



Logic Programming

Alan Smaill

Sep 16 2013

- ▶ **Logic Programming** is to a different paradigm from either **imperative** or **functional** programming languages.
- ▶ The best known Logic Programming language is **Prolog**; it is like Haskell and other functional programming languages in having a **declarative** reading.
- ▶ According to an old joke, Logic Programming was invented in Edinburgh in 1974, and implemented in Marseille in 1972.
- ▶ The approach came out of connections observed between natural language parsing, general theorem proving in first-order logic, and AI planning.

It would be great if:

Instead of writing an algorithm to solve a given problem, we can just specify the problem (in first-order logic) and let the machine solve the problem for us.

Logic Programming achieves this —

to an extent, certainly not always; but actually we know that this wish cannot be fully realised.

So languages like Prolog are practical solution, which given a good understanding of the approach, lets us solve (quite a lot of) problems quickly.

- ▶ Third year course, worth 10 points.
- ▶ The assessment is by **exams** (80%);
and two assessed courseworks (20%).
- ▶ There are tutorials (from week 3), which are an integral part
of the course.
- ▶ There will be two exams:
 - ▶ 1 hour on theoretical material
 - ▶ 2 hour programming exam in computer lab.

Official descriptor:

*[http://www.drps.ed.ac.uk/13-14/dpt/
cxinfr09031.htm](http://www.drps.ed.ac.uk/13-14/dpt/cxinfr09031.htm)*

and course web page:

*[http:
//www.inf.ed.ac.uk/teaching/courses/lp/](http://www.inf.ed.ac.uk/teaching/courses/lp/)*

It will help if you have seen first-order logic before
the aim is that you will

- ▶ Understand the principles of declarative specification.
- ▶ Be able to construct well crafted Prolog programs of moderate size and sophistication.
- ▶ To be able to interpret problems in a style that suits logic programming.

Today we aim

- ▶ to get a general grasp of the main ideas behind Logic Programming;
- ▶ why you might use it;
- ▶ and how to get started programming in LP.

- ▶ Program **specifications** can be written in logic.
- ▶ Specifications are independent of computers.
- ▶ Rules of logic can prove the correctness of a specification even if computers didn't exist.
- ▶ But proof can also be done by a computer smart enough to find the right proof.
- ▶ So specifications and programs are . . . the same.
- ▶ So specifications and programs are **nearly** the same.

- ▶ Slogan: “Algorithm = Logic + Control”
- ▶ The program should simply **describe** what counts as a solution to the program.
- ▶ The computer then **finds** the solution.
- ▶ Programmers should be able to ignore how the solution is found.

- ▶ Purely declarative programming can only get you so far
- ▶ For efficiency/termination, sometimes need finer-grained control over search.
- ▶ I/O, interaction with outside world, seem inherently "imperative"

- ▶ Prolog is the best-known LP language
 - ▶ Core based on first-order (predicate) logic
 - ▶ Algorithmic realisation via unification, search
- ▶ Many implementations that make it into a full-fledged programming language
 - ▶ I/O, primitive ops, efficiency issues complicate declarative story



- ▶ LP often great for rapidly prototyping algorithms/search strategies
- ▶ “Declarative” ideas arise in many areas of CS
 - ▶ LP concepts very important in AI, databases, PL
 - ▶ SAT solvers, model-checking, constraint programming
 - ▶ Becoming important in program analysis, Semantic Web
- ▶ Learning a very different “way to think about problems” makes you a better programmer

Well use **SICStus** Prolog.

- ▶ Available on all DICE machines
 - ▶ Tutorials, exams will be based on this version
- ▶ Windows, Mac version free for UofE
 - ▶ Can request through Computing Support
- ▶ On-line documentation

<http://www.sics.se/isl/sicstuswww/site/>

Prolog is an interactive language.

```
$ sicstus
?-
?- print( 'hello world').
hello world
yes
```

We see the result of the print command,
and also the **response** yes.

