First steps in Prolog

How to start Prolog

The GNU Prolog system is documented at `gprolog.org`. That website has a manual. Gnu Prolog is free, and there are binaries available for most operating systems. GNU Prolog can be used in interactive mode, like a command interpreter (shell): the user types in queries, called *goals*, and the system replies with solutions to the queries, and prompts the user for a new query.

This interactive system must be started from a terminal using a Unix command interpreter (sh, csh, ksh,...); the command to start Gnu Prolog is `gprolog`.

Programs are written in files, using standard text editors. The name of the program files must end with .pl. The predicate consult is used to load programs: consult('my_prog.pl'). There are a few shortcuts for this type of very frequent goals: consult(my_prog). or even [my_prog]. or ['my_prog.pl']. This shows two things:

- the ' around the file name is used to identify an atom that contains some special characters (the "." here); the quotes are not necessary if the name of the atom does not contain any special character;
- every query is terminated with a ".".

After a query has been submitted, the Prolog system will try to compute a first answer, that may be yes, no, or a list of values for the variables that appear in the query: in this case, to obtain more solutions, one must type in a semi-colon.

Exiting Prolog The query halt. terminates the prolog interpreter.

Exercise 1 This exercise uses the "movie" database[a], you should download it, have a look at its content with a text editor and "consult" it with prolog.

Question 1.1 Use prolog to find actors and actresses who have:

- been directed by Brian de Palma;
- been directed by Tim Burton and also by Francis Ford Coppola:;
- played in at least two different movies by Sofia Coppola.
- been directed by by Tim Burton and not by Francis Ford Coppola:.
- played in exactly least two different movies by Sofia Coppola.

Syntax of Edinburgh / GNU Prolog
in order of increasing priorities:

Connective	Meaning	Logic	Prolog
implication	"if"	$\leftarrow$	`:-`
disjunction	"or"	$\vee$	`;`
conjunction	"and"	$\wedge$	`,`
negation	"not"	$\neg$	`\+`
equality	"equals"	$=$	`=`
inequality	"different"	$\neq$	`\=`

[a] www.irit.fr/~Jerome.Mengin/prolog/movie_bd.pl

Remark Most prolog interpreters require that the logical negation connective \+ is followed, in some contexts, by two pairs of parenthesis. It is safe to always put two pairs of parenthesis.

Question 1.2 Using a text editor, write - at the beginning of the movie database or in a separate file - definitions for predicates specified as follows:

- play_in/2, such that play_in(A, M) is true if A is an actor or an actress in movie M;
- directed_by/2, such that directed_by(A, D) is true if A played in a movie directed by D.
- common_costar/2, such that common_costar(A_1, A_2) is true if there is a third actor or actress, different from A_1 and A_2, who played with both A_1 and A_2 but in two different movies.

(Do not forget to "re-consult" your file every time you modify it.)

A few debugging tools

Step-by-step execution It is possible to see each "call" to a given predicate during execution of a program. The predicate `spy/1` is used to declare predicates to be "traced", e.g. `spy`(isAscendantOf). It puts the interactive system in tracing mode. The predicate `debug/0` also puts the system in tracing mode. A call to `nodebug/0` exits this mode.

In tracing mode, the interpreter stops when "calling" a traced predicate and when "exiting" the call, that is, when it has found instances of the variables that make the call "true"; at each stop, the interpreter waits for an instruction:

`l` to "leap" to the next call to, or return from, a marked predicate;
`A` to see all alternatives (branches that still have to be explored);
`a` to abort execution;
`g` to see the ancestor calls;
`h` to get some help.

Exercise 2 Trace the calls to the predicates movie and director during the execution of the queries:
movie$(M, 2006) \wedge$ director$(M,$ ethan_coen$)$;   movie$(M, 2006) \wedge \neg$director$(M,$ ethan_coen$)$.

Printing messages The predicate `write/1` can be used to print out a term on the screen, e.g. $p(X) \leftarrow \texttt{write}('X =') \wedge \texttt{write}(X) \wedge q(X)$.

