PROLOG - Laborator

2018-2019

Programare Logică

Prolog

- Prolog este cel mai cunoscut limbaj de programare logică.
  - bazat pe logica clasică de ordinul I (cu predicate)
  - funcționează pe bază de unificare și căutare
- Multe implementări îl transformă în limbaj de programare "matur"
  - I/O, operații implementate deja în limbaj etc.

Prolog

- Vom folosi implementarea SWI-Prolog
  - gratuit
  - folosit des pentru predare
  - conține multe librării
  - `http://www.swi-prolog.org/`
- Varianta online SWISH a SWI-Prolog
  - `http://swish.swi-prolog.org/`

Bibliografie

- http://www.learnprolognow.org
- http://cs.union.edu/~striegnk/courses/esslli04prolog

Cuprins

Elemente de sintaxă

Sintaxă: atomi

Atomi:

- secvențe de litere, numere și _, care încep cu o literă mică
- șiruri între apostrofuri `'Atom'`
- anumite simboluri speciale

Exemplu

- `sansa`
- `jon_snow`
- `'Ser Gregor Clegane'`
- `'(@ *+'`
- `  +`

```
?- atom('(@ *+ ').
true.
```

`atom/1` este un predicat predefinit

Sintaxă: constante și variabile

Constante:

- atomi: sansa, 'I am an atom'
- numere: 2, 2.5, -33

Variabile:

- secvențe de litere, numere și _, care încep cu o literă mare sau cu _
- Variabilă specială: _ este o variabilă anonimă
  - două apariții ale simbolului _ sunt variabile diferite
  - este folosită când nu vrem detalii despre variabila respectivă

Exemplu

- `X`
- `Arya`
- `_cersei`

Sintaxă: termeni compuși

Termni compuși:

- au forma `p(t1,...,tn)` unde
  - `p` este un atom,
  - `t1,...,tn` sunt termeni.

Exemplu

- `dislike(cersei,tyrion)`
- `dislike(cersei,X)`

- Un termen compus are
  - un nume (functor): `dislike` în `dislike(cersei,tyrion)`
  - o aritate (numărul de argumente): 2 în `dislike(cersei,tyrion)`

Programe și întrebări

kb1: Un prim exemplu

Un program Prolog definește o bază de cunoștințe.

Exemplu

```
stark(eddard).
stark(jon_snow).
stark(sansa).

lannister(tyrion).
lannister(cersei).

dislike(cersei,tyrion).
```

Predicate

O bază de cunoștințe este o mulțime de predicate prin care definim lumea(universul) programului respectiv.

Exemplu

```
stark(eddard).
stark(jon_snow).
stark(sansa).

lannister(tyrion).
lannister(cersei).

dislike(cersei,tyrion).
```

Acest program conține trei predicate:
`stark/1`, `lannister/1`, `dislike/2`.

Definirea predicatelor

- Predicate cu același nume, dar cu arități diferite, sunt predicate diferite.
- Scriem `foo/n` pentru a indica că un predicat `foo` are aritatea `n`.
- Predicatele pot avea aritatea 0 (nu au argumente); sunt predefinite în limbaj (`true`, `false`).

kb2: Un exemplu cu fapte și reguli

- O **regulă** este o afirmație de forma `Head :- Body.`
- Un ***fapt*** (*fact*) este o regulă fără `Body`.

Exemplu

```
eating(joffrey).
deceased(robert).
dislike(cersei,tyrion).

happy(cersei) :- happy(joffrey).
happy(ser_jamie) :- happy(cersei), deceased(robert).
happy(joffrey) :- dislike(joffrey,sansa).
happy(joffrey) :- eating(joffrey).
```

Reguli

O **regulă** este o afirmație de forma `Head :- Body.` unde

- `Head` este un predicat (termen complex)
- `Body` este o secvență de predicate, separate prin virgulă.

Exemplu

```
happy(ser_jamie) :- happy(cersei), deceased(robert).
```

Interpretarea:

- `:-` se interpretează drept implicație ($\leftarrow$)
- `,` se interpretează drept conjuncție ($\wedge$)

Astfel, din punct de vedere al logicii, putem spune că `happy(ser_jamie)` este echivalent cu `happy(cersei)` $\wedge$ `deceased(robert)`.

