

1 The Prolog interpreter

As seen during the lecture, Prolog is an *interpreted* language:

- no executable is created (even if it is possible)
- a Prolog program is loaded into the interpreter
- queries are executed on this program through the interpreter

We will use the GNU Prolog interpreter [1, 2].

To start the interpreter:

```
c.garion@chabichou# gprolog
GNU Prolog 1.3.1
By Daniel Diaz
Copyright (C) 1999-2009 Daniel Diaz
| ?-
```

The | ?- prompt waits for a query.

To escape from the interpreter, use the `halt` predicate:

```
|?- halt.
```

2 Writing and interpreting Prolog programs

To write a Prolog program, you must respect the following guidelines (**remember that case is important**):

- identifiers beginning by an *uppercase letter* are *variables*
- when speaking of a predicate, its arity is given (`ancestor/2` for instance)
- a clause is written like this:

```
A :- B1, ..., Bn.
```

where `A` is the head of the clause and `B1, ..., Bn` its body.

When a clause is written `A :- true.`, it is simply noted `A`. Such a clause is called a *fact*.

Do not forget the “.” at the end!

For instance:

- `jack` is a parent of `mary`
 ➔ `parent(jack, mary).`
- for all `X` and `Y`, if there exists a `Z` such that `X` is an ancestor of `Z` and `Z` is a parent of `Y`, then `X` is an ancestor of `Y`
 ➔ `ancestor(X, Y) :- ancestor(X, Z), parent(Z, Y).`

- comments are written with `/*` and `*/`

We will consider in the following the “ancestors” program seen during the lecture (cf. listing 1). Download it and start GNU Prolog in the same directory.

Listing 1: The Prolog program about ancestors

```

/*****
/* Definition of parent/2 */
*****/
parent(jack, mary).
parent(louise, jack).
parent(franck, john).

/*****
/* Definition of ancestor/2 */
*****/
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- ancestor(X, Z), parent(Z, Y).

```

To load the ancestor program in the interpreter:

```

| ?- ['/home/tof/lectures/logic/prolog-labs/src/ancestors.pl'].
compiling /home/tof/Cours/IN112/exempleProlog/ancetres.pl for byte code...
/home/tof/Cours/IN112/exempleProlog/ancetres.pl compiled,
 12 lines read - 898 bytes written, 62 ms

(2 ms) yes

```

Prolog answers **yes**: the predicate **consult/1** is evaluated successfully and the program is loaded.

3 Using VSCode

You can use VSCode as an editor for your Prolog source files. You should use Arthur's Wang [VSC-Prolog](#) plugin. Change Prolog executable path to `gprolog` in VSCode settings.

Within VSCode, you just have to hit **Alt-X** and then **l** to load the current file in the interpreter.

4 Evaluating queries

To evaluate a query:

```

| ?- ancestor(jack,mary).

true ?

```

The “?” symbol signifies that Prolog waits for a user command to continue (or not) to build the search tree:

- **;** to ask for the next solution (backtracking)
- **a** to ask for all solutions
- **RETURN** to stop

With the previous program, after typing **;**:

```

Fatal Error: local stack overflow (size: 8192 Kb,
            environment variable used: LOCALSZ)

```

TODO: correct the program and evaluate some queries.

5 Unification, assignement, equality

You may be confused by four Prolog operators: unification, assignement, term equality and arithmetic equality.

1. the **unification** operator is `=` (its contrary is `\=`). When encountering this operator, Prolog tries to unify by applying substitutions on **both terms**. Try for instance:

```
X = Y.
X = Y, f(Y) = Z.
f(X) = g(Y).
f(X) \= g(a).
```

2. the **assignment** operator is `is`. Its right operand should be an **evaluable** term.

Try for instance:

```
X is 2.
X is 2 + 2.
X is Y.
Y = 2, X is Y.
```

3. the **terms equality** operator is `==` and is used to verify if two terms are syntactically identical.

Try for instance:

```
X == X.
X == Y.
X \== Y.
2 == (1 + 1).
```

4. the **arithmetic equality** operator is `:=`.

Try for instance:

```
2 := 2.
2 := (1 + 1).
2 := 3.
2 \= 3.
```

6 A first predicate: factorial

Define a predicate `fact/2` which computes the factorial of a given number (why is the arity of the predicate 2?). You can use the `trace/0` and `notrace/0` predicates to activate and deactivate the debugger.

7 Working with lists

Lists are classically defined by induction:

- empty list: `[]`
- else `[head | tail]` with `head` an element and `tail` a list

A list containing *known elements* is represented using `" , "`: `[a, b, c]`.

Try the following example to understand how Prolog use unification with lists:

```
[T | Q] = [a, b, c].
[T, Q] = [a, b, c].
[T, Q, R] = [a, b, c].
[T | Q] = [a].
[T] = [a].
```

Lots of predicates are already defined on lists: `member/2`, `append/3`, ...cf. [2]

References

- [1] D. Diaz. *GNU Prolog*. 2013. URL: <http://www.gprolog.org>.
- [2] D. Diaz. *GNU Prolog Manual*. 2013. URL: http://www.gprolog.org/manual/html_node/index.html.

License CC BY-NC-SA 3.0



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license (CC BY-NC-SA 3.0)

You are free to Share (copy, distribute and transmute) and to Remix (adapt) this work under the following conditions:



Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Noncommercial – You may not use this work for commercial purposes.



Share Alike – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

See <http://creativecommons.org/licenses/by-nc-sa/3.0/> for more details.