Laborator 2 – Prolog

2016-2017

Programare Logică

Laboratorul 2

TODO

- Cum răspunde Prolog întrebărilor.
- Aritmetica în Prolog.

Material suplimentar

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

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)).
```

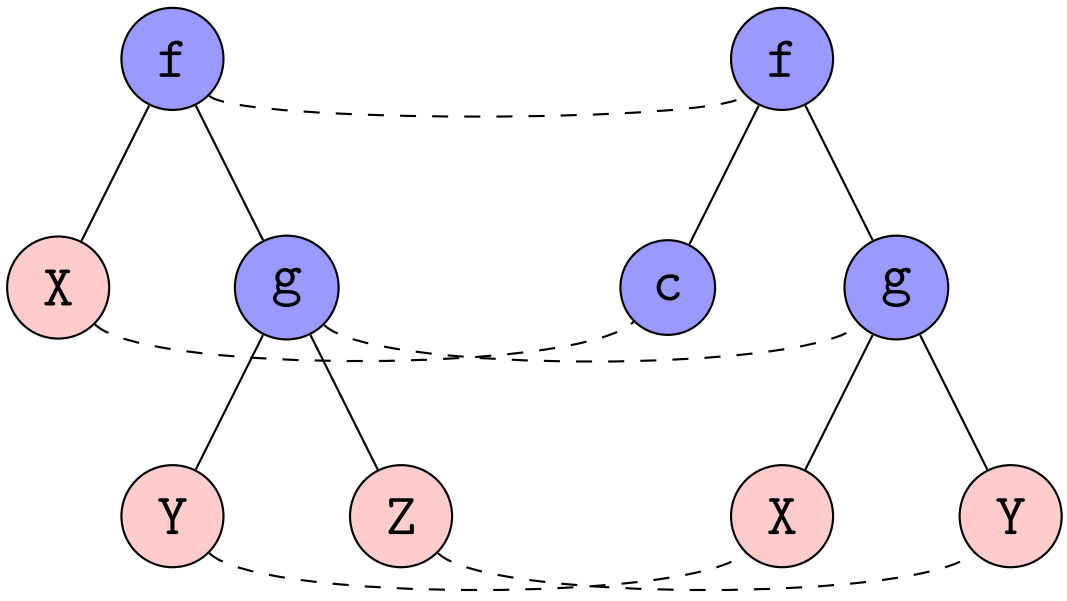
- Cum găsește aceste răspunsuri?

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))

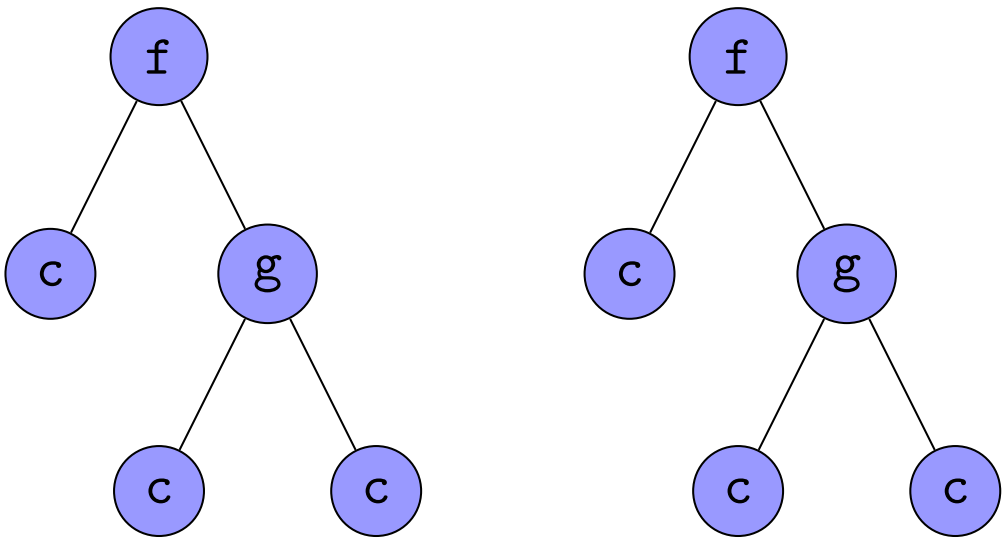


X=c
Y=X
Z=Y

Exemplu: aplicând substituția

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

X=c
Y=c
Z=c



Unificare

- Consultați suportul de curs pentru algoritmul lui Robinson care găsește un unificator pentru o mulțime de ecuații de forma:

$$t_1 = u_1, \quad t_2 = u_3, \quad \ldots, \quad t_n = u_n$$

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*).
Consultați suportul de curs pentru mai multe detalii.

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 încat $\theta(A) = \theta(B)$, 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

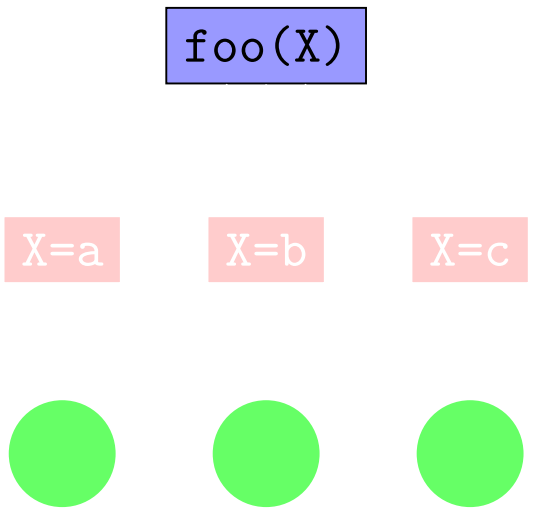
- Clauzele sunt verificate în ordinea declarării!!
- Țintele compuse (cu mai multe predicate) sunt verificate de la stânga la dreapta!!

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).
```