Definirea predicatelor

Mai multe reguli care au același Head trebuie gândite că au sau între ele.

Exemplu

```
happy(joffrey) :- dislike(joffrey,sansa).
happy(joffrey) :- eating(joffrey).
```

Dacă `dislike(joffrey,sansa)` este adevărat sau `eating(joffrey)` este adevărat, atunci `happy(joffrey)` este adevărat. Mai multe reguli cu aceeași parte stângă se pot uni folosind `;`.

Exemplu

```
happy(joffrey) :- dislike(joffrey,sansa) ;
eating(joffrey).
```

Astfel, din punct de vedere al logicii, putem spune că `happy(joffrey)` este echivalent cu $\texttt{dislike(joffrey,sansa)} \vee \texttt{eating(joffrey)}$.

Sintaxă: program

Un program în Prolog este o colecție de fapte și reguli.

Faptele și regulile trebuie grupate după atomii folosiți în `Head`.

Exemplu

Corect:

```
stark(eddard).
stark(jon_snow).
stark(sansa).

lannister(tyrion).
lannister(cersei).

dislike(cersei,tyrion).
```

Incorect:

```
stark(eddard).
dislike(cersei,tyrion).
stark(sansa).

lannister(tyrion).
stark(jon_snow).

lannister(cersei).
```

Întrebări, răspunsuri, ținte

- O `întrebare` (*query*) este o secvență de forma

```
?- p1(t1,...,tn),...,pn(t1',...,tn').
```

- Fiind dată o întrebare (deci o țintă), Prolog caută răspunsuri.
  - `true`/ `false` dacă întrebarea nu conține variabile;
  - dacă întrebarea conține variabile, atunci sunt căutate valori care fac toate predicatele din întrebare să fie satisfăcute; dacă nu se găsesc astfel de valori, răspunsul este `false`.

- Predicatele care trebuie satisfăcute pentru a răspunde la o întrebare se numes țitne (goals).

kb2: Exemple de întrebări și răspunsuri

Exemplu

```
eating(joffrey).
deceased(rickard).
dislike(cersei,tyrion).

happy(cersei) :- happy(joffrey).
happy(ser_jamie) :- happy(cersei),
                    deceased(robert).
happy(joffrey) :-
                    dislike(joffrey,sansa).
happy(joffrey) :- eating(joffrey).
```

```
?- happy(joffrey).
true

?- happy(cersei).
true

?- happy(ser_jamie).
false

?- happy(X).
X = cersei ;
X = joffrey.
```

SWISH `https://swish.swi-prolog.org`

În varianta online, puteți adăuga întrebări la finalul programului ca în exemplul de mai jos. Întrebările vor apărea în lista din *Examples* (partea dreaptă).

Exemplu

```
eating(joffrey).
deceased(rickard).
dislike(cersei,tyrion).

happy(cersei) :- happy(joffrey).
happy(ser_jamie) :- happy(cersei), deceased(robert).
happy(joffrey) :- dislike(joffrey,sansa).
happy(joffrey) :- eating(joffrey).

/** <examples>

?- happy(joffrey).
?- happy(cersei).
?- happy(ser_jamie).
?- happy(X).
*/
```

kb3: Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
father(eddard,sansa).
father(eddard,jon_snow).

mother(catelyn,sansa).
mother(wylla,jon_snow).

stark(eddard).
stark(catelyn).

stark(X) :- father(Y,X),
            stark(Y).
```

Pentru orice `X`, `Y`, dacă `father(Y,X)` este adevărat și `stark(Y)` este adevărat, atunci `stark(X)` este adevărat.

Adică, pentru orice `X`, dacă tatăl lui `X` este `stark`, atunci și `X` este `stark`.

kb3: Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
?- stark(jon_snow).
true

?- stark(X).
X = eddard
X = catelyn
X = sansa
X = jon_snow
false

?- stark(X), mother(Y,X), stark(Y).
X = sansa,
Y = catelyn
false
```

