Laborator 2

2017-2018

Programare Logică

Laboratorul 2

TODO

- Cum răspunde Prolog întrebărilor.
- Aritmetica în Prolog.
- Recursivitate în Prolog.
- Exerciții.

Material suplimentar

- `Capitolul 2`, `Capitolul 3` și `Capitolul 5` din *Learn Prolog Now!*.

Cum răspunde Prolog întrebărilor

Unificare

- În acest laborator prezentăm doar intuitiv ce înseamnă un unificator.
- Mai multe detalii și algoritmul lui Robinson care găsește un unificator pentru o mulțime de termeni, vor fi prezentate în cadrul cursului.

Unificare

- Prolog are un operator (infixat) pentru egalitate:
  `t = u` (sau echivalent `=(t,u)`)
- Ecuația $t = u$ este o țintă de bază, cu o semnificație specială.
- Ce se întâmplă dacă punem următoarele întrebări:

```
?- X = c.
?- f(X,g(Y,Z)) = f(c,g(X,Y)).
?- f(X,g(Y,f(X))) = f(c,g(X,Y)).
```

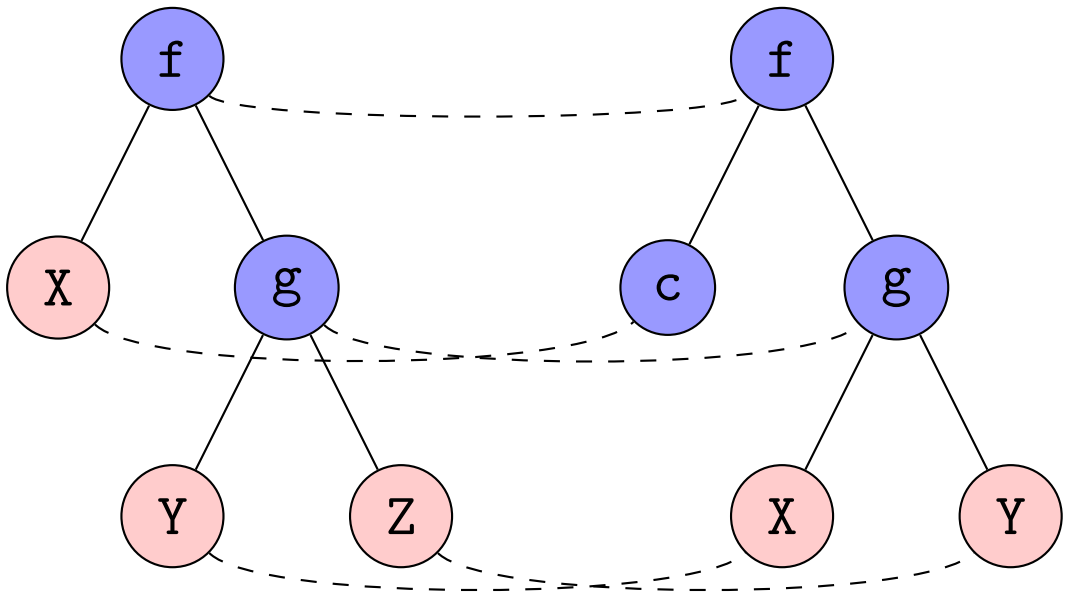
- Cum găsește aceste răspunsuri?

Unificare

- O substituție este o funcție (parțială) de la variabile la termeni.
  - $X_1 = t_1, \ldots, X_n = t_n$
- Pentru doi termeni t și u, cu variabilele $X_1, \ldots, X_n$, un unificator este o substituție care aplicată termenilor t și u îi face identici.

Exemplu

f(X,g(Y,Z)) = f(c,g(X,Y))