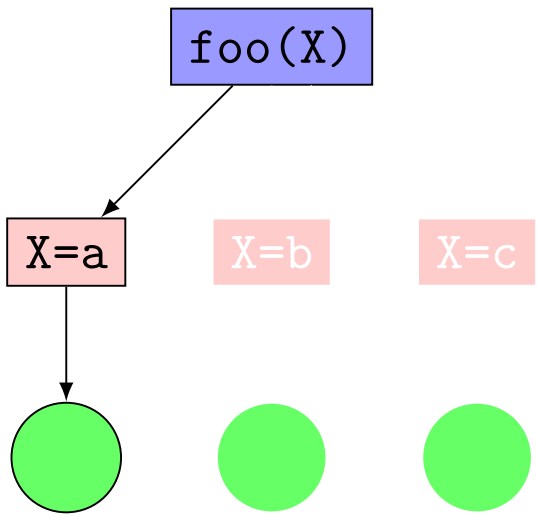


Căutare depth-first - arbori de căutare

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
```

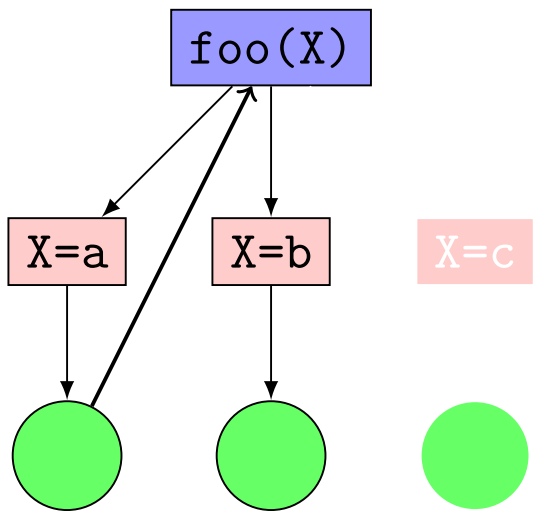


Căutare depth-first - arbori de căutare

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
```

