Imperial College London – Department of Computing

Exercise 1: 'Introduction to Prolog'

11 November 2013

Objectives

The objective of this laboratory session is to get you started with the programming language Prolog and the Sicstus Prolog environment.

I. Introduction to Sicstus Prolog

This exercise uses a simple Prolog program `sample.pl`. Download a copy from CATE.

- To start the Sicstus environment type the following command into a terminal window:

  ```
  sicstus
  ```

  The prompt `| ?-` indicates that the environment is ready to receive commands from you, such as loading a program or executing a query.

  It is normal to work with two windows simultaneously: one for the editor, the other for Sicstus. To this effect, exit Sicstus by typing `'halt.'` or pressing `Ctrl-D`, and launch the editor (in the background) and Sicstus together by typing:

  ```
  gedit sample.pl &
  sicstus
  ```

  (You can use any other text editor if you have a preferred one.)

  Alternatively (better): open two terminal windows and use one for Sicstus and the other one for everything else (editors, examining directories and files, etc.).

- Load the program into the Sicstus environment, by typing:

  ```
  consult(sample).
  ```

  or the shorthand version of the same command: `'[sample].'`. This will look for a file named `sample.pl` in the current working directory. Note that all the Prolog commands and clauses must finish with a dot (.).

- Try the following queries or commands:

```
check_number( 0 ).
check_number( 100 ).
check_number( 531 ).
```

  (The spaces around the numbers are just for readability.)

  What is wrong with the last query? It loops! To interrupt its execution type `Ctrl-C`. You'll get the prompt

```
Prolog interruption (h for help)?
```

  Reply with `a`, for abort, in order to stop the execution.

- The reason for the infinite loop is the lack of a terminating case for odd numbers. To fix this, add the clause

```
check_number( 1 ).
```

  to be the first or second clause in your copy of the `sample.pl` file. Note that this modification has to be done in the window that is running the editor, and that you have to save the file, before returning to the window running Sicstus and reloading the file by typing:

  `consult(sample).` or `[sample].`

- To exit the Sicstus Prolog environment use the command `halt`. Don't forget that all Prolog commands and clauses finish with a dot.

- You can invoke the complete on-line manual of Sicstus Prolog (PDF format) by using the shell command `sicstus-manual`. There is also a version on the web (see below). Have a brief look through it to get an idea of the range of built-in predicates and utility libraries provided. There is no need to study it in detail.

Some notes

- The Department of Computing has installed Sicstus Prolog on all lab machines, running under Linux and Windows. The assessed lab exercises will be marked under Linux, so it is essential that you ensure your submissions run properly on the lab Linux installation. You might need to use the `fromdos` utility for converting source files if you develop your programs using the Windows implementation.

  Sicstus itself is available for download under a DoC student licence agreement if you would like to install it on your own computers. See 'Software sources' under the CSG pages at `/vol/csg/download/software/sicstus`.

- In order to avoid destroying trees by printing off the manual, see

```
http://www.sics.se/sicstus/docs/latest4/html/sicstus.html
```

  or type `sicstus-manual` at the command line.

- There are several other Prolog implementations besides Sicstus. A popular and stable alternative, preferred by some users, is SWI-Prolog. It is free, and can be downloaded from `www.swi-prolog.org` if you would like to install it on your own computers. YAP Prolog, also free, is the fastest implementation generally available. You can download it at `www.yap.sourceforge.net`.

  Please note: although there should be very few differences between Sicstus and these other implementations of Prolog for the programs you will developing on this course, all assessed work is marked (automatically) using Sicstus on Linux.

- Simon Coffey (spc03) writes: After about eight seconds of using Sicstus under Linux, you will notice that it doesn't do command-line history or editing. Four seconds later, this will make you want to break something. Don't. Installed on the lab machines is a utility called `rlwrap` which will make your life much easier. If you run sicstus with the command:

  ```
  rlwrap sicstus
  ```

  command-line editing will be enabled. It even remembers the commands you used last session. If you add the following line to your `.cshrc` file (located in your home directory), you can launch it as normal by just typing `sicstus`

  ```
  alias sicstus "rlwrap sicstus"
  ```

  You can use whichever text editor you like. Some support Prolog syntax highlighting.