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)).
```

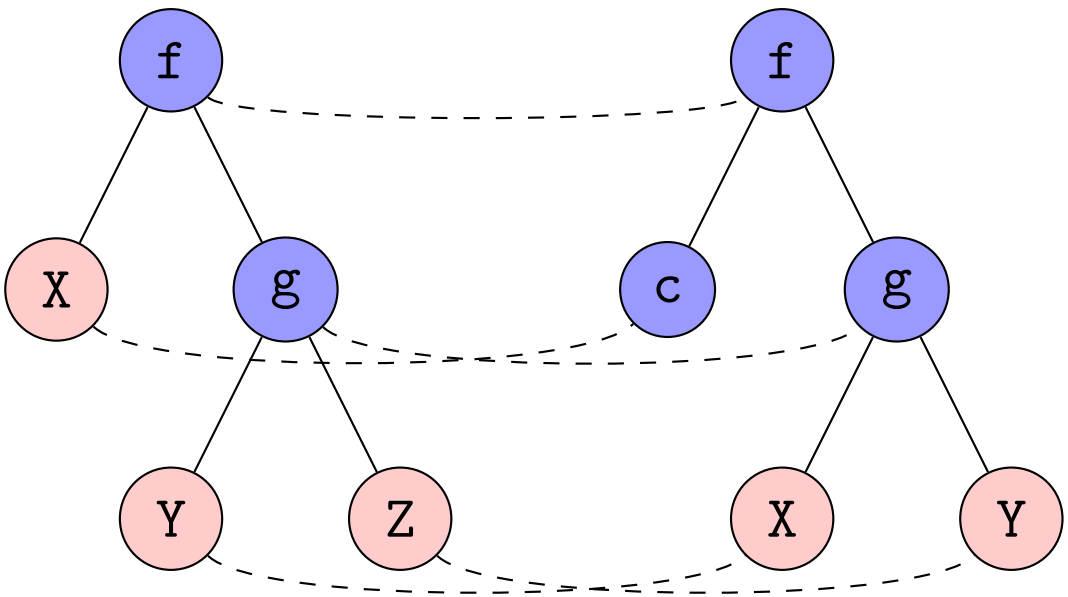
- Cum găsește aceste răspunsuri?

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))

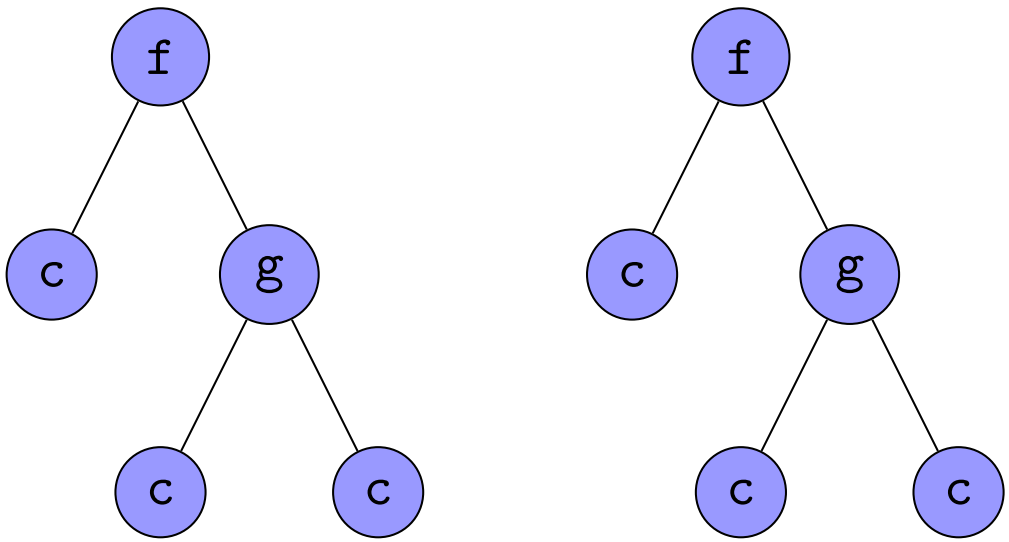


X=c
Y=X
Z=Y

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?

- Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea în care sunt scrise.
- Folosește unificarea pentru a potrivi țintele și clauzele (reguli și fapte).
- Prolog poate da 2 tipuri de răspunsuri:
  - `false` – în cazul în care întrebarea nu este o consecință a programului.
  - `true` sau valori pentru variabilele din întrebare în cazul în care întrebarea este o consecință a programului.
- Poate găsi zero, una sau mai multe soluții.
- Execuția se poate întoarce (*backtracking*).

Diverse

- Un program în Prolog are extensia `.pl`
- Comentarii:

