1

Entitățile limbajului Prolog

1.1. Fapte, întrebări, termeni

Sunt cunoscute numai trei componente ale unui program Prolog: faptele, regulile şi întrebările. Aici vom introduce noțiunile de fapte şi întrebări, iar regulile vor fi examinate mai târziu.

1.1.1. Fapte

Un fapt exprimă a relație între obiecte. Considerăm câteva exemple de fapte, în care se va utiliza următoarele notații Prolog pentru comentarii: tot ce se conține între *"/*"* şi *"*/"* este un comentariu; dacă comentariul începe într-un rând, atunci se poate scrie în fața comentariului caracterul *"%"*.

```
zboara(corb). %corbul zboară
este_autor(shakespeare,hamlet).     /*Shakespeare
       este autorul operei "Hamlet" */
traducere(bread,pain).   /*pain   este   varianta
       franceză a lui bread */
```

Astfel faptul

```
ruta(paris, roma).
```

poate semnifica că Parisul e legat cu Roma de o rută, însă noi nu cunoaştem nici numărul rutei, nici ora de plecare.

Să considerăm sintaxa ultimului fapt. El este compus dintr-un *scop* "*ruta(paris,roma)*" şi un punct, ".", la sfârşit. Şirul de simboluri *ruta* este numele *relației* care se mai numeşte *nume de predicat* sau *functor*, iar *paris* şi *roma* sunt obiectele antrenate în această relație. Întrucât predicatul *ruta* are două argumente *paris* şi *roma,* care sunt puse între paranteze şi despărțite prin virgulă, el se numeşte *predicat binar*. În caz general, un predicat poate avea orice număr de argumente (şi, bineînțeles, zero). Acest număr se numeşte aritatea predicatului, care în cazul predicatului *ruta* este 2. Deci, faptul este cea mai simplă formă de predicat ce semnifică existența unei relații între o mulțime de obiecte.

Fie că predicatele

```
ruta(oras_plecare, oras_sosire).
```

şi

```
ruta(oras_plecare, oras_sosire, ruta).
```

sunt adevărate dacă *oras_plecare* e legat cu *oras_sosire* şi dacă *oras_plecare* e legat cu *oras_sosire* prin ruta *ruta*, corespunzător. Aceste două predicate sunt diferite. Pentru a ne referi la predicatul de aritatea 3 vom utiliza notația *ruta/3*, iar pentru cel de aritatea doi - *ruta/2.*

Atunci când se declară o relație între obiecte trebuie să se atragă atenția la numărul de obiecte puse în relație de predicat şi la ordinea acestor obiecte. Pentru Prolog faptul *ruta/2* nu este decât o simplă consecutivitate de simboluri şi nu-i asociază nici un sens, nici o semnificație. Se poate însă perfect de considerat că primul obiect din predicatul *ruta/2* este oraşul plecării, iar al doilea este oraşul sosirii. Ordinea este, deci, arbitrară, dar fiind făcută o alegere, trebuie să se țină cont de ea dacă dorim ca relația *ruta/2* să reprezinte o semnificație validă (şi ca un program ce o utilizează să producă răspunsuri coerente). Aşa dar, predicatul poate fi conceput ca numele relației (proprietății) între obiecte înzestrat cu aritate în care ordinea obiectelor e importantă.

Deci interpretarea particulară a predicatelor şi argumentelor depinde de programator. Ordinea argumentelor odată fixată, este importantă şi trebuie păstrată la orice utilizare a faptului.

Să considerăm, acum, faptul "Ion este tatăl lui Vasile". Acest fapt se referă la "Ion" şi "Vasile" şi o relație "tatăl lui" (relația de paternitate). În Prolog acest fapt se poate declara prin

```
tatal_lui(ion, vasile).
```

Adică se presupune că primul argument al relației *tatal_lui/2* reprezintă tatăl, iar al doilea - feciorul.

Mulțimea tuturor faptelor formează baza de fapte. Ea constituie o parte a unui program Prolog. Fie că avem următoarea bază de fapte:

```
ruta(paris, roma).
ruta(paris, madrid).
ruta(madrid, porto).
ruta(londra, paris).
ruta(londra, madrid).
ruta(amsterdam, paris).
```

Să remarcăm, acum, următoarele:

- numele de obiecte (constantele) şi relațiile (predicatele) în limbajul Prolog încep cu minusculă (şi pot continua cu caracterul sublinierea, "_")
- predicatul (numele relației) se scrie primul, iar obiectele în relație sunt puse între paranteze şi separate prin virgule (atenție la operatori pentru care Prologul utilizează şi alte notații)
- punctul, ".", indică că declarația faptului s-a terminat.