  For `emacs` users, there is a Prolog interface. See the Sicstus manual for details.

II. Prolog Database

The rest of this exercise is based on a database representing the fragment of a family tree illustrated in Figure 1.

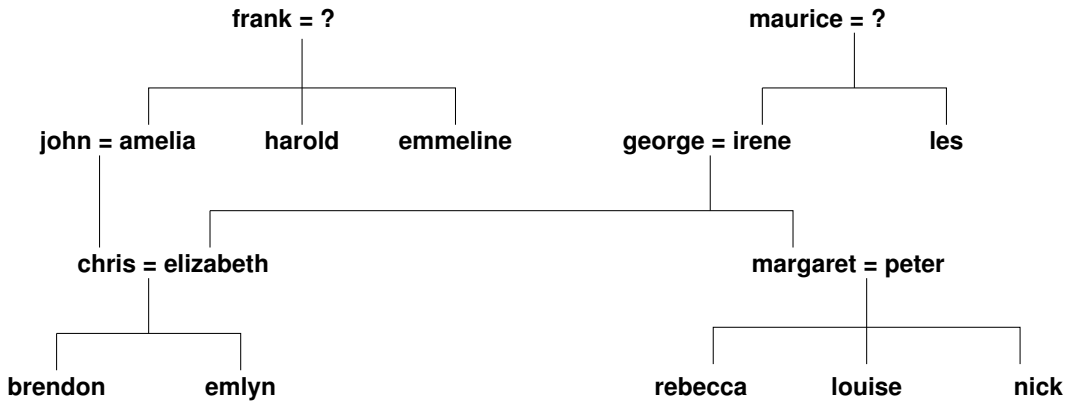

Figure 1: Family Tree

The database is the set of atomic facts contained in the file `family.pl`. Download a copy from CATE. Load it in Sicstus with '`consult(family).`' or '`[family].`'. The file contains clauses about three predicates:

```
child_of( X, Y )        "X is a child of Y"
male( X )               "X is male"
female( X )             "X is female"
```

For instance, the first fact in the database is

```
child_of( emmeline, frank ).
```

and expresses "emmeline is a child of frank".

1. Enter the following queries, finding all the solutions:

```
child_of( peter, irene ).
child_of( peter, emlyn ).
child_of( X, george ).
child_of( george, Y ).
child_of( X, X ).
child_of( X, Y ).
```

2. Add further Prolog clauses which define the following predicates:

```
mother_of( M, X )               "M is the mother of X"
grandparent_of( GP, X )         "GP is a grandparent of X"
daughter_of( D, X )             "D is a daughter of X"
uncle_of( Unc, X )              "Unc is an uncle of X"
niece_of( N, X )                "N is a niece of X"
great_grandfather_of( Gfx, X )  "Gfx is a great-grandfather of X"
ancestor_of( Anc, X )           "Anc is an ancestor of X"
```

   For the last one, compare the clauses for `superiorOf/2` in the lecture notes.

   Test each one by posing suitable queries and finding all their solutions. As a guiding example, this is how we might define "X is a sister of Y":

```
sister_of( X, Y ) :-
        child_of( X, Z ),
        child_of( Y, Z ),
        X \= Y,
        female( X ).
```

   (This definition is for illustration: it treats step-sisters as sisters.)

   Note that in Prolog `\=` means 'do not unify'. You could also use `\==` (backslash, equals, equals) which means 'not identical'.

   (If you don't understand the difference, compare what you get when executing Prolog queries `a \= b`, `a \== b`, `X \= a`, `X \== a`, `X \= Y`, `X \== Y`. Try the corresponding queries with `=` and `==`.)