Division of Computing Science and Mathematics

CSCU9Y4 Prolog Practical 1

This is a modified version of Chapter 1 of *The Professional Programmer's Guide to Prolog* by A G Hamilton

Getting Started with Prolog

Prolog is different. You will take some time to get used to it, as it requires its own distinctive way of thinking. To begin with (as in this practical) it is not too hard. But many people then find it difficult to make the leap necessary to carry out more substantial tasks. When you are familiar with it, however, it should become an interesting and rewarding language with which to work. So persevere!

Prolog is more than a language. It is a language with an **interpreter**. In fact any computer language is useless without a **compiler** or an interpreter which give the various instructions in the language actual meanings or effects. The existence of such semantics in the case of Prolog is very apparent to the user for two reasons -- first because the user interacts directly (step by step) with the interpreter, and second because the language is designed to work in an unusual way.

Starting Prolog

Make a *copy* of the prolog folder located in the CSCU9Y4 Group folder and place it a suitable place in your home directory.

Enter your new prolog folder and double-click on the icon SWI.plg. This will take you into Prolog. Try it.

Now let us suppose that you are at a computer and have started up Prolog as above. You see a window, which is your means of communicating with the Prolog interpreter. In this window, in response to the '`?-`' prompt, you can enter **goals**, and send them to the interpreter.

A goal is not a command, nor is it an instruction. As the name suggests, it is something that the interpreter will set out to 'achieve'. There is a standard form for goals, but the effects caused by trying to achieve them can be very different.

(Incidentally, to exit from Prolog either: click on the cross in the top right corner of the window, enter `CTRL-D`, or give the goal `?- halt.` Another handy hint is that the ↑ arrow key will recall the last typed line. So if you made a typing error, use ↑ and then ← arrow to correct.).

Try the following seven goals (separately, remembering to finish each goal with a full-stop, before hitting the return key). You will see that after each one the interpreter will respond with a message, and may wait for you to press `RETURN` again before the `?-` prompt appears again ready for the next goal. **Remember that the interpreter is not ready to accept a new goal unless the `?-` prompt is showing on a new line.** If you make a mistake, use the arrow keys to bring back the previous line and edit it.

```
?- write('Hello user!').

?- write(1987).

?- 2 < 3.

?- 3 < 2.

?- not(3 < 2).

?- 5 = 5.

?- 5 \= 5.

?- 5 = X.
```

The last is the only one that requires comment. Here `X` is a Prolog **variable** (as is any upper case letter). This goal can be achieved by giving the value `5` to `X`, so this is what the interpreter does. The message which appears on the screen will show this by including `X = 5`. After this the interpreter waits for you to press `RETURN` before allowing you to proceed. This is our first experience of variables in Prolog, so it is an appropriate place to give a warning. Variables behave in an unusual (and different) way in Prolog. In particular there are *no global variables* (all variables are in effect parameters of one kind or another), and there is *no such thing as assignment*.

A goal that is achieved by the interpreter is said to '**succeed**'. A goal that is not achieved is said to '**fail**'. Examples of both occur amongst the goals that you have just tried. Note the different forms of response that the interpreter gives. Now let us move on to some goals that involve computation. The first two might be described as 'false starts'.

```
?- 2 + 3.
```

This doesn't get us anywhere. It is an inappropriate form for a goal. A goal is something that may turn out to be true or false (although there are exceptions, like goals involving '`write`', as above). Now try

```
?- X = 2 + 3.
```

Surprisingly perhaps, this doesn't get us very far either. This goal is achieved by the interpreter giving `X` the value `2 + 3`. But `2 + 3` is a formal expression and as such (according to Prolog) is different from `5`.

```
?- X is 2 + 3.
```

At last, this is what we are looking for. `X` is now given the value `5`.
The reserved word '`is`' causes evaluation of arithmetic expressions.

```
?- Y is (4 + 2) * 3.
```

```
?- R is 4 + 2 * 3.
```