1.1.2. Întrebări asupra unei baze de fapte

Îndată ce sunt declarate faptele e posibilă interogarea Prologului despre subiectele lor. Întrebarea se mai numeşte scop

şi se deosebeşte la unele sisteme prin prezenţa simbolului *":-"* sau *"?"*, ce de fapt este invitaţia Prologului de a o insera. Considerăm, de exemplu, întrebarea

```
?-ruta(paris, roma)
```

Precum se va ştie din ultimele capitole ale lucrării de faţă, a răspunde la o întrebare pusă unui program constă în determinarea dacă ea poate fi dedusă din program, aplicând regulile de deducere (fie regula rezoluţiei). Dacă programul conţine un fapt ce coincide cu întrebarea, atunci răspunsul sistemului este afirmativ, *yes*, în caz contrar negativ, *no*. Răspunsul *no* semnifică numai că faptul nu este în baza de fapte. Mai târziu se va vedea că sistemul, în cazul unui răspuns afirmativ la o întrebare, poate furniza şi alte informaţii din baza de fapte.

La baza de fapte de mai sus se poate, în particular, de pus următoarele întrebări la care Prologul va răspunde precum urmează.

```
?-ruta(paris, roma)
yes
?-ruta(roma, paris)
no
```

1.1.3. Termenii în Prolog

Descrierea sintactică a limbajului Prolog are două aspecte: sintaxa termenilor şi sintaxa componentelor unui program. Aici, vom întrerupe pentru un timp descrierea componentelor unui program pentru a defini termenii.

Termenii sunt formaţi dintr-o mulţime de caractere. Caracterele recunoscute în Prolog se divizează în patru categorii:

- mulţimea de litere majuscule: `A B C D … X Y Z`
- mulţimea de litere minuscule: `a b c d … x y z`
- mulţimea de caractere numerice: `0 1 2 3 4 5 6 7 8 9`
- mulţimea de caractere speciale: `! " # $ % ' ( ) = - ~ ^ \ | { } [ ] _ ` @ + : ; * < > , . ? /`

E cunoscut faptul că termenul este singura structură de date utilizată în programarea logică. Mulţimea de termeni ce poate fi creată şi utilizată în Prolog se grupează conform structurii ierarhice reprezentate în Fig. 1.1.

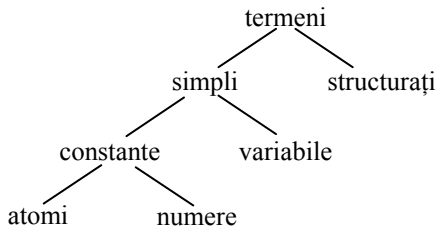

Fig. 1.1. Termenii în Prolog

Diferenţa dintre diverşi termeni se bazează doar pe sintaxa lor. Astfel, numai considerând componentele termenului se poate concluziona ce fel de termen el este (un atom, un număr, o variabilă sau un termen structurat) fără a ne referi la declaraţia lor. De fapt, spre deosebire de limbajele imperative, în Prolog nu se declară datele (cu mici excepţii, de exemplu, Turbo Prolog sau Visual Prolog).

Constante. Orice constantă se referă la un obiect individual. Constantele în Prolog servesc pentru definirea obiectelor şi relaţiilor particulare. Există două categorii de constante: *atomi* şi *numere*. Modul de interpretare a constantei, cum ar fi locul ei într-o relaţie, depinde de programator.

Atomi. Atomul este o varietate de termen ce începe cu o literă minusculă. Celelalte simboluri pot fi litere (inclusiv majuscule), numere şi simbolul de subliniere, "_". O serie de simboluri sistemul le consideră atomi *("+", "-", "=", ":-"* etc.). Perechea de simboluri *[ ]* de asemenea este un atom, însă nici *[*, nici *]* aparte nu sunt atomi. Exemplu de atomi:

atom	*+*
a2000	*:-*
a_Fi_sau_A_nu_fi	*-->*

Orice nume de predicat este un atom. Deci, în faptul

```
ruta(paris, roma).
```

termenii *ruta*, *paris* şi *roma* sunt atomi.

O secvenţă de caractere incluse între ghilimele simple '…' (în Turbo Prolog se folosesc ghilimele duble "…") de asemenea este considerat în calitate de atom. Pentru o mai bună reprezentare a atomilor între ghilimele poate fi inserat şi caracterul blanc, de exemplu:

'32'	*'Trăiască regele!'*
'Rebreanu'	*'+important'*

Deci atomii pot desemna:

- obiecte constante ce sunt argumente ale predicatelor
- nume de predicate, fie definite de utilizator, fie cele predefinite în sistem
- atomi speciali

Numere. Numerele sunt formate din cifre, punct şi din semnele "+" şi "-". Precizia şi tipul numerelor depind de implementarea limbajului Prolog. Astfel

32	*+1.2*
28.9	*-1*

sunt numere. Operaţiile aritmetice pot fi făcute doar asupra numerelor. Asupra atomului '32' nu pot fi aplicate operaţii aritmetice.