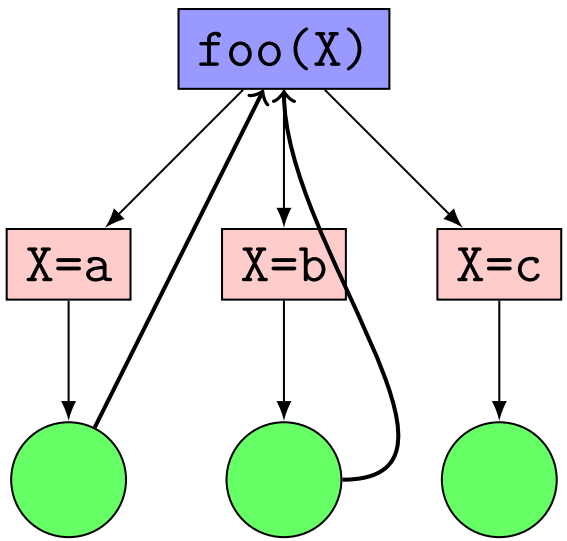


Căutare depth-first - arbori de căutare

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
```

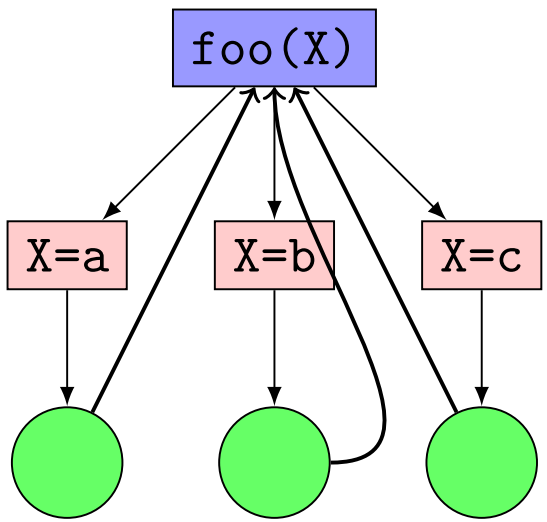


Căutare depth-first - arbori de căutare

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
```

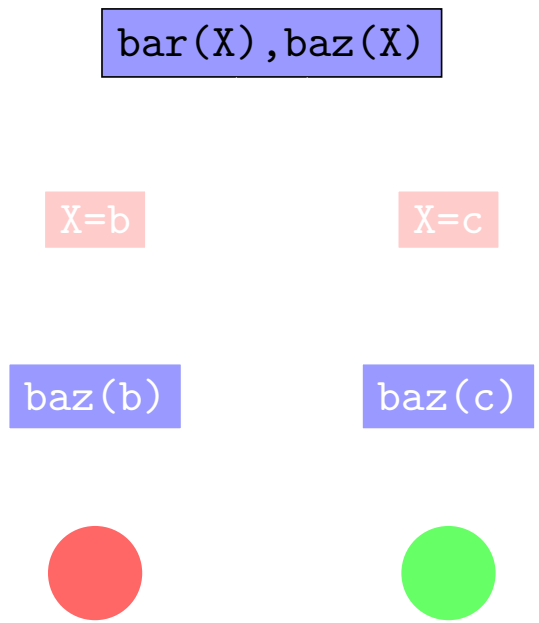


Căutare depth-first - arbori de căutare (cont.)

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).
```

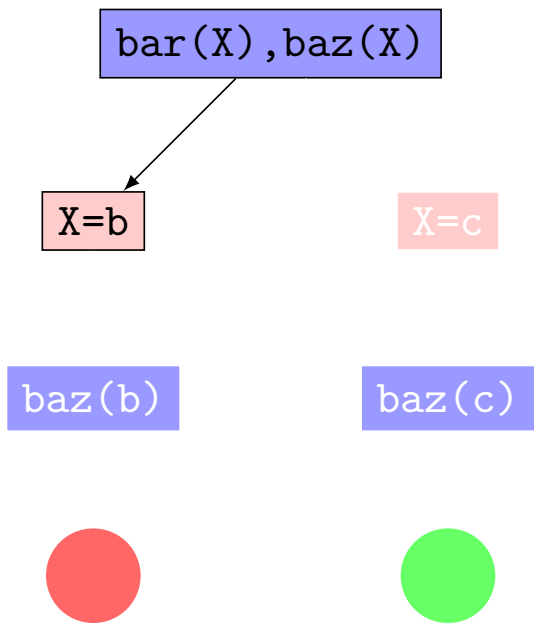


Căutare depth-first - arbori de căutare (cont.)

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).
```

