

Laboratorul 4 - Programare Logică și Funcțională

Seria 36

Martie 2025

În acest laborator vom continua să aplicăm **algoritmul de unificare** de la seminar, și vom învăța să utilizăm limbajul Prolog pentru probleme de căutare de soluții.

1 Exercițiul 1

O *substituție* este o funcție parțială de la variabile la termeni, adică $\sigma : V \rightarrow Trm_{\mathcal{L}}$. Un *unificator* pentru doi termeni t_1 și t_2 este o substituție θ astfel încât $\theta(t_1) = \theta(t_2)$. Un unificator ν pentru t_1 și t_2 este un *cel mai general unificator* dacă pentru orice alt unificator ν' pentru t_1 și t_2 , există o substituție μ astfel încât $\nu' = \nu; \mu$.

Algoritmul de unificare:

	Lista soluție S	Lista de rezolvat R
Inițial	\emptyset	$t_1 = t'_1, \dots, t_n = t'_n$
SCOATE	S	$R', t = t$
	S	R'
DESCOMPUNE	S	$R', f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)$
	S	$R', t_1 = t'_1, \dots, t_n = t'_n$
REZOLVĂ	S	$R', x = t \text{ sau } t = x, x \text{ nu apare în } t$
	$x = t, S[x \leftarrow t]$	$R'[x \leftarrow t]$
Final	S	\emptyset

Algoritmul *se termină normal* dacă $R = \emptyset$ (în acest caz, în S are un unificator pentru termenii din lista inițială R).

Algoritmul este oprit cu concluzia *inexistenței unui unificator* dacă:

- În R există o ecuație de forma $f(t_1, \dots, t_n) = g(t'_1, \dots, t'_k)$ cu $f \neq g$. Simbolurile de constantă se consideră simboluri de funcție de aritate 0.
- În R există o ecuație de forma $x = t$ sau $t = x$ și variabila x apare în termenul t .

Fie limbajul de ordinul I $\mathcal{L} = (\mathbf{F}, \mathbf{R}, \mathbf{C}, \text{ari})$ unde $\mathbf{F} := \{h, g, f, *, +, p\}$, $\mathbf{C} := \{a, b, c\}$, iar $x, y, z, u, v \in Var$, descrise astfel:

- $x, y, z, u, v \in Var$;
- $a, b, c \in \mathbf{C} (= \mathcal{F}_0)$; (doar informal! am stabilit la seminar că \mathcal{F}_m are sens pentru $m \geq 1$)
- $h, g \in \mathcal{F}_1$;
- $f, *, + \in \mathcal{F}_2$;
- $p \in \mathcal{F}_3$

unde $\mathcal{F}_m := \{f \in \mathcal{F} \mid \text{ari}(f) = m\}$.

Decideți dacă există unificatori pentru următorii termeni și verificați că Prolog răspunde conform rezultatului pe care l-ați obținut.

În Prolog, putem găsi unificatori prin apelarea predicatului

`unify_with_occurs_check/2`

Recomandare. Creați un fișier în care să definiți un predicat cu nume mai simplu pentru unificare, de exemplu

```
eq(X, Y) :- unify_with_occurs_check(X, Y).
```

Exemple:

```
?- eq(h(a, X), h(Y, b)).  
X = b, Y = a
```

```
?- eq(A, f(A)).  
false
```

De ce e **false** în al doilea caz?

Continuați cu exercițiile:

1. $f(h(a), g(x))$ cu $f(y, y)$;
2. $p(a, x, g(x))$ cu $p(a, y, y)$;
3. $p(x, y, z)$ cu $p(u, f(v, v), u)$;
4. $p(x, f(x, x))$ cu $f(g(y), f(z, g(a)))$;
5. $x + (y * y)$ cu $(y * y) + z$;
6. $f(g(x), x)$ cu $f(y, y)$;
7. $p(a, u, h(x))$ cu $p(y, f(y, z), z)$.

2 Exercițiul 2

Fie următoarele șase cuvinte:

FLOWERS

PROLOG

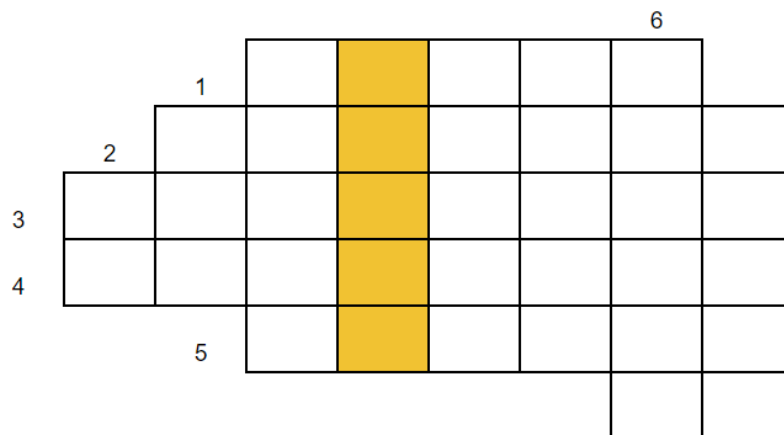
ENTIRELY

SCHOOL

OREGANO

SLEEP

Vom așeza cuvintele în următorul puzzle:



(cinci cuvinte sunt pe orizontală, iar al șaselea este pe verticală).

Pentru rezolvare:

1. Implementați o bază de cunoștințe, definită prin predicatul `word/1`.
2. Utilizați predicatul `string_chars/2`, care primește un atom și returnează lista caracterelor conținute în atomul respectiv.
3. Definiți un predicat `list_pos/3` care primește o listă, un index și returnează elementul de pe indexul dat dacă acesta există, și `false` altfel.

Veți scrie un predicat principal

```
solution(W1, W2, W3, W4, W5, W6, L) :- ...
```

care se va asigura că cele șase cuvinte sunt cele din baza de cunoștințe și sunt mutual diferite, iar în `L` va întoarce o listă conținând soluția acestui puzzle.

3 Exercițiul 3

Fie următoarea bază de cunoștințe, formată prin intermediul predicatului `connected/2`:

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

1. Scrieți un predicat `path/2` care indică dacă dintr-un punct puteți să ajungeți într-un alt punct (în mai mulți pași), legând conexiunile din baza de cunoștințe. Drumurile se consideră unidirecționale (relația nu este simetrică).

Întrebări pentru verificare:

- puteți ajunge din punctul 5 în punctul 10?
- în ce puncte puteți să ajungeți plecând din 1?
- din ce puncte puteți să ajungeți în punctul 13?

Testați programul pentru baza de cunoștințe:

```
connected(1,2).
connected(2,1).
connected(1,3).
connected(3,4).
```

cu întrebarea

```
?- path(1,4).
```

Observație. Dacă graful conține cicluri, este posibil ca programul să nu își termine execuția.

2. Scrieți un predicat care determină dacă există drumuri, evitând ciclurile din graf.

Hint. Utilizați un predicat auxiliar care reține într-o listă punctele vizitate până în momentul curent.