Variabile. Variabila este un termen ce începe cu o majusculă sau cu caracterul de subliniere, *"_"*. Celelalte simboluri pot fi numere, litere majuscule şi/sau minuscule. Variabilele se utilizează pentru desemnarea obiectelor necunoscute. O astfel de interpretare diferă de interpretările variabilei în alte limbaje de programare. În limbajele imperative o variabilă este concepută ca un loc de memorare, în care în diferite momente de timp pot fi plasate diferite valori ale variabilei. În programarea funcţională (de exemplu, în limbajul Lisp) variabilele sunt nume cu care se asociază valori odată şi pentru totdeauna. E important de reţinut aceste trei concepţii diferite de variabile.

Termenii ce urmează sunt variabile

Atom	*_aceasta_de_asemenea*

Rebreanu *_2000*
Aceasta_e_1_variabila *_ _*

dar termenii

aCESTA_NU
NICI-ACESTA

nu sunt variabile. Un singur caracter *"_"* este o formă specială de variabilă numită *variabila anonimă*. Ea se utilizează pentru a arăta că pe poziția dată se acceptă orice valoare care nu ne interesează.

Termeni structurați (compuşi sau complecşi). Termenul structurat desemnează un obiect în care sunt regrupate alte obiecte numite componente. Regruparea permite o manipulare mai uşoară a obiectelor complexe.

Un termen structurat se defineşte de functorul său, componentele sale şi aritatea sa (numărul de componente). Considerăm, de exemplu, următorul termen structurat

autor(blaise, pascale)

El poate fi interpretat că există un *autor* al cărui nume este *pascal* şi prenume *blaise*. Acest termen are functorul *autor*, componentele *blaise* şi *pascale* şi aritatea 2. E evident că componentele unui termen structurat pot fi la rândul său termeni structurați. Astfel termenul structurat

cartea(cugetari, autor(blaise, pascale))

semnifică că *cugetări* este o carte a cărei *autor* este *blaise pascale*. Acest termen are două componente *cugetari* şi termenul structurat *autor(blaise, pascale).*

Un termen structurat poate fi reprezentat sub forma unui arbore a cărui rădăcină e simbolul funcțional (adică, functorul), numărul de descendenți ai rădăcinii fiind aritatea, adică numărul de componente ale termenului structurat, ordonarea descendenților corespunde ordonării componentelor. Dacă, la rândul său, o componentă e un termen structurat, ea se descompune în acelaşi mod. De exemplu termenul structurat *cartea(cugetari, autor(blaise, pascale))* e reprezentat de arborele din Fig 2.2.

Sintaxa termenilor structurați e asemănătoare cu cea a faptelor. Deci, un predicat poate fi considerat ca un termen structurat a cărui functor este numele predicatului, iar argumentele reprezintă componentele termenilor structurați. Acest lucru este utilizat în diverse implementări Prolog pentru tratarea uniformă şi sinteza dinamică de programe.

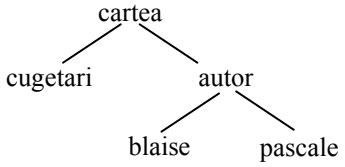

Fig. 1.2. Reprezentarea unui termen structurat

1.1.4. Unificarea termenilor

Unificarea este unul din mecanismele ce situează Prologul completamente pe altă parte decât limbajele tradiționale de programare. Programatorii începători frecvent greşesc, fiindcă unificarea se aseamănă atât cu testul de egalitate ("==" în C), cât şi cu procesul de asignare a unei valori unei variabile (ca ":=" in Pascal).

Unificarea se aseamănă cu un test de egalitate, fiindcă ea permite Prologului de a verifica dacă doi termeni sunt identici. Pentru aceasta se utilizează un predicat predefinit *"="*. Pentru a testa dacă doi termeni se unifică, e suficient să se scrie expresia *termen1=termen2*.

Urmează regulile generale pentru determinarea în ce mod doi termeni *termen1* şi *termen2* pot fi unificați.