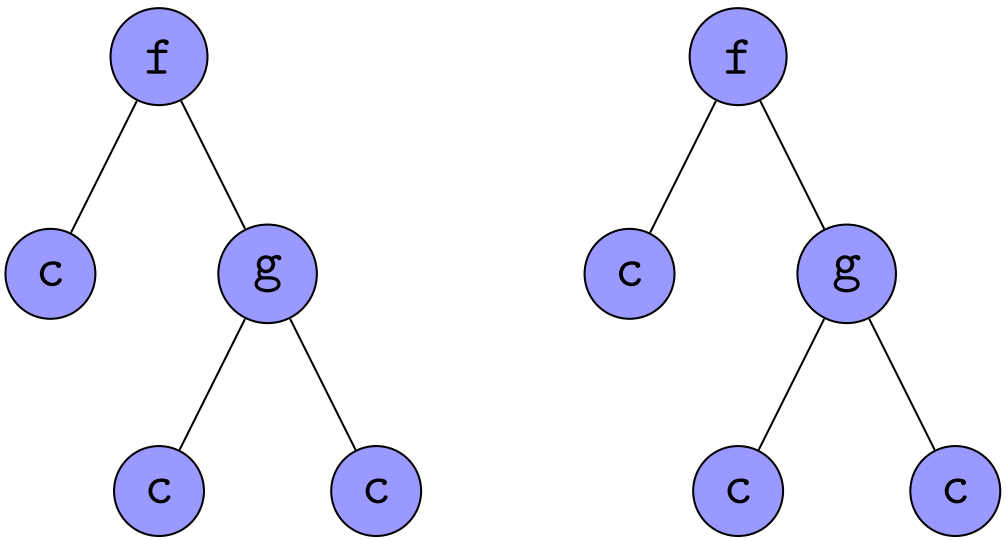


X=c
Y=X
Z=Y

Exemplu: aplicând substituția

f(X,g(Y,Z)) = f(c,g(X,Y))

X=c
Y=c
Z=c



Unificare

- Ce se întâmplă dacă încercăm să unificăm X cu ceva care conține X? Exemplu: `?- X = f(X).`
- Conform teoriei, acești termeni nu se pot unifica.
- Totuși, multe implementări ale Prolog-ului sar peste această verificare din motive de eficiență.
  - putem folosi `unify_with_occurs_check/2`

Ce se întamplă în Prolog când punem o întrebare?

- Folosește unificarea pentru a potrivi țintele și clauzele (reguli și fapte).
- Poate găsi zero, una sau mai multe soluții.
- Execuția se poate întoarce (*backtrack*).

Procesul din spatele Prolog-ului se numește rezoluție SLD (*backchain*).

În acest laborator prezentăm doar ideea intuitivă din spatele Prolog-ului. Detaliile tehnice vor fi prezentate în cadrul cursului.

Căutare depth-first

Ideea de bază:

Pentru a rezolva o țintă A:

- **dacă** B este un fapt în program și există o substituție θ astfel încât $\theta(A) = \theta(B)$, atunci întoarce răspunsul θ;
- **altfel**
  - **dacă** $B :- G_1, \ldots, G_n$ este o regulă în program și θ unifică A și B, atunci rezolvă $\theta(G_1), \ldots, \theta(G_n)$,
  - **altfel** renunță la această țintă:
    - întoarce-te la ultima decizie

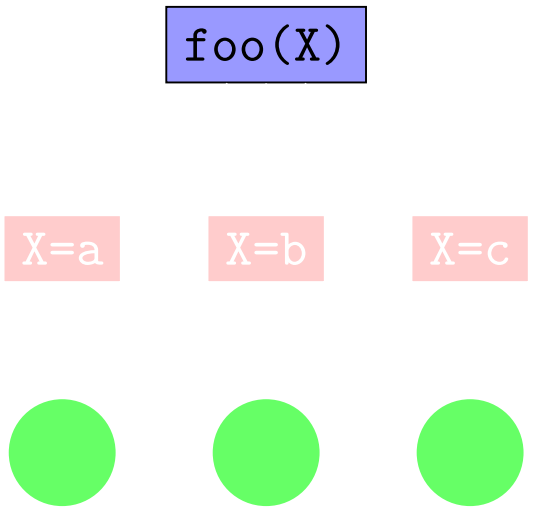
- Clauzele sunt verificate în ordinea declarării!!
- Țintele compuse (cu mai multe predicate) sunt verificate de la stânga la dreapta!!

Căutare depth-first - arbori de căutare

Prolog încearcă clauzele în ordinea apariției lor în program.

