- **Introducem acum relatii de familie suplimentare:**

  - **relatia *mama*, definita astfel:**

  mama (X, Y) :- parinte (X,Y), feminin (X).

  - **relatia *bunic*, definita astfel:**

  bunic (X, Z) :- parinte (X, Y), parinte(Y, Z).

  - **relatia *sora*, definita astfel:**

  sora (X, Y) :-
  parinte (Z, X),
  parinte (Z, Y),
  feminin (X).

- **<u>Interogarea Prologului:</u>**

  - **daca ann si pat sunt surori**

?- sora (ann, pat).

yes

  - **cine este sora lui Pat**

?- sora (X, pat).

X = ann;

X = pat

<u>Obs.</u>: Relatia *sora* ar trebui sa fie definita dupa cum urmeaza

```
sora (X, Y) :-
        parinte (Z, X),
        parinte (Z, Y),
        feminin (X),
        diferit (X, Y).
```

unde predicatul diferit (X,Y) trebuie definit astfel incat el sa fie satisfacut daca si numai daca X e diferit de Y.

Reguli recursive

- **relatia *predecesor*: un predecesor poate fi *direct* (parinte)**

predecesor (X, Z) :- parinte (X, Z).

sau *indirect*

```
predecesor (X, Z) :-
        parinte (X, Y),
        parinte (Y, Z).

predecesor (X, Z):-
        parinte (X, Y1),
        parinte (Y1, Y2),
        parinte(Y2, Z).
```

OBS.: Pentru ca relatia *predecesor* sa lucreze corect in cazul predecesorilor aflati la orice adancime, definitia ei trebuie gandita in felul urmator:

- **Pentru toti X si Z,**

  X este predecesor al lui Z daca

  Exista un Y astfel incat

  (1) X este parinte al lui Y si

  (2) Y este un predecesor al lui Z.

Clauza Prolog corespunzatoare (*recursiva*) este:

```
Predecesor (X,Z) :-
        parinte (X,Y),
        predecesor (Y,Z).
```

Definirea relatiei *predecesor*:

- **consta din doua reguli, una care se refera la predecesori *directi* si cealalta la predecesori *indirecti*. Definitia completa este urmatoarea:**

```
predecesor (X, Z) :-
        parinte (X,Z).

predecesor (X, Z) :-
        parinte (X, Y),
        predecesor (Y, Z).
```

Exemplu de interogare a Prologului (care sunt succesorii lui Pam):

```
?- predecesor (pam, X).

X = bob;

X = ann;

X = pat;

X = jim
```

Semnificatia declarativa si procedurala a programelor Prolog

Semnificatia declarativa a unui program Prolog se refera la *interpretarea strict logica a clauzelor acelui program*, rezultatul programului fiind reprezentat de toate consecintele logice ale acestuia. Semnificatia declarativa determina daca un scop este adevarat (poate fi satisfacut) si, in acest caz, pentru ce instante de variabile este adevarat scopul.

Semnificatia procedurala a unui program Prolog se refera la *modul in care sistemul incearca satisfacerea scopurilor*, deci la *strategia de control* utilizata.

Diferenta dintre semnificatia declarativa si cea procedurala este aceea ca cea de-a doua defineste, pe langa *relatiile logice* specificate de program, si *ordinea* de satisfacere a scopurilor si subscopurilor.

OBS.1: Datorita semnificatiei procedurale a limbajului Prolog, trebuie urmarit cu atentie modul de definire a unui predicat, atat din punctul de vedere al ordinii clauzelor, cat si din punctul de vedere al ordinii scopurilor in corpul regulilor.

OBS. 2: Ceea ce este neobisnuit in raport cu alte limbaje de programare este faptul ca semnificatia declarativa a programului este corecta, indiferent de ordonarea clauzelor, in timp ce programul este procedural incorect, avand *comportari diferite* in functie de aceasta ordonare.

Satisfacerea clauzelor in Prolog (Backtracking)

- **Presupunem ca programul nu contine reguli, ci numai fapte**

- **Presupunem ca avem un program ce contine si reguli**

1. Programul nu contine reguli, ci numai fapte:

Daca *scopul* are forma p_1, p_2, … , p_n, se incearca satisfacerea lui de la stanga la dreapta. Atunci cand *acesta nu contine variabile*, daca baza de fapte satisface intreg scopul, Prolog raspunde cu YES, altfel cu NO. Daca *scopul contine variabile*, atunci obiectivul este acela de a gasi toate legaturile posibile pentru variabile, care sa satisfaca scopul (deci de a da toate solutiile). Mai precis, se parcurge baza de fapte si se satisface o data p_1. Se trece apoi la satisfacerea lui p_2. Daca p_2 este satisfacut, atunci se trece la p_3. Daca p_2 nu se satisface, atunci se dezleaga variabilele lui p_1 si se inspecteaza baza de fapte pentru a gasi alte valori care legate de variabilele lui p_1 sa resatisfaca p_1, apoi se reincearca satisfacerea lui p_2 cu noile legaturi. Blocarea acestui proces de Backtracking se poate face cu predicatul ! ("cut"), asezat intre doua clauze consecutive p_i si p_{i+1} ale scopului. In acest caz, nu se incearca resatisfacerea lui p_i.