- Înainte de toate, menționăm că dacă *termen1* (sau *termen2*) e o variabilă instanțiată, atunci *termen1* (sau *termen2*) se comportă ca termenul cu care a fost instanțiată, adică ca o constantă sau o structură, sau o altă variabilă.
- Dacă *termen1* şi *termen2* ambii sunt atomi sau numere, ei pot să se unifice când fiecare este acelaşi atom sau număr. În particular, un număr real nu poate să se unifice cu un număr întreg, chiar dacă exprimă aceeaşi valoare.
- Dacă *termen1* şi *termen2* ambii sunt termeni structurați, ei pot fi unificați când fiecare are acelaşi functor principal, acelaşi număr de argumente şi argumentele ce corespund aceloraşi poziții la rândul său pot să se unifice.
- În sfârşit, dacă *termen1* şi *termen2* sunt ambii variabile neinstanțiate, ei pot fi unificați. În acest caz variabilele rămân neinstanțiate, dar legate una de alta, până cel puțin una din ele nu va fi instanțiată şi atunci, şi cealaltă devine automat instanțiată cu acelaşi termen.

Să aducem, acum, nişte exemple.

Atomii şi numerele se unifică numai cu ei însăşi. Prologul procedează astfel. El compară termenii de la stânga la dreapta simbol cu simbol. Dacă toate simbolurile acestor doi termeni coincid, atunci ei se unifică, în caz contrar ei nu se unifică (vezi Fig 1.3).

Termenul 1	Termenul 2	Se unifică?
Ion	*ion*	*Da*
Ion	*petru*	*Nu*
12	*12*	*Da*
12	*12.0*	*Nu*
'Liviu Rebreanu'	*'Liviu Rebreanu'*	*Da*
'Liviu Rebreanu'	*scriitor*	*Nu*
+	*plus*	*Nu*
12	*3*4*	*Nu*

Fig. 1.3. Unificarea atomilor şi numerelor

Un termen structurat se unifică cu altul dacă şi numai dacă:

- ei au acelaşi functor
- ei au acelaşi număr de argumente

- orice argument al unui termen se unifică cu argumentul corespunzător (de pe aceeaşi poziţie) al celuilalt termen

Exemple de unificare a termenilor structuraţi sunt aduse în Fig 2.4.

Termenul 1	Termenul 2	Se unifică?
Autor(blaise,pascale)	*autor(blaise,pascale)*	*da*
Autor(blaise,pascale)	*nume(blaise,pascale)*	*nu*
Autor(blaise,pascale)	*autor(blaise,pascale,1623)*	*nu*
*3*4*	**(3,4)*	*da*
3+4	*4+3*	*nu*

Fig. 1.4. Unificarea unor termeni structuraţi

O variabilă se unifică cu orice termen dacă ea nu este instanţiată. În caz contrar, ea se comportă ca valoarea sa. *Variabilele instanţiate* (Instantiated Variables) sunt variabilele care au denotat obiecte necunoscute şi în procesul de instanţiere s-au transformat în variabile ce denotă obiecte cunoscute. Variabila este *neinstanţiată* dacă nu desemnează nici un obiect.

Să observăm aici că mecanismul de unificare a variabilei poate juca două roluri: de testare a egalităţii şi de asociere a unei valori variabilei. Dacă variabila *X* nu este instanţiată, atunci expresia *"X=a"* îi va atribui în calitate de valoare atomul *a*. Când variabila are o valoare asociată, se spune că ea s-a unificat cu această valoare. De exemplu, dacă variabila *X* nu este instanţiată, ea poate să se unifice cu *a(x,y,z)*. Dar, *a(x,y,z)* nu se poate unifica decât cu un termen structurat care are în calitate de functor *"a"* şi are trei argumente, primul argument al cărui se poate unifica cu *x*, al doilea cu *y* şi al treilea cu *z*. Deci *a(G,y,K)* se poate unifica cu variabila instanţiată *X*, adică cu *a(x,y,z)*, dacă *G* şi *K* nu sunt instanţiate (sau dacă *G* are valoarea *x* şi/sau dacă *K* are valoarea *z*).

Să menţionăm că, în limbajul Pascal expresia *"X:=a"* (ce denotă plasarea valorii *a* în memoria notată de *X*) variabila se găseşte întotdeauna în partea stângă a semnului de atribuire, *":="*. În Prolog, însă, expresiile *"X=a"* şi *"a=X"* sunt echivalente.

1.1.5. Întrebări cu variabile

Să revenim la întrebări. Întrebările asupra bazei de fapte de tipul celor considerate mai sus nu prezintă prea mare interes. Singura alegere permisă constă în verificarea dacă o informaţie este sau nu în baza de fapte. Din cele cunoscute de mai sus, pare dificil ca Prologul să poată obţine un răspuns la o întrebare simplă (pentru noi) cum ar fi: "În ce oraşe zboară avioanele din Paris?". Ar trebui de întrebat Prologul dacă orice oraş posibil e legat cu Parisul. Acest lucru e foarte ineficient, deoarece sunt multe oraşe şi, atunci, se presupune că le cunoaştem pe toate.

