu incertitudine este necesară specificarea modului în care se combină certitudinile propozițiilor și ale regulilor. Spre exemplu, să presupunem că sunt date două propoziții $P1$ și $P2$ având certitudinile $c(P1)$ și respectiv $c(P2)$. Atunci putem defini

$$c(P1 \text{ and } P2) = \min(c(P1), c(P2))$$

$$c(P1 \text{ or } P2) = \max(c(P1), c(P2))$$

Dacă există regula

if P1 then P2: C

cu C reprezentând factorul de certitudine, atunci

$$c(P2) = c(P1) * C$$

Pentru simplitate, vom presupune, în cele ce urmează, că nu există mai mult de o regulă susținând o aceeași afirmație. Dacă ar exista două astfel de reguli în baza de cunoștințe, ele ar putea fi transformate, cu ajutorul operatorului OR, în reguli echivalente care satisfac această presupunere.

Implementarea în Prolog a unui interpretor de reguli corespunzător schemei de incertitudine descrise aici va presupune specificarea de către utilizator a estimațiilor de certitudine corespunzătoare datelor observate (nodurile cel mai din stânga ale rețelei) prin relația

dat(Propozitie, Certitudine).

Iată un asemenea interpretor pentru reguli cu factor de certitudine:

```
% certitudine (Propozitie, Certitudine)
```

```
certitudine(P,Cert):-
```

```
    dat(P,Cert).
```

```
certitudine(Cond1 and Cond2, Cert):-
```

```
    certitudine(Cond1,Cert1),
```

```
    certitudine(Cond2,Cert2),
```

```
    minimum(Cert1,Cert2,Cert).
```

```
certitudine(Cond1 or Cond2, Cert):-
```

```
    certitudine(Cond1,Cert1),
```

```
    certitudine(Cond2,Cert2),
```

```
    maximum(Cert1,Cert2,Cert).
```

```
certitudine(P,Cert):-
```

```
    if Cond then P:C1,
```

```
    certitudine(Cond,C2),
```

```
    Cert is C1*C2.
```

Regulile bazei de cunoștințe studiate anterior pot fi acum rafinate ca în următorul exemplu :

```
if hol_ud and baie_uscat
```

```
then
```

problema_la_bucatarie: 0.9.

O situație în care holul este ud, baia este uscată, bucătăria nu este uscată, fereastra nu este închisă și credem - dar nu suntem siguri - că afară nu plouă, poate fi specificată după cum urmează:

```
dat(hol_ud,1).  
dat(baie_uscat,1).  
dat(bucatarie_uscat,0).  
dat(fara_ploaie,0.8).  
dat(fereastră_inchisa,0).
```

Interogarea Prologului referitoare la o scurgere în bucătărie se face astfel:

```
?-certitudine(scurgere_in_bucatarie,C).
```

```
C = 0.8
```

Factorul de certitudine C este obținut după cum urmează: faptul ca holul este ud iar baia este uscată indică o problemă în bucătărie cu certitudine 0.9. Întrucât a existat o oarecare posibilitate de ploaie, factorul de certitudine corespunzător lui fara_apa_din_afara este 0.8. În final, factorul de certitudine al lui scurgere_in_bucatarie este calculat ca fiind $\min(0.8, 0.9) = 0.8$.

Chestiunea manevrării incertitudinii în sistemele expert a fost îndelung dezbătută în literatura de specialitate. Abordări matematice bazate pe teoria probabilităților există, în egală măsură. Ceea ce li se reproșează, cel mai adesea, este faptul că abordări corecte din punct de vedere probabilistic necesită fie informație care nu este întotdeauna disponibilă, fie anumite presupuneri simplificatoare care, de regulă, nu sunt justificate în aplicațiile practice și care fac, din nou, ca abordarea să nu fie suficient de riguroasă din punct de vedere matematic.

Una dintre cele mai cunoscute și mai utilizate scheme cu factori de certitudine este cea dezvoltată pentru sistemul MYCIN, un sistem expert folosit în diagnosticarea infecțiilor bacteriene. Factorii de certitudine MYCIN au fost concepuți pentru a produce rezultate care păreau corecte experților, din punct de vedere intuitiv. Alți cercetători au argumentat conceperea unor factori de certitudine bazați într-o mai mare măsură pe teoria probabilităților, iar alții au experimentat scheme mult mai complexe, proiectate pentru a modela mai bine lumea reală. Factorii MYCIN continuă însă să fie utilizați cu succes în multe aplicații cu informație incertă.