```
% comentează restul liniei

/* comentariu
pe mai multe linii */
```

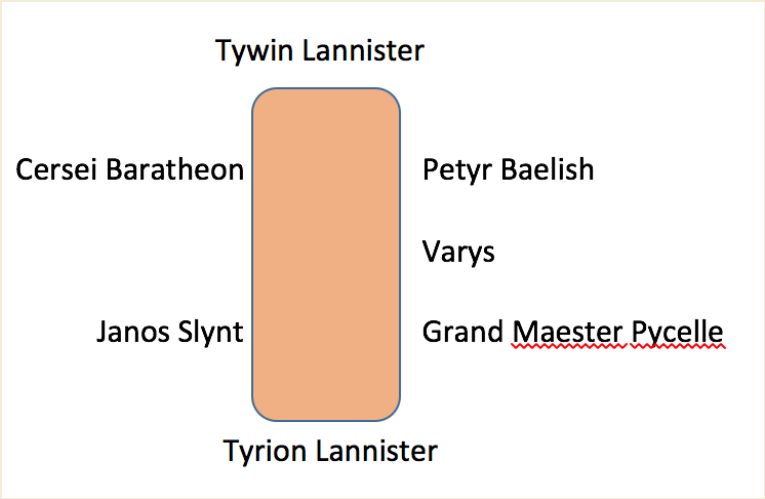
- Nu uitați să puneți . la sfârșitul unui fapt sau al unei reguli.
- un program(o bază de cunoștințe) se încarcă folosind:

```
?- [nume].
?- ['...cale.../nume.pl'].
```

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`.

Exercițiul 1 (cont.)

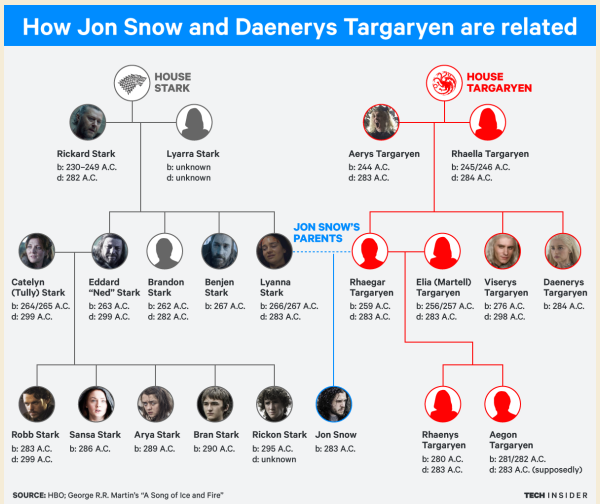
Testați implementarea voastră punând următoarele întrebări:

- 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?

Practică

Exercițiul 2: arbore genealogic

Folosiți predicatele `male/1`, `female/1` și `parent_of/2` pentru a reprezenta următorul arbore genealogic ca bază de cunostințe în Prolog.



`Aici` găsiți poza mărită.
Folosiți baza de cunpștințe "kb4.pl" ca punct de plecare.

Practică

Exercițiul 2 (cont.)

Folosind predicatele `male/1`, `female/1` și `parent_of/2`, adăugați reguli pentru următoarele predicate:

- `father_of(Father, Child)`
- `mother_of(Mother, Child)`
- `grandfather_of(Grandfather, Child)`
- `grandmother_of(Grandmother, Child)`
- `sister_of(Sister,Person)`
- `brother_of(Brother,Person)`
- `aunt_of(Aunt,Person)`
- `uncle_of(Uncle,Person)`

Practică

Exercițiul 2 (cont.)

Verificați predicate definite punând diverse întrebări.

Folosiți acest program în Prolog pentru a afla dacă Daenerys Targaryen este matușa lui Jon Snow.

Practică

Exercițiul 3: recursie/arbore genealogic

Folosind baza de cunoștințe anterioară, scrieți un predicat `ancestor_of (Ancestor, Person)` care să verifice dacă al primul argument este strămoș al celui de-al doilea.

Exemplu

