### Running and Testing a Program

To check, that your system is set up properly, you should try to type in the following text in the window:

GOAL write("Hello world"),nl.

This is a GOAL in the Prolog and is enough to be a program that can be executed. To execute the GOAL, you should activate the menu item **Project | Test Goal**, or just press the accelerator key **Ctrl+G**. If your system is installed properly, your screen will look like the following:



The result of the execution will come up in a separate window, which you must close before you can test another GOAL.

Prolog Exercises

**Program 1:**

PREDICATES

nondeterm likes(symbol,symbol)

CLAUSES

likes(ellen,tennis).

likes(john,football).

likes(tom,baseball).

likes(eric,swimming).

likes(mark,tennis).

likes(bill,Activity):-

likes(tom, Activity).

GOAL

likes(bill, baseball).

The system replies in the Dialog window

yes

It has combined the rule

likes(bill, Activity):- likes(tom, Activity).

with the fact

likes(tom, baseball).

to decide that

likes(bill, baseball).

***Exercise Try the query:***

likes(bill, tennis).

The system replies

no

Visual Prolog replies no to the latest query ("Does Bill like tennis?") because

i) There is no fact that says Bill likes tennis.

ii) Bill's relationship with tennis can't be inferred using the given rule and the available facts.

**Program 2**

Program designed to find solutions to a car-buying problem:

/\* Program ch01e02.pro \*/

PREDICATES

nondeterm can\_buy(symbol, symbol)

nondeterm person(symbol)

nondeterm car(symbol)

likes(symbol, symbol)

for\_sale(symbol)

CLAUSES

can\_buy(X,Y):-

person(X),

car(Y),

likes(X,Y),

for\_sale(Y).

person(kelly).

person(judy).

person(ellen).

person(mark).

car(lemon).

car(hot\_rod).

likes(kelly, hot\_rod).

likes(judy, pizza).

likes(ellen, tennis).

likes(mark, tennis).

for\_sale(pizza).

for\_sale(lemon).

for\_sale(hot\_rod).

What can Judy and Kelly buy? Who can buy the hot rod? You can try the following goals:

can\_buy(Who, What).

can\_buy(judy, What).

can\_buy(kelly, What).

can\_buy(Who, hot\_rod).

**Experiment**

Add other facts and maybe even a rule or two to this Prolog program. Test the new program with queries that you make up. Does Prolog respond in a way you would expect it to?

**Program 3**

Take a look at the following example, which uses program 3 to demonstrate how and when variables get their values.

/\* Program ch01e03.pro \*/

PREDICATES

nondeterm likes(symbol,symbol)

CLAUSES

likes(ellen,reading).

likes(john,computers).

likes(john,badminton).

likes(leonard,badminton).

likes(eric,swimming).

likes(eric,reading).

Consider this query: Is there a person who likes both reading and swimming?

likes(Person, reading), likes(Person, swimming).

Prolog will solve the two parts of this query by searching the program's clauses from top to bottom. In the first part of the query

likes(Person, reading)

the variable *Person* is free; its value is unknown before Prolog attempts to find a solution. On the other hand, the second argument, *reading*, is known. Prolog searches for a fact that matches the first part of the query. The first fact in the program

likes(ellen, reading)

is a match (*reading* in the fact matches *reading* in the query), so Prolog binds the free variable *Person* to the value *ellen*, the relevant value in the fact. At the same time, Prolog places a pointer in the list of facts indicating how far down the search procedure has reached.

Next, in order for the query to be fully satisfied (find a person who likes both reading and swimming), the second part must also be fulfilled. Since *Person* is now bound to *ellen*, Prolog must search for the fact

likes(ellen, swimming)

Prolog searches for this fact from the beginning of the program, but no match occurs (because there is no such fact in the program). The second part of the query is not true when *Person* is *ellen*.

Prolog now "unbinds" *Person* and attempts another solution of the first part of the query with *Person* once again a free variable. The search for another fact that fulfills the first part of the query starts from the pointer in the list of facts. (This returning to the place last marked is known as *backtracking*, which we'll cover in chapter **3**.)

Prolog looks for the next person who likes reading and finds the fact likes(eric, reading). *Person* is now bound to *eric*, and Prolog tries once again to satisfy the second part of the query, this time by looking in the program for the fact

likes(eric, swimming)

This time it finds a match (the last clause in the program), and the query is fully satisfied. Prolog returns

Person=eric

1 Solution