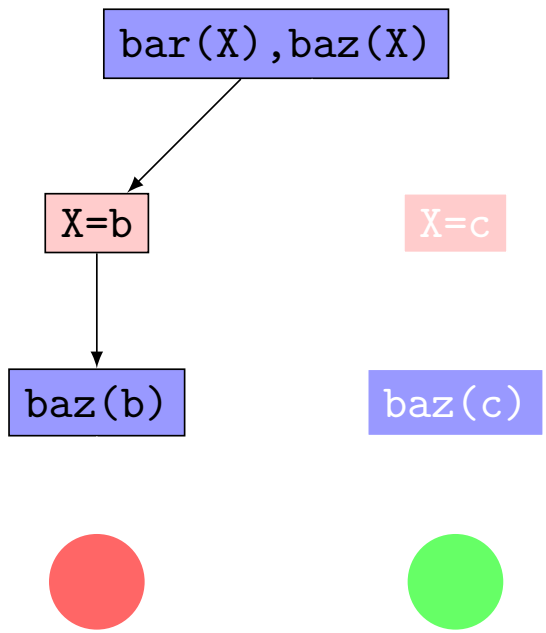


Căutare depth-first - arbori de căutare (cont.)

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).
```

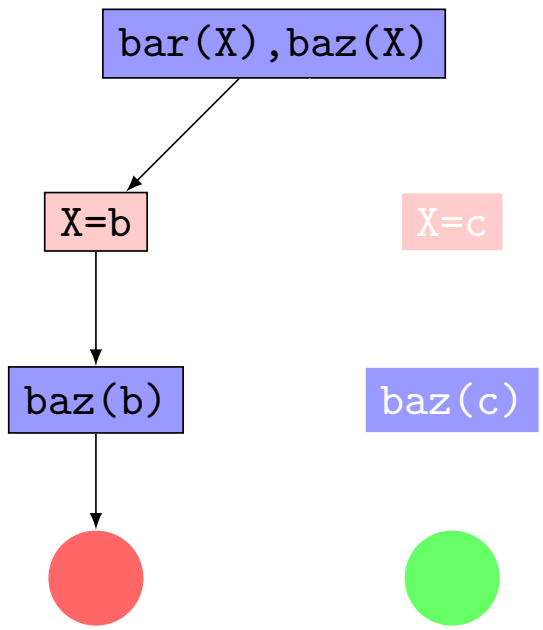


Căutare depth-first - arbori de căutare (cont.)

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).
```

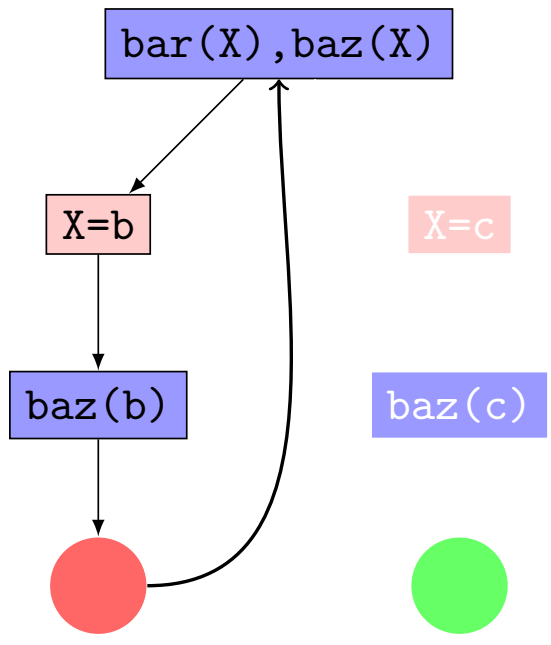


Căutare depth-first - arbori de căutare (cont.)