```
?- ancestor_of(rickardStark,jonSnow).
true
```

Negația în Prolog

Negația

În Prolog există predicatul predefinit `not` cu următoarea semnificație: `not(goal)` este `true` dacă `goal` nu poate fi demonstrat în baza de date curentă.

Atenție: `not` nu este o negație logică, ci exprimă imposibilitatea de a face demonstrația (sau instanțierea) conform cunoștințelor din bază ("closed world assumption"). Pentru a marca această distincție, în variantele noi ale limbajului, în loc de `not` se poate folosi operatorul `\+`.

Exemplu

```
not_parent(X,Y) :- not(parent_of(X,Y)). % sau
not_parent(X,Y) :- \+ parent_of(X,Y).
```

Practică

Exercițiul 4: negația

Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

```
?- not_parent(rickardStark,viserysTargaryen).
?- not_parent(X,jonSnow).
?- not_parent(X,Y).
```

Ce observați? Încercați să analizați răspunsurile primite.

Practică

Exercițiul 4: negația

Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

```
?- not_parent(rickardStark,viserysTargaryen).
?- not_parent(X,jonSnow).
?- not_parent(X,Y).
```

Ce observați? Încercați să analizați răspunsurile primite.

- Corectați `not_parent` astfel încât să dea răspunsul corect la toate întrebările de mai sus.
- Scrieți un predicat `ancestor_not_parent(X,Y)` care să verifice dacă `X` este strămoș dar nu este părinte al lui `Y`.

Practică

În GOT apare următoarea ghicitoare:

`https://goodriddlesnow.com/posts/view/riddle-from-game-of-thrones`

Varys - "Power is a curious thing, my lord. Are you fond of riddles?"
Tyrion - "Why? Am I about to hear one?"
Varys - "Three great men sit in a room, a king, a priest and the rich man. Between them stands a common sellsword. Each great man bids the sellsword kill the other two. Who lives? Who dies?"
Tyrion - "Depends on the sellsword."

Vom descrie această problemă în Prolog, făcând urmă toarele presupuneri:
- dacă mercenarul este un om religios, atunci nu va ucide preotul,
- dacă mercenarul este un om care respectă autoritatea, atunci nu va ucide regele,
- dacă mercenarul este un om care dorește bani, atunci nu va ucide omul bogat.

Practică

Exercițiul 5: ghicitoare

În fișierul kb5.pl este începută o modalitate de a descrie ghicitoarea de mai sus. Completați baza de cunoștințe și scrieți un predicat

`is_killed(C,X,Y)`

care verifică faptul că `X` și `Y` sunt uciși de alegerea `C` a mercenarului.

```
?- is_killed(god, X,Y).
X = king,
Y = richMan
```

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
```

Practică

Exercițiul 6: 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
```

Recursivitate

Bază de conștințe

În laboratorul trecut am folosit următoarea bază de cunoștițe:

```
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

Am definit un predicat `ancestor_of(X,Y)` care este adevărat dacă X este un strămoș al lui Y.

Definiția recursivă a predicatului `ancestor_of(X,Y)` :

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

Practică

Exercițiul 7: 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 7 (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.