An **atom** is:

- ▶ a sequence of alphanumeric characters
 - ▶ usually started with a lower case letter
- ▶ or a string enclosed in single quotes

Examples:

```
homer marge17 'Mr. Burns'
```

A **variable** is a sequence of alphanumeric characters, usually starting with an uppercase letter.

Examples:

X Y Parent Foo

A predicate has the form

$$p(t_1, \dots, t_n)$$

where p is an atom, and t_1, \dots, t_n are terms.

For now, a term is just an atom or variable

Examples:

`father(homer, bart)`

`mother(marge, bart)`

A predicate has

- ▶ a name – father in father(homer, bart)
- ▶ an arity – how many arguments: 2 in father(homer, bart)

Predicates with the same name, but different arity, are **different**.

We write `foo/1`, `foo/2`, ... to refer to these different predicates.

A **fact** is an assertion that in instance of the predicate is true:

```
father(homer, bart).
```

```
mother(marge, bart).
```

Notice the full stops!!

A collection of facts is sometimes called a **knowledge base**.

A **goal** is a sequence of predicates, connected by commas – we understand this as **conjunction**:

$$p(t_1, \dots, t_n), \dots, q(t_1', \dots, t_n').$$

We read this as saying p holds of t_1, \dots, t_n , and also similarly for other predicates.

Predicates can be 0-ary (no arguments); there are some built-ins: `true`, `false`, `fail`

Given a goal, Prolog searches for **answers**

- ▶ yes (possible with answer substitution)
- ▶ no

Substitutions are bindings of variables that make goal true

Use “;” to see more answers.



Suppose we have Prolog facts (here in simpsons.pl):

```
father(abe,homer).
father(homer, bart).
father(homer, lisa).
father(homer, maggie).
father(ned, rod).
father(ned, todd).
father(chief_wiggum,ralph).

mother(marge, bart).
mother(marge, lisa).
mother(marge, maggie).
...
```

We can now query, and Prolog will search for possible answers:

```
?- father(X,bart).
```

```
X = homer ;
```

```
no
```

```
?- father(X,Z), mother(Y,Z).
```

```
X = homer, Y = marge, Z = bart ;
```

```
X = homer, Y = marge, Z = lisa ;
```

```
X = homer, Y = marge, Z = maggie ;
```

```
no
```

A **Rule** is an assertion of the form

$$p(ts) \text{ :- } q(ts), \dots, r(ts).$$

where t_1, t_2, t_3 are sequences of terms.

This means:

$p(ts)$ holds if $q(ts)$ holds and ... and $r(ts)$ holds.

$\text{sibling}(X,Y) \text{ :- } \text{parent}(Z,X), \text{parent}(Z,Y).$

Is this a good definition of sibling?

▶ Comments:

```
% percent comments out rest of line
```

```
/* multiple  
    line comment */
```

▶ To quit Sicstus, type

```
?- halt.
```

...or control-D.

- ▶ A Prolog program is a collection of facts and rules; together these are known as **clauses**
 - ▶ stored in one or more files
- ▶ The predicate `consult/1` loads the clauses in a file:

```
?- consult('simpsons.pl').
```

or without the `.pl` extension:

```
?- consult(simpsons). or
```

```
?- [simpsons].
```

```
/* hello.pl
 * James Cheney
 * Sept. 20, 2010
 */

main :- print('hello world').
```

Most Prolog implementations have good tracing facilities.

- ▶ `trace/0` turns on tracing
- ▶ `notrace/0` turns tracing off
- ▶ `debugging/0` shows tracing status

- ▶ Course text:
“Learn Prolog Now!” (Blackburn et. al.): on-line at:
<http://www.learnprolognow.org/>
- ▶ Quick Start Prolog notes (David Robertson):
<http://www.inf.ed.ac.uk/teaching/courses/lp/2008-9/prolognotes.pdf>

Using `simpsons.pl`, write goal bodies for:

- ▶ `classmate(X,Y)`
- ▶ `employer(X)`
- ▶ `parent(X,Y)`
- ▶ `grandparent(X,Y)`

- ▶ Compound Terms
- ▶ Equality and Unification
- ▶ How Prolog searches for answers