Daca programul contine si reguli:

Satisfacerea scopului se realizeaza in urmatorii pasi:

a) **Presupunem: clauzele $p_1,\ldots, p_i$ au fost satisfacute.**

b) **In vederea satisfacerii lui p_{i+1} se inspecteaza mai intai baza de fapte. Daca p_{i+1} se satisface cu o fapta, se trece la p_{i+2}. Daca nu exista elemente in baza de fapte care sa satisfaca pe p_{i+1}, se inspecteaza baza de reguli si se identifica prima dintre regulile neluate in consideratie pana in prezent, al carei *cap* se poate *unifica* cu p_{i+1} : se intra in corpul regulii identificate, considerand clauzele care il compun drept componente ale scopului. Daca una din clauzele corpului nu poate fi satisfacuta, se identifica o alta regula al carei cap sa se unifice cu p_{i+1}. In cazul in care nu exista o asemenea regula, se incearca resatisfacerea clauzelor p_i, p_{i-1}, …**

***Unificare*: Satisfacerea unui scop de tipul X = Y se face prin incercarea de a *unifica* X cu Y. Din aceasta cauza, dandu-se un scop de tipul X = Y, regulile de decizie care indica daca scopul se indeplineste sau nu sunt:**

1. **Daca X este variabila neinstantiata, iar Y este instantiata la orice obiect Prolog, atunci scopul reuseste. Ca efect lateral, X se va instantia la aceeasi valoare cu cea a lui Y.**
2. **Daca atat X cat si Y sunt variabile neinstantiate, scopul X = Y reuseste, variabila X este legata la Y si reciproc, i.e.: ori de cate ori una dintre cele doua variabile se instantiaza la o anumita valoare, cealalta variabila se va instantia la aceeasi valoare.**
3. **Atomii si numerele sunt intotdeauna egali cu ei insisi.**
4. **Doua structuri (a se vedea sintaxa limbajului) sunt egale daca au acelasi functor, acelasi numar de componente si fiecare componenta dintr-o structura este egala cu componenta corespunzatoare din cealalta structura.**

Sintaxa limbajului Prolog

- **Constantele definesc:**
  - **Obiecte particulare**
  - **Relatii particulare**

❖ **Constantele sunt de urmatoarele *tipuri*:**
  - **atomi – constante simbolice (desemneaza predicate Prolog sau argumente ale predicatelor)**
  - **numere**

Atomii **pot fi construiti in 3 moduri; ei pot constitui:**

1) **siruri de litere, cifre si caracterul "underscore", '_', siruri care incep cu o litera mica;**
2) **siruri de caractere speciale;**
3) **siruri de caractere incluse intre paranteze simple (ex.: 'South_America').**

***Numerele*: majoritatea implementarilor admit o gama de valori cuprinse intre –16383 si +16383, dar gama poate fi si mai larga.**

- **Variabilele:**
  - ✓ **au denumiri care incep cu litere mari; numele de variabila poate incepe si cu simbolul "_", ceea ce indica o *variabila anonima*;**
  - ✓ **utilizarea unei variabile anonime semnifica faptul ca nu intereseaza valoarea la care se va instantia acea variabila;**
  - ✓ **scopul lexical al unui nume de variabila il constituie o unica clauza. (Daca, spre exemplu, numele X15 intervine in doua clauze diferite, atunci el semnifica doua variabile diferite. Fiecare ocurenta a lui X15 in interiorul aceleiasi clauze semnifica insa o aceeasi variabila).**

- **Structurile sunt obiecte ce desemneaza o colectie de obiecte corelate logic, care formeaza componentele structurii.**

  Ex: poseda(mihai, *carte(prolog, clocksin, 1981)*)

- **Sunt folosite la reprezentarea structurilor de date (liste, arbori)**
- **O structura se defineste prin specificarea:**
  - ***numelui* structurii (*functorul* structurii);**
  - ***elementelor* sau *componentelor* structurii.**
- **Desi alcatuite din mai multe componente, sunt tratate de program ca obiecte unice. Pentru combinarea componentelor intr-un unic obiect, este folosit un functor. Acesta va fi radacina arborelui intr-o reprezentare arborescenta.**