Evident că, ideal ar fi dacă Prologul singur ar căuta faptele despre oraşele în care sosesc avioane din Paris şi ar afişa pentru orice fapt găsit oraşul de sosire.

Graţie utilizării *variabilelor* este posibilă obţinerea rezultatului dezirabil, rămânând ca Prologul să facă lucrul de căutare. Precum se ştie, o variabilă este un nume în care prima literă este o majusculă sau caracterul de subliniere, *"_"* şi permite desemnarea unui obiect oarecare. Într-o întrebare, de exemplu, ea serveşte pentru desemnarea unui obiect necunoscut (oraşul de sosire, ca în întrebarea de mai jos).

Deci, când se pune o întrebare care conţine o variabilă, Prologul parcurge baza de fapte pentru a căuta un obiect pe care variabila poate să-l desemneze. Aşa dar, pornind de la baza noastră de fapte şi fiind pusă întrebarea

```
?-ruta(paris, X)
```

ce semnifică "În ce oraşe zboară avioanele din Paris?", sistemul va răspunde

```
X=roma
```

Aici trebuie să remarcăm, că unele sisteme Prolog enumără toate oraşele ce sunt legate cu Parisul, iar altele aşteaptă o instrucţiune specială, dacă dorim ca după fiecare răspuns Prologul să mai continue căutarea. În sistemul Turbo Prolog, de exemplu, dacă întrebarea este externă în sensul că este pusă în fereastra de dialog, atunci sistemul enumără toate oraşele în care sosesc avioane din Paris. Însă, dacă întrebarea este pusă în program, atunci se afişează primul răspuns, iar de obţinerea şi celorlalte răspunsuri posibile trebuie să aibă grijă programatorul.

Dar acum presupunem, că se doreşte de a afla dacă zboară undeva avioane din Paris, fără a fi preocupaţi care sunt aceste oraşe. Pentru aceasta se utilizează *variabila anonimă* "_", iar scopul se formulează astfel

```
?-ruta(paris, _)
```

Prologul răspunde simplu *yes* sau *no* în dependenţă dacă există un astfel de oraş sau nu.

Se poate, de asemenea, de pus variabile şi în fapte. Astfel faptul

```
ruta(paris, _).
```

presupune că Parisul e legat cu toate celelalte oraşe. Mai departe, în alte secţiuni, se va vedea cum pot fi utilizate variabilele pentru a formula proprietăţi mai complexe referitor la obiecte.

Spre deosebire de variabila obişnuită, variabila anonimă are următoarele particularităţi:

- Prologul nu întoarce valoarea cu care variabila este instanţiată
- Dacă variabila apare de mai multe ori în aceeaşi clauză (a se vedea noţiunea mai departe) sau întrebare, fiecare apariţie a ei nu desemnează neapărat acelaşi obiect. Cu alte cuvinte, orice variabilă anonimă este o variabilă independentă de celelalte.

1.1.6. Întrebări cu termeni structuraţi

E posibilă formularea întrebărilor asupra faptelor ce conţin termeni structuraţi. De exemplu, cu baza de fapte

```
poseda(marcel, cartea(cugetari, autor(blaise, pascale))).
poseda(marcel, cartea(moby_dick, autor(herman, melville))).
```

la întrebarea

```
?-poseda(marcel, Obiect)
```

care semnifică "Ce obiecte posedă Marcel?", Prologul răspunde

```
Obiect=cartea(cugetari, autor(blaise, pascale));
Obiect=cartea(moby_dick, autor(herman, melville))
```

La întrebarea

```
?-poseda(marcel, cartea(_, autor(Prenume, _)))
```

ce semnifică “Care sunt prenumele autorilor cărților ce le posedă Marcel?”, Prologul răspunde

```
Prenume=blaise;
Prenume=herman
```

Să se observe aici utilizarea variabilelor anonime pentru obținerea doar a prenumelor.

Dacă la baza de fapte de mai sus se adaugă faptul

```
poseda(marcel, bicicleta).
```

atunci la întrebarea

```
?-poseda(marcel, Obiect)
```

care semnifică “Ce obiecte posedă Marcel?”, Prologul va răspunde

```
Obiect=cartea(cugetari, autor(blaise, pascale))
Obiect=cartea(moby_dick, autor(herman, melville))
Obiect=bicicleta
```

Deci, nimic nu ne impune ca obiectele despre care este vorba să fie neapărat cărți.

