

LAB 1

Prolog (Sicstus, SWI) is a symbolic programming language, suitable for solving problems that involve objects and relations between objects. A Prolog program is a knowledge base that can be questioned.

Consider the following Prolog program (a text file with the extension **pl**):

```
parent(ion,maria).
parent(ana,maria).
parent(ana,dan).
parent(maria,elena).
parent(maria,radu).
parent(elena,nicu).
parent(radu,george).
parent(radu,dragos).
```

A Prolog program must first be consulted.

```
consult('c:\\prolog\\pro.pl').
```

In the query window, enter the following questions:

```
parent(ana,maria).
parent(ion,radu).
parent(X,maria).
parent(X,Y).
```

Observations:

- 1) alphanumeric constants (atoms) start with a lower-case letter;
- 2) variables start with an upper-case letter or an underscore character;
- 3) the answers are obtained in the order in which the information appears in the knowledge base;
- 4) if the question contains variables, Prolog finds the particular objects (instances) for which the answer is true.

To find out who Radu's grandfather is, we can ask

```
parent(X,Y), parent(Y,radu).
```

Prolog clauses

A clause has the general form $R:-C_1, C_2, \dots, C_N$, with the meaning:

$\forall X_1, X_2, \dots, X_k$ variables in the clause, R is true if C_1, C_2, \dots, C_N .

Example

```
child(X, Y):-parent(Y, X).
```

i.e. $\forall X, Y$ if Y is parent of X then X is child of Y .

Starting from the parent relation, define the following relations:

brother(X, Y) (X and Y are brothers)

grandparent(X, Y) (X is grandparent of Y).

To define a predecessor type relation, we need a recursive rule:

```
pred(X,Y):-parent(X,Y).  
pred(X,Z):-parent(X,Y), pred(Y,Z).
```

Obs. The scope of a variable is the clause in which the respective variable appears.

Write a different version of the predecessor relation.

In Prolog, we can use **structures** – i.e. many components combined in one object. A structure can be generally described as:

```
functor(arg1, arg2, ..., argN).
```

Obs.

- 1) A functor is an atom (it starts with a lower-case letter);
- 2) Arguments can be structures;
- 3) Each structure is well defined by the main functor and the arity (number of arguments).

Example

```
date(1,octomber,2019).  
segment(point(1,2),point(6,7)).  
point(1,2) and point(1,2,3) are different structures
```

Matching

It is the only operation allowed between Prolog terms.

Def It is called substitution the set

$$\Theta = \{(X_i, t_i)_{i=1..n} \mid X_i \text{ variables, } t_i \text{ Prolog terms and } X_i \neq X_j \text{ if } i \neq j\}$$

If T is a Prolog term, we denote by TQ the term resulted by simultaneous replacement in T of all appearances of X_i cu t_i , where $(X_i, t_i) \in \Theta$.

Def A term is completely instantiated if it does not contain any variable.

Def T1 matches T2 if $\exists \Theta$ so that $T1\Theta = T2\Theta$. Θ is called unifier.

Examples

```
1) T1=f(X,a)  
   T2=f(b,Y)  
    $\Theta = \{(X,b), (Y,a)\}$   
2) T1=X  
   T2=f(g(Y,Z),b)  
    $\Theta_1 = \{(X, f(g(Y,Z),b))\}$   
    $\Theta_2 = \{(Y,a), (Z,c), (X,f(g(a,c),b))\}$ 
```

Θ_1 is more general than Θ_2 .

The predicate = tests if two Prolog terms match. If yes, the most general substitution is returned.

Test the following question

```
X=f(X).
```

Obs.

- 1) two atoms match iff they are identical;
- 2) a variable matches anything;
- 3) two structures match if they have the same main functor, the same arity and the corresponding arguments match.

Arithmetic in Prolog

`=` = equality of terms

`1+2=` `2+1` no

`=:` = equality of the numerical values of two arithmetic expressions

`1+2=:` `2+1` yes

`\=` = different terms

`X is Y` variable X is instantiated with the value of Y.

`X is 3+2.`

Lists

A list is a sequence of any number of comma-separated items:

`[1,2,3,4,5,a,b,c]`

The empty list is denoted by `[]`.

A non-empty list can be viewed as consisting of head and tail `[A|B]`. The head A is one item, the tail B is a list.

We can indicate several elements at the beginning of the list:

`[A,B,C,...|T]`.

Example

`[1,2,3]=[1|[2,3]]=[1,2|[3]]=[1,2,3|[]]`.

We can collect elements that satisfy a certain property with the help of 3 predefined predicates:

1) `bagof(X,P,L)` collects in the list L the items X that satisfy P. If there is no such element, the answer is no.

2) `setof(X,P,L)` similar as `bagof`, but it eliminates duplicates and the resulting list is sorted

3) `findall(X,P,L)` if there is no element satisfying P the result is yes and `L=∅`. It ignores variables that appear in P and do not appear in X.

Example:

Assuming we have relations `boy(name, age)`, we calculate the sum of the ages of all the boys from the knowledge base.

`ageboys(L):-findall(Age,boy(Name,Age),L).`

`sum(S):-ageboys(L),sum(L,S).`

`sum([],0).`

`sum([H|T],S):-sum(T,S1), S is S1+H.`

The efficiency of the Prolog programs

The function

$$f(x) = \begin{cases} 0, & x \leq 3 \\ 2, & x \in (3,6] \\ 4, & x > 6 \end{cases}$$

can be implemented in Prolog by the following clauses:

`f(X,0):-X=<3.`

`f(X,2):-3<X, X=<6.`

`f(X,4):-6<X.`

If we ask

$f(1,Y), 2 < Y.$

Prolog searches the solution on all three clauses, although logically it should have stopped after the first clause. We can prevent backtracking with the help of the predicate ! (cut). The program becomes as follows:

$f(X,0):-X \leq 3, !.$

$f(X,2):-3 < X, X \leq 6, !.$

$f(X,4):-6 < X.$

If $X \leq 3$ is true, the search stops (it cuts backtracking).

In the clause 2, the test $3 < X$ it is redundant because it gets here only if the cut in the first clause has not been reached.

So, the program may be written as:

$f(X,0):-X \leq 3, !.$

$f(X,2):-X \leq 6, !.$

$f(X,4).$

Exercises

- 1) Write the max predicate that calculates the maximum between 2 values.
- 2) Write the *member* and *concat* predicates.
- 3) Calculate the alternate sum of the elements of a list.
- 4) Eliminate an element from a list (one/all the occurrences of that element).
- 5) Reverse a list; generate all the permutations of the elements of a list.
- 6) Find the number of occurrences of an element in a list.
- 7) Insert an element on a certain position in a list.
- 8) Merge two ascending ordered lists.