Laboratorul 4 - Programare Logică și Funcțională

Seria 36

Martie 2024

În acest laborator vom continua să aplicăm **algoritmul de unificare** de la seminar, şi vom învăţa să utilizăm limbajul Prolog pentru probleme de căutare de soluţii.

1 Exerciţiul 1

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

Algoritmul de unificare:

	Lista soluţie S	Lista de rezolvat R
Iniţial	$\emptyset$	$t_1 = t'_1, \ldots, t_n = t'_n$
SCOATE	S	$R', t = t$
	S	R'
DESCOMPUNE	S	$R', f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)$
	S	$R', t_1 = t'_1, \ldots t_n = t'_n$
REZOLVĂ	S	$R', x = t$ sau $t = x$, x 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 iniţială R).

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

1. În R există o ecuaţie de forma $f(t_1, \ldots, t_n) = g(t'_1, \ldots, t'_k)$ cu $f \neq g$. Simbolurile de constantă se consideră simboluri de funcţie de aritate 0.
2. În R există o ecuaţie de forma $x = t$ sau $t = x$ şi variabila x apare în termenul t.

Fie limbajul de ordinul I $\mathcal{L} = (\mathbf{F}, \mathbf{R}, \mathbf{C}, ari)$ 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 $\mathcal{F}_m := \{f \in \mathcal{F} \mid ari(f) = m\}$.

Decideţi dacă există unificatori pentru următorii termeni şi verificaţi că Prolog răspunde conform rezultatului pe care l-aţi obţinut.

În Prolog, putem găsi unificatori prin apelarea predicatului

`unify_with_occurs_check/2`

Recomandare. Creaţi un fişier în care să definiţi 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?

Continuaţi cu exerciţiile:

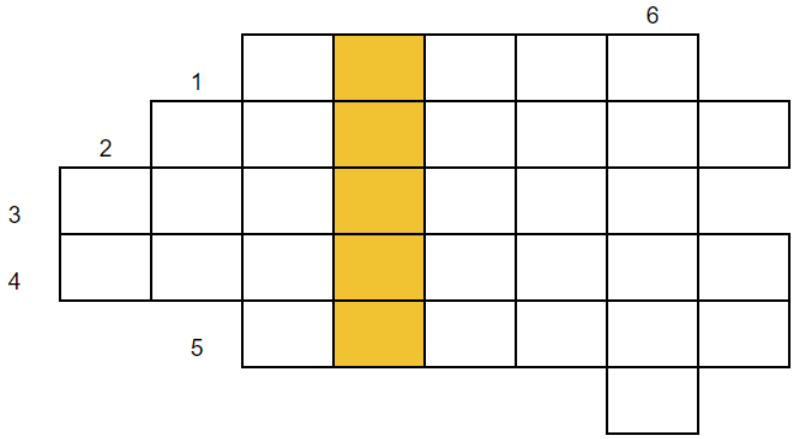
1. $f(h(a), g(x))$ cu $f(y, y)$;
2. $p(a, x, g(x))$ cu $p(a, y, y)$;
3. $p(x, y, z)$ cu $p(u, f(v, v), u)$;
4. $p(x, f(x, x))$ cu $f(g(y), f(z, g(a)))$;
5. $x + (y * y)$ cu $(y * y) + z$;
6. $f(g(x), x)$ cu $f(y, y)$;
7. $p(a, u, h(x))$ cu $p(y, f(y, z), z)$.

2 Exerciţiul 2

Fie următoarele şase cuvinte în limba engleză:

`LOVABLE` `ENERGY` `SAVAGE` `WHISKERS` `AGILE` `PURRING`

Vom aşeza cuvintele în următorul puzzle:



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

Scrieţi un algoritm în Prolog pentru a rezolva acest puzzle şi pentru a găsi soluţia (coloana portocalie), care reprezintă numele acestei pisici:

Pentru rezolvare:

1. Implementaţi o bază de cunoştinţe, definită prin predicatul `word/1`.
2. Utilizaţi predicatul `string_chars/2`, care primeşte un atom şi returnează lista caracterelor conţinute în atomul respectiv.
3. Definiţi un predicat `list_pos/3` care primeşte o listă, un index şi returnează elementul de pe indexul dat dacă acesta există, şi `false` altfel.

Veţi scrie un predicat principal

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

care se va asigura că cele şase cuvinte sunt cele din baza de cunoştinţe şi sunt mutual diferite, iar în `L` va întoarce o listă conţinând soluţia acestui puzzle.

3 Exerciţiul 3

Fie următoarea bază de cunoştinţe, 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. Scrieţi un predicat `path/2` care indică dacă dintr-un punct puteţi să ajungeţi într-un alt punct (în mai mulţi paşi), legând conexiunile din baza de cunoştinţe. Drumurile se consideră unidirecţionale (relaţia nu este simetrică).

   Întrebări pentru verificare:

   - puteţi ajunge din punctul 5 în punctul 10?
   - în ce puncte puteţi să ajungeţi plecând din 1?
   - din ce pucnte puteţi să ajungeţi în punctul 13?

   Testaţi programul pentru baza de cunoştinţe:

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

cu întrebarea

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

   Observaţie. Dacă graful conţine cicluri, este posibil ca programul să nu îşi termine execuţia.

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

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