1.1.7. Mecanismul de unificare şi substituția

Să examinăm neformal mecanismul utilizat de Prolog în căutarea soluțiilor la întrebări. Prologul caută în baza de fapte un fapt ce se unifică cu scopul formulat, adică încearcă să facă ca variabila, care este liberă în întrebarea pusă, să coincidă cu orice argument ce se găseşte pe acelaşi loc într-un fapt cu acelaşi nume de predicat şi acelaşi număr de argumente ca şi întrebarea. În cazul întrebării

```
?-ruta(paris, X)
```

Prologul parcurge baza de fapte şi căută un fapt cu numele de predicat *ruta* şi două argumente, primul dintre care este *paris*. Când Prologul găseşte un astfel de fapt, el unifică *X* cu argumentul doi al faptului. În acest caz se mai spune că variabila *X* este *instanțiată* de acest argument.

Precizăm în ce mod Prologul parcurge baza de fapte. El “examinează” faptele în ordinea în care sunt stocate în baza de fapte, începând cu primul. De aceea, în exemplul nostru Prologul afişează mai multe răspunsuri în următoarea ordine:

```
X=roma
X=madrid
```

După unificare Prologul efectuează o operație invizibilă pentru utilizator: el marchează faptul din baza de fapte cu care s-a produs unificarea.

Utilitatea acestui marcaj se explică prin următoarele. Îndată ce este găsit un fapt cu care se unifică, faptul este marcat. Dacă e necesară parcurgerea mai departe a bazei de fapte pentru a căuta şi alte posibile unificări, sistemul nu cercetează decât partea baza de fapte ce urmează după faptul marcat. Dacă nu ar fi fost marcate faptele unificate, atunci Prologul ar relua parcurgerea bazei de fapte de la început şi ar fi afişat acelaşi răspuns - primul găsit. Adică programul ar fi intrat într-o buclă. Deci, marcajul permite Prologului să recunoască partea din baza de fapte ce nu a fost încă examinată într-un anumit context. Reluând parcurgerea mai jos de faptul marcat, marcajul se suprimă şi Prologul caută un nou fapt cu care scopul ar putea fi unificat etc.

În exemplul nostru, dacă se foloseşte sistemul Turbo Prolog şi scopul este inserat în program, sistemul nu consideră faptele ce urmează după faptul

```
ruta(paris, roma).
```

dacă, bineînțeles, scopul nu presupune căutarea tuturor soluțiilor. Dacă, însă, întrebarea este inserată în fereastra de dialog, atunci sistemul marchează faptul de mai sus, afişează răspunsul

```
X=roma
```

Apoi se reîntoarce la faptul marcat, suprimă marcajul şi examinează faptele ce urmează după el. Deci, sistemul încearcă să unifice scopul cu faptul

```
ruta(paris, madrid).
```

Unificarea, evident, are loc. Acest fapt este marcat şi se afişează soluția

```
X=madrid
```

1.1.8. Conjuncții de scopuri şi variabile partajate

Fie că dorim să punem o întrebare de felul “Care sunt oraşele în care aterizează avioane ce sosesc din Paris şi din Londra?”. E evident, că mai întâi se poate de întrebat sistemul ce oraşe sunt legate cu Parisul, apoi, dacă răspunsul conține careva oraşe, de întrebat ce oraşe din cele legate cu Parisul sunt legate cu Londra. Deci, întrebarea de fapt e descompusă în două. Însă Prologul oferă posibilitatea de a pune direct întrebarea globală, separând subscopurile prin virgulă:

```
?-ruta(paris, X), ruta(londra, X)
```

Virgula se citeşte “şi” şi permite separarea a orice număr de subscopuri ce trebuie satisfăcute pentru a răspunde la scopul întreg. Deci, când o consecutivitate de subscopuri sunt puse Prologului, acesta încearcă să treacă (pentru a satisface) succesiv orice subscop, căutând pentru el un fapt corespunzător din baza de fapte. Consecutivitatea de subscopuri, adică întreg scopul, va putea fi parcurs (şi atunci Prologul va putea răspunde la întrebare) numai dacă vor fi satisfăcute succesiv toate subscopurile. Deci, pentru a răspunde la întrebarea de mai sus Prologul procedează astfel. El încearcă să treacă primul subscop; dacă subscopul este unificat, Prologul marchează faptul cu care se unifică şi încearcă să treacă la următorul subscop; dacă reuşeşte să unifice şi al doilea subscop, Prologul îl marchează şi afişează răspunsul. Dacă, din potrivă, al doilea subscop nu poate fi trecut, atunci Prologul se întoarce la primul subscop, suprimă marcarea faptului ce s-a unificat cu primul subscop şi încearcă să treacă primul subscop, începând cu faptele din bază ce urmează după faptul marcarea căruia a fost suprimată. Acest comportament se numeşte backtracking.