Next we can explore ways in which more complicated goals can be formed.

```
?- write('Hello'), write(' user'), write('!').
```

```
?- write('Hello'), nl, write('How are you?').
```

Note the effect of the goal '`nl`'. It causes the next '`write`' to output on the next line.

```
?- X is 2 + 3, write(X).
```

```
?- P = 117, V is P * 2, write('Value: '), write(V).
```

So this is the way we write sequences of goals. Just separate them by commas, and the interpreter takes each in turn. Note the way in which variables are used as parameters. In the first goal, `X` is first given a value, and then the value is used by '`write`'. In the second, the value `P * 2` is computed, then given to `V` and finally used by '`write`'.

There is also another way to combine goals. Goals can be given as alternatives, in effect combined by **or**. Prolog uses the semicolon for this (where in the previous two goals the comma in effect means **and**).

```
?- write('Hello'); write(' user!').
```

```
?- X is 2 + 3; write(X).
```

In each of these cases the compound goal is achieved as soon as the first part is achieved, so the second alternative is ignored.

```
?- 3 < 2; write('Wrong!').
```

This succeeds -- the first alternative fails, but then the second alternative succeeds. A goal involving '`write`' always succeeds.

Next, try these:

```
?- old(methuselah).
```

```
?- father(adam,cain).
```

```
?- greaterthan(3,2).
```

```
?- positive(X * X + 1).
```

All of these receive the 'thumbs down' from the interpreter. It cannot achieve them. The reason is that it does not have any information about the terms used, namely '`old`', '`1`', '`greaterthan`', '`positive`'. These words make sense to us, and we can imagine easily what are the questions that are being asked when we give these goals. But to the Prolog interpreter they have no meaning at all. (We shall see later how we can arrange for the interpreter to 'know' about such things). It does know about '`write`', '`is`', +, *, =, <, and various other basic 'built-in' facilities, and so was able to interpret goals which involved these sensibly.

Using a database
We are going to consult the file `database1.plg`. The 'plg' extension tells us this is a Prolog file (sometimes files might have the extension .pl, and this causes problems because it's the same extension as is used for Perl files so be careful). Firstly examine the contents of file `database1.plg` by opening it up using textpad (wordpad/notepad whatever your favourite text editor is).

Now back in Prolog, issue the following command:

```
?- consult(database1).
```

You may get some warning messages appearing whenever you consult a file. Ignore these for now. We are telling Prolog to read into its database all facts and rules contained in the file database1. To view this information try the following:

```
?- listing.
```

The database now contains three 'facts': two about '`old`' and one about '`young`'. The words '`old`' and '`young`' here are **predicates**, which means that they are used to represent properties. For example, the term `old(jane)` is to be regarded as the assertion 'jane is old'. The words '`john`', '`jane`' and '`bill`' are names, and as such they are constants, not variables. For the time being we shall use only single upper-case letters for variables. Try the following (separate) goals:

```
?- old(jane).

?- young(jane).

?- old(X).
```

Modify the file database1.plg in the text editor and add the following facts and rules.

```
old(john).
old(anne).
young(bill).
wiser(X,Y) :- old(X), young(Y).
```

Remember to include the full stops! Save the file, and in the Prolog window, again issue the command:

```
?- consult(database1).
```

A shorthand for the same command is `[database1].`

Check your updated database using

```
?- listing.
```

Note that you can also get a listing of a specific predicate using

```
?- listing(old).
```

See what happens when you try the goals

```
?- wiser(jane,bill).

?- wiser(jane,john).

?- wiser(jane,emily).

?- wiser(john,Z).
```

Besides the facts about '`old`' and '`young`', the database now contains a general **rule**, which specifies certain circumstances in which two objects are to satisfy the predicate '`wiser`'. (Notice that a predicate can express a property of any number of objects, not just of a single object, as '`old`' and '`young`' did. The term `wiser(jane,bill)` is intended to mean 'jane is wiser than bill'.) The rule allows the Prolog interpreter to infer new facts from the given ones.