Exemplu:
Să presupunem că avem programul: `foo(a).  foo(b).  foo(c).`
Punem intrebarea:

```
?- foo(X).
```

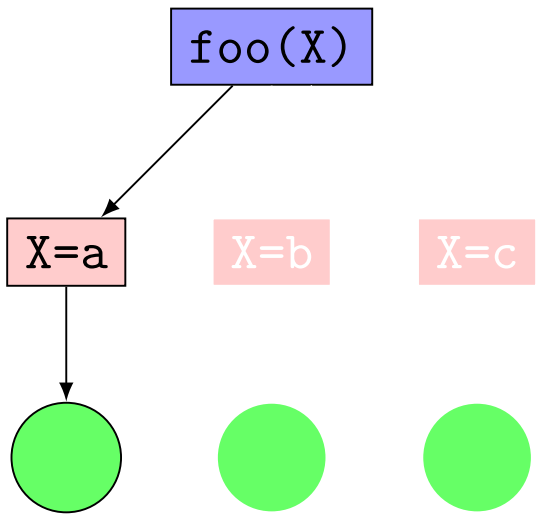


Căutare depth-first - arbori de căutare

Prolog incearcă clauzele în ordinea apariției lor în program.

Exemplu:
Să presupunem că avem programul: `foo(a).  foo(b).  foo(c).`
Punem intrebarea:

```
?- foo(X).
X = a
```

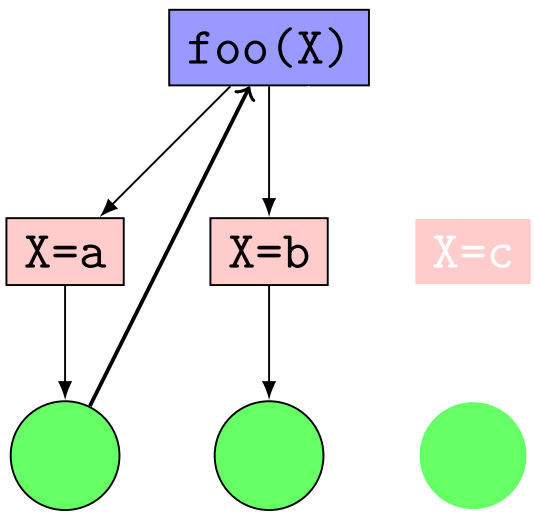


Căutare depth-first - arbori de căutare

Prolog incearcă clauzele în ordinea apariției lor în program.

Exemplu:
Să presupunem că avem programul: `foo(a).  foo(b).  foo(c).`
Punem intrebarea:

```
?- foo(X).
X = a
X = b
```

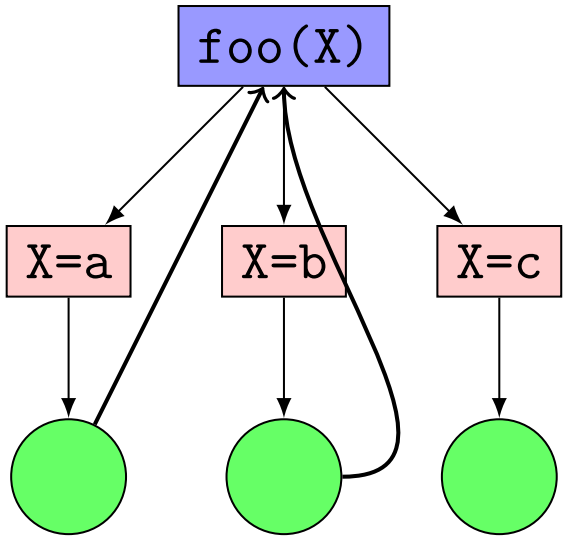


Căutare depth-first - arbori de căutare

Prolog incearcă clauzele în ordinea apariției lor în program.

Exemplu:
Să presupunem că avem programul: `foo(a).  foo(b).  foo(c).`
Punem intrebarea:

```
?- foo(X).
X = a
X = b
X = c
```

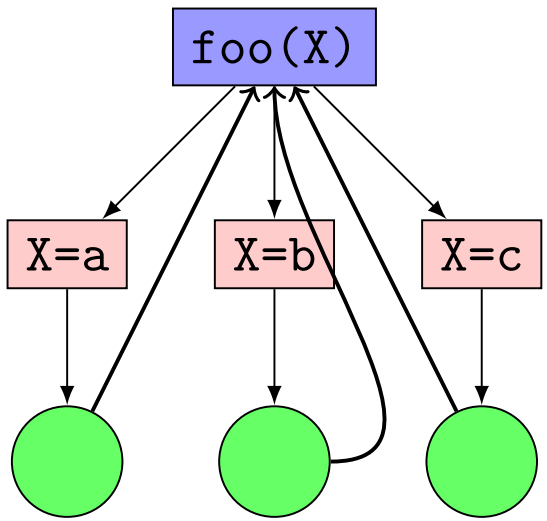


Căutare depth-first - arbori de căutare

Prolog incearcă clauzele în ordinea apariției lor în program.