Să considerăm, acum, întrebarea

```
?-ruta(paris, X), ruta(amsterdam, Y)
```

care semnifică “Unde pleacă avioanele din Paris şi din Amsterdam?”. Răspunsul bineînțeles va fi

```
X=roma, Y=paris;
X=madrid, Y=paris
```

Să aducem o scurtă explicație. După primul răspuns (*X=roma, Y=paris),* Prologul încearcă să parcurgă în alt mod subscopul *ruta(amsterdam, Y),* suprimând marcajul de la faptul

```
ruta(amsterdam, paris).
```

Întrucât acest lucru nu e posibil, sistemul revine la faptul ce a fost unificat cu primul subscop, îi suprimă marcarea şi încercă să unifice primul subscop cu faptele ce urmează mai jos de faptul marcat. Primul subscop se unifică cu faptul

```
ruta(paris, madrid).
```

îl marchează şi purcede la unificarea subscopului doi. El poate fi unificat doar cu acelaşi fapt

```
ruta(amsterdam, paris).
```

în urma căruia s-a obținut cea dea doua soluție. E evident, că mai multe soluții nu pot fi.

Acum, să considerăm puțin mecanismul backtracking. El constă în satisfacerea şi resatisfacerea repetată a subscopurilor unui scop. La satisfacerea unei conjuncții de subscopuri, se încearcă satisfacerea fiecărui subscop pe rând, de la stânga la dreapta. Prima satisfacere a unui subscop determină plasarea unui marcaj în dreptul faptului care a determinat satisfacerea subscopului. Dacă un subscop nu poate fi satisfăcut, adică eşuează, sistemul se întoarce şi încearcă resatisfacerea subscopului cel mai aproape din stânga, pornind căutarea în bază de la marcaj în jos. Înainte de resatisfacerea unui subscop, se elimină toate instanțierile de variabile determinate de ultima satisfacere a acestuia.

1.2. Reguli

Presupunem că se afirmă: "Toți atenienii sunt greci". Această afirmație poate fi descrisă dacă pentru orice nume de persoană se construieşte un fapt ce indică că persoana este atenian şi un fapt că persoana este grec. Dar acest lucru este foarte ineficient. Din fericire, Prologul permite de a descrie propoziții de acest tip prin intermediul regulilor.

Înainte de a arăta cum se descriu regulile în Prolog, menționăm că afirmația de mai sus poate de asemenea să se formuleze: "Dacă cineva e atenian, atunci el este grec" sau "Orice persoană e grec, dacă e atenian". În Prolog se scrie

```
grec(X):-atenian(X).
```

şi se citeşte "*X* este grec, dacă *X* este atenian" sau "*X* este atenian implică *X* este grec".

Aşadar, Prologul permite de a formula reguli de tipul: "Un fapt *a* este adevărat, dacă un fapt *b* şi un fapt *c* etc. sunt adevărate". De exemplu, "*X* este sora lui *Y*, dacă *X* e de sex feminin şi *X,* şi *Y* au aceiaşi părinți" sau "*X* este o pasăre, dacă *X* este animal şi *X* are pene".

Deci, o regulă Prolog e compusă din *cap* şi *corp* separați de simbolul ":-", care se citeşte "dacă". În exemplul de mai sus *grec(X)* este capul regulii, iar *atenian(X)* corpul său.

Menționăm că variabilele sunt locale în regula în care figurează. Această înseamnă, de exemplu, că dacă sunt regulile

```
grec(X):-atenian(X).
grec(X):-spartan(X).
```

variabila *X,* apărută în prima regulă, nu are nimic comun cu cea din regula a doua, cu toate că au acelaşi nume. Regulile de mai sus sunt "traduceri" ale expresiilor logice: "Pentru orice *X*, *X* este grec, dacă *X* este atenian" sau "Pentru orice *X*, *X* este grec, dacă *X* este spartan". Dar în astfel de expresii numele variabilelor nu este semnificativ. Astfel, regulile de mai sus reprezintă acelaşi lucru ca şi regulile

```
grec(X):-atenian(X).
grec(Y):-spartan(Y).
```

Să vedem acum cum poate fi parcursă o regulă. Regula $p:\text{-}q_1,q_2,...,q_n$ se poate citi: pentru a trece *p,* trebuie de trecut succesiv $q_1,q_2,...,q_n$. Să detaliem acest mecanism asupra unui exemplu. Fie baza de fapte:

```
barbat(paul).
barbat(andrei).
femeie(maria).
femeie(eliza).
femeie(emilia).
parinti(emilia, maria, paul).
parinti(andrei, maria, paul).
parinti(eliza, maria, paul).
```

unde semnificația faptului *parinti(E, M, T)* este "*M* şi *T* sunt părinții lui *E*, *M* fiind mama, iar *T* - tata". Să definim regula *sora_lui*, în conformitate cu definițiile date mai sus (caracterul % indică un comentariu).

```
sora_lui(X,Y):- % X e sora lui Y, dacă
        femeie(X), % X e femeie şi
        parinti(X,M,T), % părinții lui X, M şi T, sunt
        parinti(Y,M,T). % şi părinții lui Y
```