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).
```

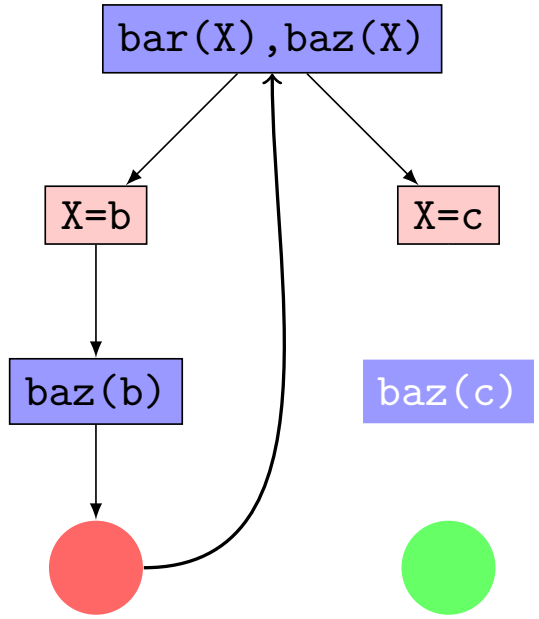


Căutare depth-first - arbori de căutare (cont.)

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).
```

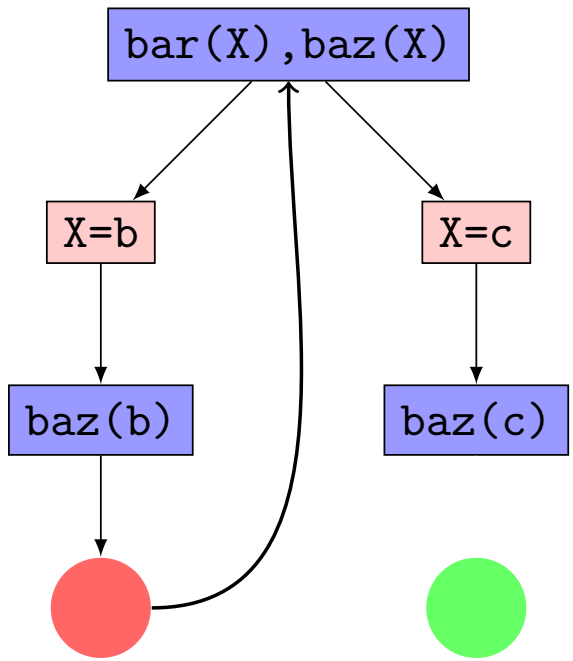


Căutare depth-first - arbori de căutare (cont.)

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).
```

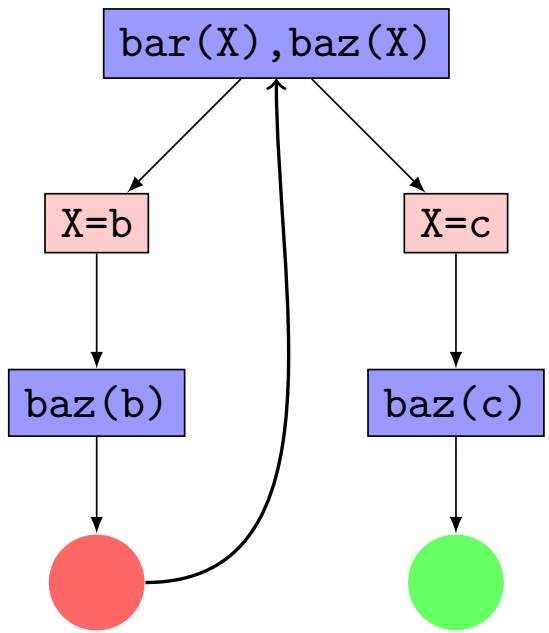


Căutare depth-first - arbori de căutare (cont.)

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
```