Exemplu:
Să presupunem că avem programul: `foo(a).  foo(b).  foo(c).`
Punem intrebarea:

```
?- foo(X).
X = a
X = b
X = c
```

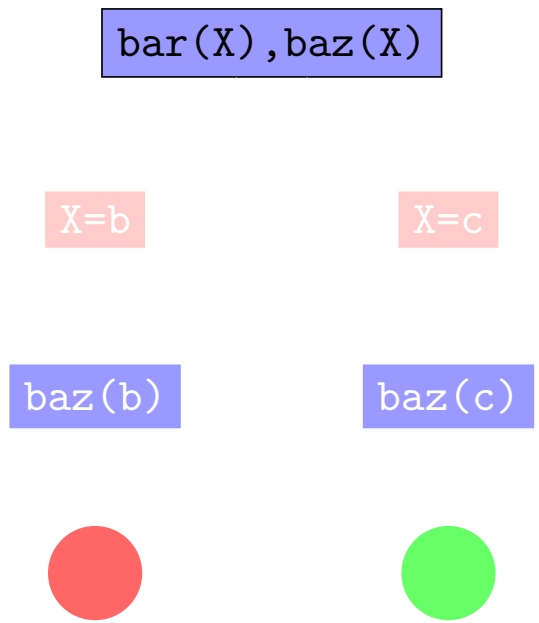


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

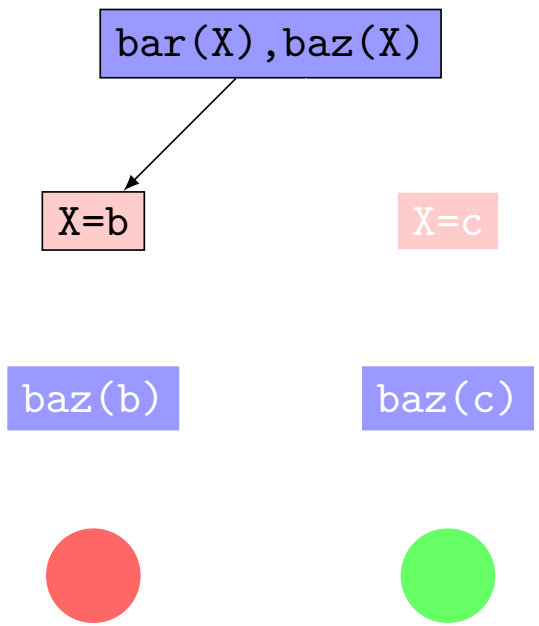


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

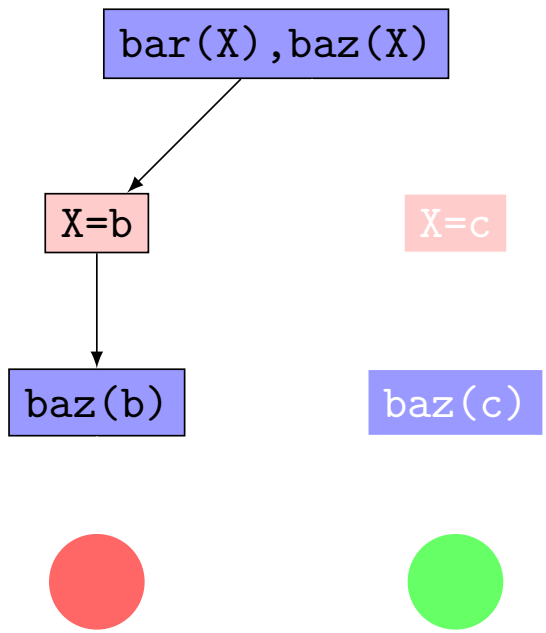