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
```

Practică

Exercițiul 8: 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, '* ').
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Liste

Liste

- Listele în Prolog sunt un tip special de date (termeni speciali).
- Listele se scriu între paranteze drepte, cu elementele despărțite prin virgulă.
- `[]` este lista vidă.

Exemplu

- `[elephant, horse, donkey, dog]`
- `[elephant, [], X, parent(X, tom), [a, b, c], f(22)]`

Head & Tail

- Primul element al unei liste se numește *head*, iar restul listei *tail*.
- Evident, o listă vidă nu are un prim element.
- În Prolog există o notație utilă pentru liste cu separatorul |, evidențiind primul element și restul listei.

Exemplu

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
```

Cu această notație putem să returnăm ușor, de exemplu, al doilea element dintr-o listă.

```
?- [quod, licet, jovi, non, licet, bovi] = [_, X | _].
X = licet
```

Lucrul cu liste

Exemplu (`elements_of/2`)

- un predicat care verifică dacă o listă conține un anumit termen
- `element_of(X,Y)` trebuie să fie adevărat dacă `X` este un element al lui `Y`.

```
/* Dacă primult element al listei este termenul
pe care îl căutăm, atunci am terminat. */

element_of(X,[X|_]).

% Altfel, verificăm dacă termenul se află în restul
listei.

element_of(X,[_|Tail]) :- element_of(X,Tail).

?- element_of(a,[a,b,c]).
?- element_of(X,[a,b,c]).
```

Lucrul cu liste

Exemplu (`concat_lists/3`)

- un predicat care este poate fi folosit pentru a concatena două liste
- al treilea argument este concatenarea listelor date ca prime două argumente

```
concat_lists([], List, List).
concat_lists([Elem | List1], List2, [Elem | List3]) :-
      concat_lists(List1, List2, List3).


?- concat_lists([1, 2, 3], [d, e, f, g], X).
?- concat_lists(X, Y, [a, b, c, d]).
```

Lucrul cu liste

În Prolog există niște predicate predefinite pentru lucrul cu liste. De exemplu:

- `length/2`: al doilea argument întoarce lungimea listei date ca prin argument
- `member/2`: este adevărat dacă primul argument se află în lista dată ca al doilea argument
- `append/3`: identic cu predicatul anterior `concat_lists/3`
- `last/2`: este adevărat dacă al doilea argument este identic cu ultimul element al listei date ca prim argument
- `reverse/2`: lista din al doilea argument este lista data ca prim elemenent în oglindă.

Practică

Exercițiul 9

A) Definiți un predicat `all_a/1` care primește ca argument o listă și care verifică dacă argumentul său este format doar din a-uri.

```
?- all_a([a,a,a,a]).
?- all_a([a,a,A,a]).
```

B) Scrieti un predicat `trans_a_b/2` care "traduce" o listă de a-uri într-o listă de b-uri. `trans_a_b(X,Y)` trebuie să fie adevărat dacă "intrarea" `X` este o listă de a-uri și "ieșirea" `Y` este o listă de b-uri, iar cele două liste au lungimi egale.

```
?- trans_a_b([a,a,a],L).
?- trans_a_b([a,a,a],[b]).
?- trans_a_b(L,[b,b]).
```

Practică

Exercițiul 10: Operații cu vectori

A) Scrieți un predicat `scalarMult/3` al cărui prim argument este un întreg, al doilea argument este o listă de întregi, iar al treilea argument este rezultatul înmulțirii cu scalari al celui de-al doilea argument cu primul.

De exemplu, la intrebarea

```
?-scalarMult(3,[2,7,4],Result).
```

ar trebui să obțineți `Result = [6,21,12]`.

Practică

Exercițiul 10 (cont.)

B) Scrieți un predicat `dot/3` al cărui prim argument este o listă de întregi, al doilea argument este o listă de întregi de lungimea primeia, iar al treilea argument este produsul scalar dintre primele două argumente.

De exemplu, la întrebarea

```
?-dot([2,5,6],[3,4,1],Result).
```

ar trebui să obțineți `Result = 32`.

Practică

Exercițiul 10(cont.)

C) Scrieți un predicat `max/2` care caută elementul maxim într-o listă de numere naturale.