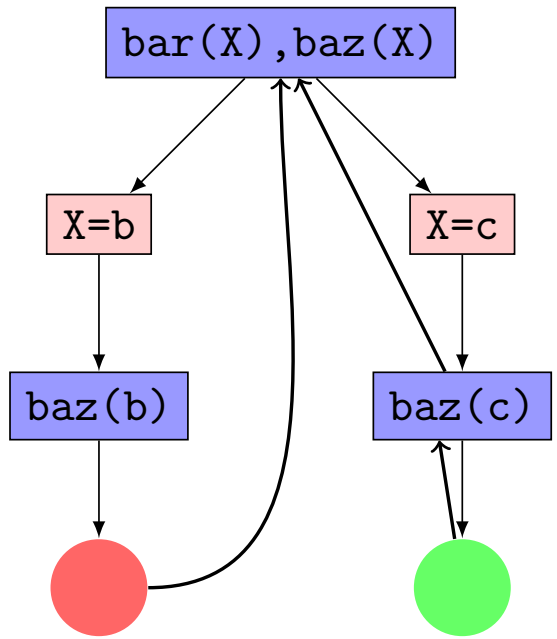


Căutare depth-first - arbori de căutare (cont.)

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
```



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

Exercițiu. *Analizați următoarele exemple:*

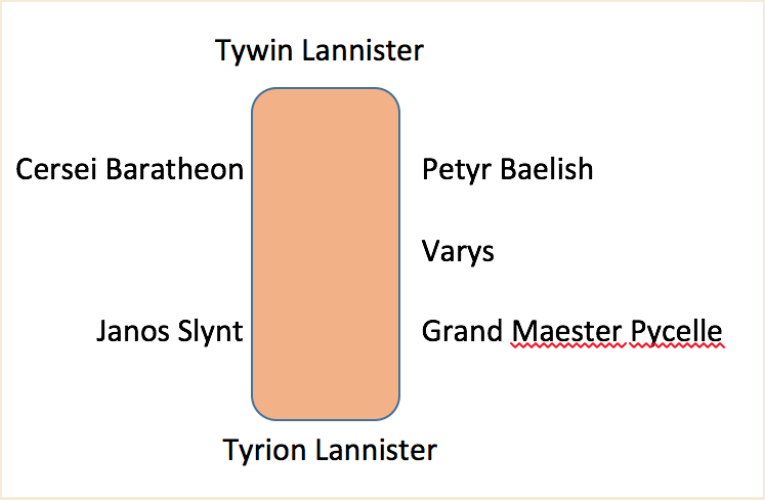
```
?- 8 > 3.
true

?- 8+2 > 9-2.
true

?- 8 < 3.
false
```

```
?- 8 >= 3.
true

?- 8 =:= 3.
false

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

Practică

Exercițiul 1: consiliu

Imaginea de mai jos arată cum sunt asezați membrii consiliului lui Joffrey:



Definiți predicatul `sits_right_of/2` pentru a reprezenta cine lângă cine stă. `sits_right_of(X,Y)` trebuie să fie adevărat dacă `X` este la dreapta lui `Y`.

Practică

Exercițiul 1 (cont.)

Adăugați următoarele predicate:

- `sits_left_of/2`: `sits_left_of(X,Y)` trebuie să fie adevărat dacă `X` este la stânga lui `Y`.
- `are_neighbors_of/3`: `are_neighbors_of(X,Y,Z)` trebuie să fie adevărat dacă `X` este la stângă lui `Z` și `Y` este la dreapta lui `Z`.
- `next_to_each_other/2`: `next_to_each_other(X,Y)` trebuie să fie adevărat dacă `X` este lângă `Y`.

Practică

Exercițiul 1 (cont.)

Testați implementarea voastră punând următoarele întrebări:

- Este Petyr Baelish la dreapta lui Cersei Baratheon?
- Este Petyr Baelish la dreapta lui Varys?
- Cine este la dreapta lui Janos Slynt?
- Cine stă doua scaune la dreapta lui Cersei Baratheon?
- Cine stă între Petyr Baelish și Grand Master Pycelle?

Pe săptămâna viitoare!