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

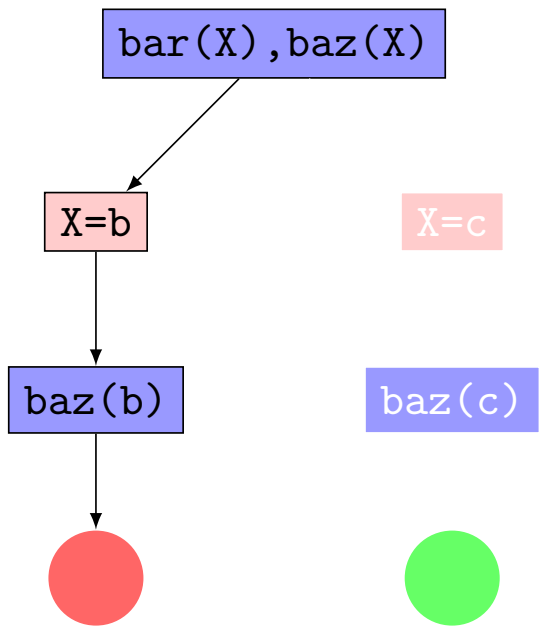


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

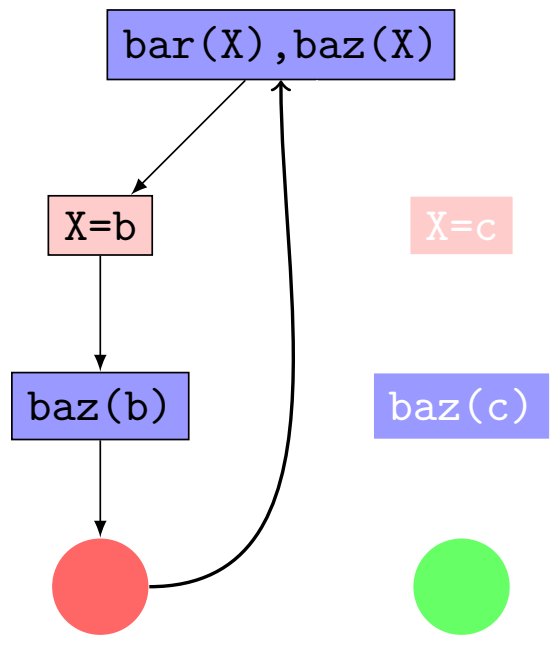


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

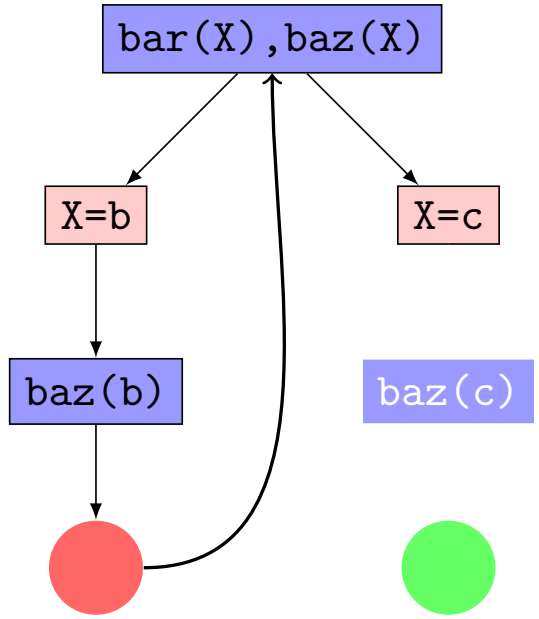


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).  bar(c).  baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