The anonymous variable The variable _ is anonymous: it does not really have a name; each occurrence refers to a different variable.
It should be used whenever a variable has only one occurrence in a clause,
otherwise Prolog will write a *"singleton variable..."* message.

For instance, the clause $p(X,Y) \leftarrow q(X) \wedge r(X,Z) \wedge s(Z,U)$ should be written $p(X,\_) \leftarrow q(X) \wedge r(X,Z) \wedge s(Z,\_)$.

Exercise 3 Rewrite the programs of the exercises above so that you do not have the "singleton variable..." message when you consult them.

Always useful: working with numbers in Prolog

- Most usual arithmetic expressions are part of the Prolog language, for instance $\mathsf{sqrt}(4 * X - 3) + 2.3$.

However :

- Prolog was first designed to manipulate non numerical data ⇒the *evaluation* of arithmetic expressions is not automatic. For instance, try the query $X = \mathsf{sqrt}(4 * 5 - 3) + 2.3$.
- A Prolog predicate does not *return* a value

⇒ **The binary predicate** is evaluates an arithmetic expression, and instantiates a variable with the result. For instance, to the query: X is $\mathsf{sqrt}(4 * 2 - 3) + 2.3$ Prolog replies: $X = 4.53\ldots$

The following infix binary predicates expect arithmetic expressions on both sides: `<`, `>`, `=<`, `>=`, `=:=`, `=\=`.
They evaluate the two expressions, and compare the results.
(`X=:=Y` is true if the value of `X` is equal to the value of `Y`,
and `X=\=Y` is true if the value of `X` is different from that of `Y`).

Exercise 4 Define a predicate that can compute the value of $n! = n \times (n-1) \times \ldots \times 3 \times 2 \times 1$.

Exercise 5 Define a predicate quadEq/4 that computes the solutions of a quadratic equation: quadEq(A, B, C, X) should be true if $AX^2 + BX + C = 0$.

A bit of logic

Exercise 6 Once upon a time a farmer went to the market and purchased a fox, a goose, and a bag of beans. On his way home, the farmer came to the bank of a river and hired a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases - the fox, the goose, or the bag of the beans. If left alone, the fox would eat the goose, and the goose would eat the beans. The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact.[1] You will later write a Prolog program to discover how he did it, with a predicate that computes a sequence of crossings that leads from the initial state (the farmer and his goods on one side of the river) to the final state (the farmer and his goods on the other side). A state of the problem can be described with a lists of the elements on the initial side of the river. For instance, the initial state could be described by [f, g, x, c], the final state is just [ ].

Question 6.1 Define a predicate safeState/1 such that safeState(E) is true if E represents a state where the fox is not left alone with the goose, and the goose is not left with the corn. (Use the built-in predicate member/2.)

Exercise 7 Cannibals ambush a safari in the jungle and capture three men. The cannibals give the men a single chance to escape uneaten.

The captives are lined up in order of height, and are tied to stakes. The man in the rear can see the backs of his two friends, the man in the middle can see the back the man in front, and the man in front cannot see anyone. The cannibals show the men five hats. Three of the hats are black and two of the hats are white.

Blindfolds are then placed over each man's eyes and a hat is placed on each man's head. The two hats left over are hidden. The blindfolds are then removed and it is said to the men that if one of them can guess what color hat he is wearing they can all leave unharmed.

[1] en.wikipedia.org/wiki/Fox,_goose_and_bag_of_beans_puzzle

The man in the rear who can see both of his friends' hats but not his own says, ”I don't know”. The middle man who can see the hat of the man in front, but not his own says, ”I don't know”. The front man who cannot see ANYBODY'S hat says ”I know!”

How did he know the color of his hat and what color was it?[2]

(Hint: Prolog's negation as failure is perfect to represent “I cannot guess the color of my hat knowing. . . ”. You can define a first predicate that enumerates the possible hat combinations.)

[2]Excerpt from mathsisfun.com