De exemplu, la întrebarea

```
?-max([4,2,6,8,1],Result).
```

ar trebui să obțineți `Result = 8.`

Practică

Exercițiul 11

Definiți un predicat `palindrome/1` care este adevărat dacă lista primită ca argument este palindrom (lista citită de la stânga la dreapta este identică cu lista citită de la dreapta la stânga).

De exemplu, la întrebarea

```
?-palindrome([r,e,d,i,v,i,d,e,r]).
```

ar trebui să obțineți `true`.

Nu folosiți predicatul predefinit `reverse`, ci propria implementare a acestui predicat.

Practică

Exercițiul 12

Definiți un predicat `remove_duplicates/2` care șterge toate duplicatele din lista dată ca prim argument și întoarce rezultatul în al doilea argument.

De exemplu, la întrebarea

```
?- remove_duplicates([a, b, a, c, d, d], List).
```

ar trebui să obțineți `List = [b, a, c, d]`.

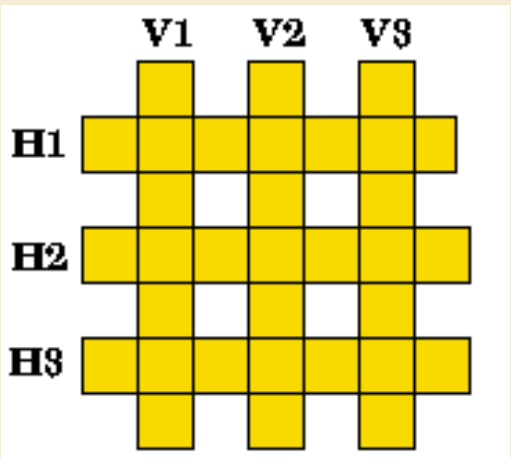
Practică

Exercițiul 13: cunvinte încrucișate

Șase cuvinte din engleză

abalone, abandon, anagram, connect, elegant, enhance

trebuie aranjate într-un puzzle de cuvinte încrucișate ca în figură.



Fișierul `words.pl` este o bază de cunoștințe ce conține aceste cuvinte.

Practică

Exercițiul 13 (cont.)

Definiți un predicat `crosswd/6` care calculează toate variantele în care puteți completa grila. Primele trei argumente trebuie să fie cuvintele pe verticală, de la stânga la dreapta, (V1,V2,V3), iar următoarele trei argumente trebuie să fie cuvintele pe orizontală, de sus în jos (H1,H2,H3).

Hint: Specificați că V1, V2, V3, H1, H2, H3 sunt cuvinte care au anumite litere comune. Unde este cazul, folosiți variabile anonime.

Practică

Exercițiul 14: baza de date

În acest exercițiu vom demonstra cum se poate implementa o bază de date simplă în Prolog.

Folosiți în programul vostru următoarea bază de cunoștințe:

```
born(jan, date(20,3,1977)).
born(jeroen, date(2,2,1992)).
born(joris, date(17,3,1995)).
born(jelle, date(1,1,2004)).
born(joan, date(24,12,0)).
born(joop, date(30,4,1989)).
born(jannecke, date(17,3,1993)).
born(jaap, date(16,11,1995)).
```

Reprezentăm datele calendaristice ca termeni de forma `date(Day,Month,Year)`.

U. Endriss, Lecture Notes. An Introduction to Prolog Programming, ILLC, Amsterdam, 2018.

Practică

Exercițiul 14 (cont.)

a) Scrieți un predicat `year/2` care găsește toate persoanele născute într-un anumit an.

Exemplu:

```
?- year(1995, Person).
Person = joris
Person = jaap
```

Hint: Folosiți variabile anonime.

Exercițiul 14 (cont.)

b) Scrieți un predicat `before/2` care primește două date calendaristice și care este adevărat dacă prima expresie reprezintă o dată calendaristică înaintea datei reprezentate de a doua expresie (puteți presupune că datele sunt corecte, e.g., nu puteți primi 31 Aprilie).