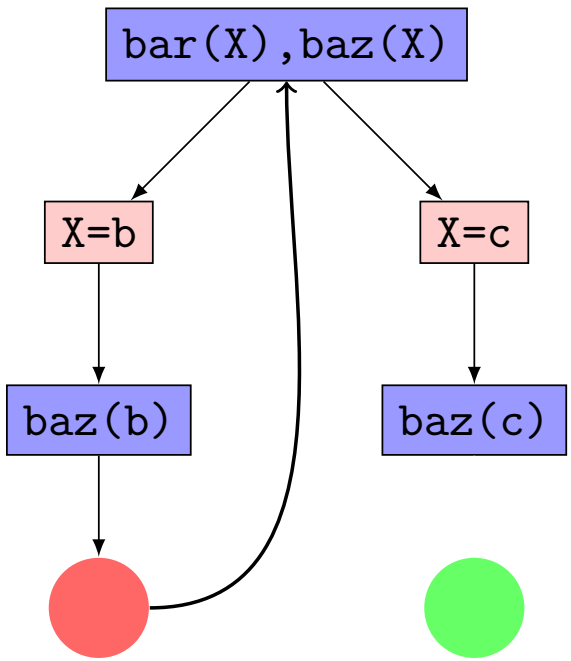


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).  bar(c).  baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
```

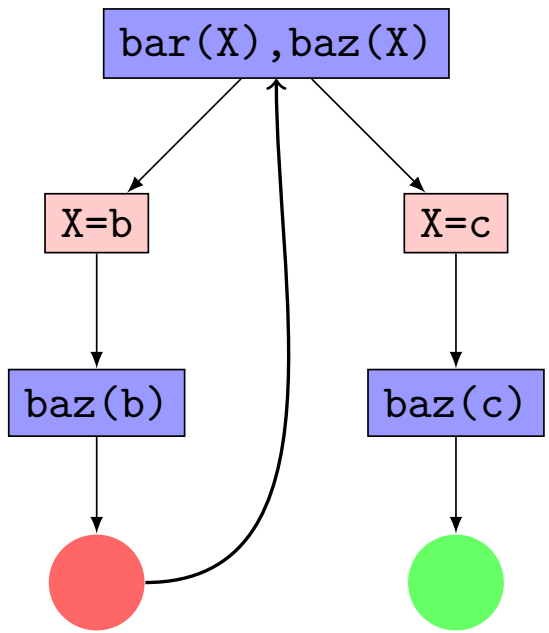


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).` `bar(c).` `baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
X = c
```

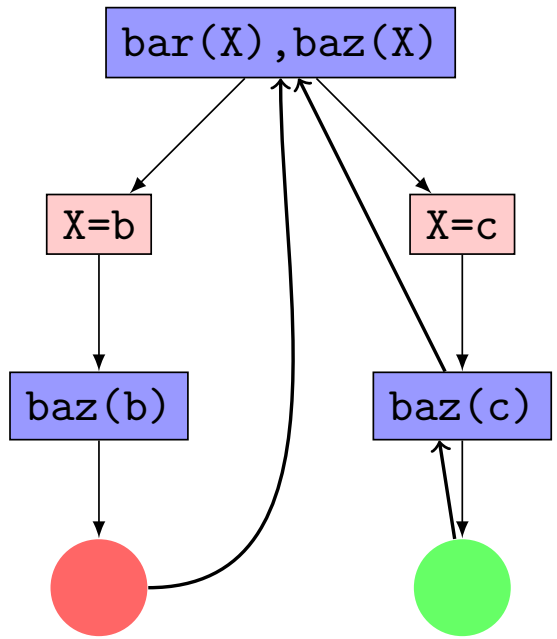


Căutare depth-first - arbori de căutare (cont.)

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu:
Să presupunem că avem programul: `bar(b).  bar(c).  baz(c).`
Punem intrebarea:

```
?- bar(X),baz(X).
X = c
```



Afișări în Prolog

- Pentru a afișare se foloseste predicatul `write/1`.
- Predicatul `nl/0` conduce la afișarea unei linii goale.

Exemplu

```
?- write('Hello World!'), nl.
Hello World!
true

?- X = hello, write(X), nl.
hello
X = hello
```

Aritmetica în Prolog

Aritmetica în Prolog

Exemplu

```
?- 3+5 = +(3,5).
true
?- 3+5 = +(5,3).
false
?- 3+5 = 8.
false
```

Explicații:

- 3+5 este un termen.
- Prolog trebuie anunțat explicit pentru a îl evalua ca o expresie aritmetică, folosind predicate predefinite în Prolog, cum sunt `is/2`, `=:=/2`, `>/2` etc.

Aritmetica în Prolog

Exercițiu. *Analizați următoarele exemple:*

```
?- 3+5 is 8.
false

?= X is 3+5.
X = 8

?- 8 is 3+X.
is/2: Arguments are not sufficiently instantiated

?- X=4, 8 is 3+X.
false
```

Aritmetica în Prolog

Exercițiu. *Analizați următoarele exemple:*

```
?- X is 30-4.
X = 26

?- X is 3*5.
X = 15

?- X is 9/4.
X = 2.25
```

Aritmetica în Prolog

Operatorul `is`:

- Primește două argumente
- Al doilea argument trebuie să fie o expresie aritmetică validă, cu toate variabilele inițializate
- Primul argument este fie un număr, fie o variabilă
- Dacă primul argument este un număr, atunci rezultatul este `true` dacă este egal cu evaluarea expresiei aritmetice din al doilea argument.
- Dacă primul argument este o variabilă, răspunsul este pozitiv dacă variabila poate fi unificată cu evaluarea expresiei aritmetice din al doilea argument.