The Trace command

Trace is a useful mechanism for seeing what is going on behind the scenes when Prolog is trying to achieve a goal. To switch trace on we use the command `trace`. To switch it off we use the command `notrace.` If you enter debug mode (sometimes done whilst tracing) to turn off debug issue the command `nodebug.`

Try the following commands:

```
?- trace.
?- wiser(jane,emily).
```

You should see all the steps performed in achieving this goal. Try issuing some of the other goals we've looked at using trace (for example, try `wiser(jane,john)` to see what happens when a goal fails, and `wiser(john,Z)` to see how Prolog uses internal variables) and make sure you understand what is happening.

More Examples

The examples so far illustrate the basic processes that the interpreter uses. But there is rather more to it, as we shall see later. Try the goal

```
?- wiser(X,Y).
```

The interpreter will achieve this in the first way it can, finding values for `X` and `Y` in the process. It will find the rule which says

```
wiser(X,Y) :- old(X), young(Y).
```

It will respond by showing values for `X` and `Y`. For example

```
 X = jane
 Y = bill
```

Let's think about how it does this. It tries to achieve `old(X)` in the first way it can, i.e. with `X = jane`. Then it proceeds to achieve `young(Y)` in the first way it can, i.e. with `Y = bill`. Thus these are the values chosen to achieve the goal `?- wiser(X,Y).`

Remember that `:-` means 'if', and the comma means 'and'.

Let's take this example a bit further. Extend the database again (edit the file `database1.plg`) so that it contains all of the following:

```
dark_haired(john).
grey_haired(jane).
dark_haired(bill).
wiser(X,Y) :- grey_haired(X), dark_haired(Y).
```

Save the file, and in the Prolog window, again issue the command:

```
?- consult(database1).
```

Next try the goal

```
?- wiser(jane,john).
```

Where the answer previously had been '`No`', the extra information now means that this goal can be achieved. Note that there is no difficulty associated with having two (or more) rules associated with the predicate '`wiser`'. They complement each other.

One More Example
1. A set of facts involving father, mother, male and female is given in the `archers.plg` file. These will be used again next week, and also in the tutorials.

```
?- consult(archers).
```

What are the results of giving the following goals? (Note the answers here – you will need to tell the tutor the answers to get the checkpoint.)

```
        ?- mother(jill, david).
        ?- mother(jill, X).
        ?- female(X).
        ?- father(brian, X).
        ?- father(brian, X), male(X).
        ?- father(X,phil), father(X,Y), female(Y).
```

What fact does the last goal capture about the relationship between phil and christine?

If we are dealing with father facts and mother facts, it might be useful to have a predicate parent which embodies the ‘parent’ relationship, defined by the following rule:

```
parent(X,Y) :- father(X,Y); mother(X,Y).
```

Why did we use a “;” here instead of a “,”?

`parent` is already in the archers database.

2. **Trace** through what happens when you issue the following goals to Prolog:

```
   ?- parent(dan, christine).
   ?- parent(pat, tommy).
   ?- parent(jenny,X).
   ?- parent(X,Y).
```

Again, note your answers. Be sure to follow every step. Do you understand what the Prolog interpreter is doing?

3. It’s time you wrote a predicate of your own. Create a predicate

```
brother(B,S)
```

which determines whether `B` is the brother of `S`. (`B` stands for `Brother`, `S` for `Sibling`.) You’ll need the `parent` predicate defined above.
Test your new predicate with the following goals:

```
   ?- brother(adam, debbie).
   ?- brother(pip, josh).
   ?- brother(adam, kate).
   ?- brother(dan, jack).
   ?- brother(jack, jack).
```

The answers should be Yes, No, Yes, No, No.

Checkpoint

Now demonstrate to a tutor that you have tried all the goals associated with archers (steps 1 and 2 above), and successfully created the brother predicate (step 3 above).

On Your Own
Try for yourself entering further information (more facts) into the database, and then giving further goals to the interpreter.

Lastly, SWI Prolog has a website! See www.swi-prolog.org.