Exemplu:

```
?- before(date(31,1,1990), date(7,7,1990)).
true
```

Exercițiul 14 (cont.)

c) Scrieți un predicat `older/2` care este adevărat dacă persoana dată ca prim argument este mai în vârstă (strict) decât persoana dată ca al doilea argument.

Exemplu:

```
?- older(jannecke,X).
X = joris
X = jelle
X = jaap
```

Practică

Exercițiul 15: Cel mai lung cuvânt

Acest exemplu este din
Ulle Endriss, *Lecture Notes – An Introduction to Prolog Programming.*

Countdown este un joc de televiziune popular în Marea Britanie în care jucătorii trebuie să găsească un cuvânt cât mai lung cu literele dintr-o mulțime dată de nouă litere.

Să încercăm să rezolvăm acest joc cu Prolog!

Practică

Exercițiul 15: Cel mai lung cuvânt

Concret, vom încerca să găsim o soluție optimă pentru următorul joc:

Primind o listă cu litere din alfabet (nu neapărat unice),
trebuie să construim cel mai lung cuvânt format din literele date
(pot rămâne litere nefolosite).

Vom rezolva jocul pentru cuvinte din limba engleză.

Scorul obținut este lungimea cuvântului găsit.

Exercițiul 15 (cont.)

Scopul final este de a construi un predicat în Prolog `topsolution/2`:

dându-se o listă de litere în primul său argument, trebuie să returneze în al doilea argument o soluție cât mai bună, adică un cuvânt din limba engleză de lungime maximă care poate fi format cu literele din primul argument.

```
?- topsolution([r,d,i,o,m,t,a,p,v],Word).
Word = dioptra
```

Practică

Exercițiul 15(cont.)

Începeți prin a descărca fișierul `words.pl` în același director cu fișierul programului vostru.

Acest fișier conține o listă cu peste 350.000 de cuvinte din limba engleză, de la `a` la `zyzzyva`, sub formă de fapte.

Includeți linia `:- include('words.pl').` în programul vostru pentru a putea folosi aceste fapte.

Exercițiul 15 (cont.)

Predicatul predefinit în Prolog `atom_chars(Atom,CharList)` descompune un atom într-o listă de caractere.

Folosiți acest predicat pentru a defini un predicat `word_letters/2` care transformă un cuvânt (i.e, un atom în Prolog) într-o listă de litere.
De exemplu:

```
?- word_letters(hello,X).
X = [h,e,l,l,o]
```

Ca o paranteza, observați că puteți folosi acest predicat pentru a găsi cuvinte în engleză de 45 de litere:

```
?- word(Word), word_letters(Word,Letters),
   length(Letters,45).
```

Practică

Exercițiul 15 (cont.)

Mai departe, scrieți un predicat `cover/2` care, primind două liste, verifică dacă a doua listă "acoperă" prima listă (i.e., verifică dacă fiecare element care apare de k ori în prima listă apare de cel puțin k ori în a doua listă).

De exemplu

```
?- cover([a,e,i,o], [m,o,n,k,e,y,b,r,a,i,n]).
true

?- cover([e,e,l], [h,e,l,l,o]).
false
```

Practică

Exercițiul 15 (cont.)

Scrieți un predicat `solution/3` care primind o listă de litere ca prim argument și un scor dorit ca al treilea argument, returnează prin al doilea argument un cuvânt cu lungimea egală cu scorul dorit, "acoperit" de lista respectivă de litere.

De exemplu

```
?- solution([g,i,g,c,n,o,a,s,t], Word, 3).
Word = act
```

Practică

Exercițiul 15 (cont.)

Acum implementați predicatul `topsolution/2`.

Testați, de exemplu, predicatul definit pe mulțimea de litere:

```
[y,c,a,l,b,e,o,s,x]
```

Aceasta este una listele de litere folosite în ediția de *Countdown* din 18 Decembrie 2002 din Marea Britanie în care Julian Fell a obținut cel mai mare scor din istoria concursului. Pentru lista de mai sus, el a găsit cuvântul *cables*, câștigând astfel 6 puncte.

Poate programul vostru să bată acest scor?

Mai multe exerciții

P-99: Ninety-Nine Prolog Problems