Totuși, nu este recomandat să folosiți `is` pentru a compara două expresii aritmetice, ci operatorul `=:=`.

Aritmetica în Prolog

Exercițiu. *Analizați următoarele exemple:*

```
?- 8 > 3.
true

?- 8+2 > 9-2.
true

?- 8 < 3.
false
```

```
?- 8 >= 3.
true

?- 8 =:= 3.
false

?- 8 =\= 3.
true
```

Operatori aritmetici

Operatorii aritmetici predefiniți în Prolog sunt de două tipuri:

- funcții
- relații

Funcții

- Adunarea și înmulțirea sunt exemple de funcții aritmetice.
- Aceste funcții sunt scrise în mod uzual și în Prolog.

Exemplu

```
2 + (-3.2 * X - max(17,X)) / 2 ** 5
```

- 2**5 înseamnă 2^5
- Exemple de alte funcții disponibile:
  `min/2`, `abs/1` (modul), `sqrt/1` (radical), `sin/1` (sinus)
- Operatorul `//` este folosit pentru împărțire întreagă.
- Operatorul `mod` este folosit pentru restul împărțirii întregi.

Relații

- Relatiile aritmetice sunt folosite pentru a compara evaluarea expresiilor aritmetice (e.g, `X > Y`)
- Exemple de relații disponibile:
  <, >, <=, >=, =/= (diferit), =:= (aritmetic egal)
- Atenție la diferența dintre =:= și =:
  - =:= compară două expresii aritmetice
  - = caută un unificator

Exemplu

```
?- 2 ** 3 =:= 3 + 5.
true
?- 2 ** 3 = 3 + 5.
false
```

Recursivitate

Recursivitate - Strămoși

```
parent_of(rickardStark,eddardStark).
parent_of(rickardStark,lyannaStark).
parent_of(lyarraStark,eddardStark).
parent_of(lyarraStark,lyannaStark).

parent_of(aerysTargaryen,rhaegarTargaryen).
parent_of(rhaellaTargaryen,rhaegarTargaryen).

parent_of(rhaegarTargaryen,jonSnow).
parent_of(lyannaStark,jonSnow).
```

Recursivitate - Strămoși (cont.)

Vrem sa definim un predicat `ancestor_of(X,Y)` care este adevărat dacă X este un strămoș al lui Y.

Definim predicatul strămoș în mod recursiv:

```
ancestor_of(X,Y) :- parent_of(X,Y).
ancestor_of(X,Y) :- parent_of(X,Z), ancestor_of(Z,Y).
```

Exerciții

Practică

Exercițiul 1: distanța dintre două puncte

Definiți un predicat `distance/3` pentru a calcula distanța dintre două puncte într-un plan 2-dimensional. Punctele sunt date ca perechi de coordonate.

Exemple:

```
?- distance((0,0), (3,4), X).
X = 5.0

?- distance((-2.5,1), (3.5,-4), X).
X = 7.810249675906654
```

Practică

Exercițiul 2: afișarea unui pătrat de caractere

Scrieți un program în Prolog pentru a afișa un pătrat de $n \times n$ caractere pe ecran.

Denumiți predicatul `square/2`. Primul argument este un număr natural diferit de 0, iar al doilea un caracter (i.e, orice termen în Prolog) care trebuie afișat.

Exemplu:

```
?- square(5, '* ').
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Practică

Exercițiul 3: numerele Fibonacci

Scrieți un predicat `fib/2` pentru a calcula al n-ulea număr Fibonacci. Secvența de numere Fibonacci este definită prin:

$$\begin{aligned} F_0 &= 1 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-1} \text{ for } n \geq 2 \end{aligned}$$

Exemple:

```
?- fib(1,X).
X=1.
true

?- fib(2,X).
X=2.
true
```

```
?- fib(5,X).
X=8.
true
```

Practică

Exercițiul 3 (cont.)

Programul scris anterior vă găsește răspunsul la întrebarea de mai jos?

```
?- fib(50,X).
```

Dacă da, felicitări! Dacă nu, încercați să găsiți o soluție mai eficientă!

Hint: Încercați să construiți toate numerele Fibonacci până ajungeți la numărul căutat.

Pe data viitoare!