Considerăm acum o întrebare mai complicată care conține o variabilă şi apelează la backtracking. Aici se va plasa marcajul, fiindcă vor fi soluții de alternativă.

```
?-sora_lui(eliza, X)

Y=emilia
Y=andrei
Y=eliza
```

Să revenim pentru un moment la răspunsul obținut. Nu e curios acest răspuns? Întrebarea pusă, de fapt, este "Cui Eliza îi este soră?", iar Prologul, după ce a răspuns corect că Eliza este sora lui Emilia şi a lui Andrei, ne afirmă că ea este de asemenea soră sie. Din punctul de vedere al mecanismului de căutare a soluțiilor răspunsul este just. Examinând regula *sora_lui/2,* constatăm că nimic nu interzice ca *X* şi *Y* să primească aceleaşi valori. Pentru ca regula să fie într-adevăr corectă, trebuie adăugată o condiție suplimentară, ce ar spune că variabilele *X* şi *Y* nu pot desemna aceeaşi persoană. În Prolog aceasta se va scrie

```
sora_lui(X,Y):- % X e sora lui Y, dacă
        femeie(X), % X e femeie şi
        parinti(X,M,T), % părinții lui X, M şi T, sunt
        parinti(Y,M,T), % şi părinții lui Y şi
        X<>Y. % X şi Y sunt persoane diferite
```

1.3. Reguli recursive

E cunoscut faptul că definițiile unor operații sau funcții pot fi sau analitice, sau recursive. De exemplu, definiția analitică a factorialului este următoarea:

$$n!=1\times2\times...\times(n-1)\times n$$

care înseamnă că factorialul unui număr este produsul tuturor numerelor de la *1* la cel specificat.

Pe de altă parte subexpresia $1 \times 2 \times \ldots \times (n-1)$ este realmente $(n-1)!$. Prin urmare, definiția factorialului poate fi:

$$n! = n \times (n-1)!$$

care se numeşte recursivă, fiindcă factorialul se defineşte prin însăşi factorial.

Dar definițiile recursive trebuie să conțină şi partea ce nu este recursivă. Această parte se numeşte baza definiției, condiția limită sau condiția de terminare a recursiei. Necesitatea condiției e evidentă. Atunci definiția completă a factorialului este

$$n! = \begin{cases} 1, \text{ dacă } n=0 \\ n \times (n-1)!, \text{ dacă } n>0 \end{cases}$$

Definiția recursivă se bazează pe principiul inducției matematice şi este un instrument declarativ, deoarece definiția recursivă defineşte mai degrabă "ce" este funcția definită, decât "cum" ea poate fi calculată. În Prolog mai greu se exprimă definițiile analitice, fiindcă el nu posedă instrucțiuni de tipul celor din limbajele imperative (de exemplu, *while, for, repeat ... until*). Deci, recursia este larg utilizată drept tehnică de programare ce substituie deseori necesitățile iterației.

Un principiu de elaborare a programelor recursive în Prolog poate fi:

- Pentru a soluționa problema de ordinul *N*, se presupune că această problemă este rezolvată pentru un ordin mai mic (fie *N-1*) şi apoi se leagă rezultatul problemei mici cu rezultatul problemei inițiale
- În final se scriu condițiile limită corespunzătoare

Astfel definiția corectă a predicatului *factorial/2* poate avea forma:

```
factorial(0,1).
factorial(N,Fac):-
       N>0,
       N1 is N-1,
       factorial(N1,Fac1),
       Fac is Fac1*N.
```

Deci, regula recursivă este regula în corpul căreia există cel puțin un apel la capul regulii.

Acum putem defini predicatul *cale(A,B),* care este adevărat dacă există o cale din oraşul *A* în oraşul *B*, adică se poate pleca din oraşul *A* spre oraşul *B*, dacă ele sunt legate cu o rută directă sau există un oraş *C* legat cu *A* cu o rută directă, din care se poate apoi pleca în oraşul *B*.

```
cale(OrasA, OrasB):-
       ruta(OrasA, OrasB).
cale(OrasA, OrasB):-
       ruta(OrasA, OrasC),
       cale(OrasC, OrasB).
```

Menționăm că acest predicat nu poate evita buclele. O variantă mai generală va fi adusă la momentul potrivit în secțiunea următoare.