Logic Programming

Programming Lecture 1:
Getting started with Prolog

Recap

- Logic programming is a form of *declarative* programming
  - "Algorithm = logic + control"
  - Specify a problem, let computer find solution
  - This does not always work out as well as we would wish
  - Writing effective logic programs generally still requires pragmatic knowledge

Why learn LP?

- Learning a very different "way to think about problems" makes you a better programmer
    - LP works well for rapidly prototyping algorithms/search strategies, which can be transferred to mainstream language
- "Declarative" ideas arise in many areas of CS and AI
    - LP concepts very important in AI, databases, PL
    - SAT solvers, model-checking, constraint programming
    - Becoming important in program analysis, Semantic Web
- Learning how *logic* provides a foundation for *computation* can improve your understanding of both

Further reading

- Quick Start Prolog notes (Dave Robertson)

http://www.inf.ed.ac.uk/teaching/courses/lp/prolognotes.pdf

- Learn Prolog Now! (Blackburn, Bos, Striegnitz)
  - online, free

http://www.learnprolognow.org/

- Programming in Prolog (Clocksin & Mellish)
  - a standard/classic text, many library copies

Hello World

- Prolog is an interactive language.

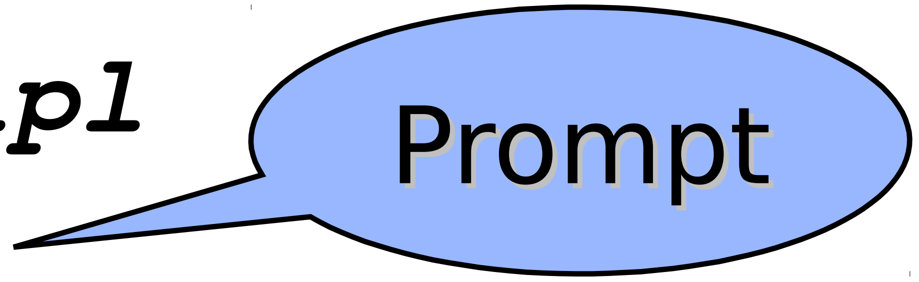
```
$ swipl
```

Hello World

- Prolog is an interactive language.

```
$ swipl

?-
```



Hello World

- Prolog is an interactive language.

```
$ swipl

?- print('hello world').
```

Goal

Hello World

- Prolog is an interactive language.

```
$ swipl

?- print('hello wo

hello world

true
```

Output

response

Atoms

- An **atom** is
  - a sequence of alphanumeric characters
    - usually starting with lower case letter
  - or, a string enclosed in single quotes
- Examples:

```
homer   marge   lisa   bart
‘Mr. Burns’ ’Principal Skinner’
```

Variables

- A **variable** is a sequence of alphanumeric characters
  - usually starting with an uppercase letter
- Examples:

`X  Y  Z  Parent Child Foo _Bar`

Predicates

- A **predicate** has the form

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

where p is an atom and $t_1 \ldots t_n$ are terms

(For now a term is just an atom or variable)

- Examples:

```
father(homer, bart)

mother(marge, bart)
```

Predicates (2)

- A predicate has a **name**
  - = atom `p` in `p(t_1,...,t_n)`
- and an **arity**
  - = number of arguments (n)
- Predicates with **same** name but **different** arity are **different**
- We write `foo/1`, `foo/2`, ... to refer to these different predicates

Facts

- A **fact** is an asserti predicate is true:

Punctuation is important!

```
father(homer, bart).

mother(marge, bart).
```

- A collection of facts is sometimes called a **knowledge base** (or **database**).

Goals

- A **goal** is a sequence of predicates

`p(t₁,...,tₙ), ..., q(t₁',...,tₙ').`

- We interpret "`,`" as **conjunction**
- Logically, read as "`p` holds of $t_1 ... t_n$ and ... and `q` holds of $t_1' ... t_m'$"
- Predicates can be 0-ary
  - Some built-ins: `true`, `false`, `fail`

Answers

- Given a goal, Prolog searches for **answer(s)**
  - "***`true`***" (possibly with **answer substitution**)
  - "***`false`***"
- **Substitutions** are bindings of variables that make goal true
- Use ";" to see more answers

Examples

```
?- father(X,bart).
X = homer ;
false
?- father(X,Z), mother(Y,Z).
X = homer, Y = marge, Z = bart ;
X = homer, Y = marge, Z = lisa ;
X = homer, Y = marge, Z = maggie ;
false
```

Rules

- A **rule** is an assertion of the form

```
p(ts) :- q(ts'), ..., r(ts'').
```

where `ts`, `ts'`, `ts''` are sequences of terms

- "`p(ts)` holds if `q(ts')` holds and ... and `r(ts'')` holds"
- Example:

```
sibling(X,Y) :- parent(Z,X),
                parent(Z,Y).
```

Miscellaneous

- Comments

```
% single line comment
/* multiple
   line
   comment */
```

- To quit swipl, type

```
?- halt.
```

  - (or just control-D)

Consulting

- A Prolog program is a collection of facts and rules, or **clauses**
  - stored in one or more files
- The predicate ***`consult/1`*** loads the facts/rules in a file

```
?- consult(‘simpsons.pl’).
```

Consulting (2)

- If the file name ends with '.pl', can just write:

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

- Also, can just write

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

A complete program

```
/* hello.pl
 * James Cheney
 * Sept. 18, 2014
 */

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

Tracing

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

More kinds of terms

- Story so far...
  - Atoms: ***homer marge 'Mr. Burns'***
  - Variables: ***X Y Z MR_BURNS***
- Also have...
  - **Numbers**: *1 2 3 42 -0.12345*
  - **Lists** *[1,2,3]* and other **complex terms**
  - Additional **constants** and **infix operators**

Complex terms

- A complex term is of the form

$$\texttt{f(}t_1\texttt{,...,}t_n\texttt{)}$$

- where ***f*** is an atom and $t_1 ... t_n$ are terms
- Examples:

f(1,2) node(leaf,leaf) cons(42,nil)

household(homer, marge, bart, lisa, maggie)

More about lists

- Lists are built-in (and very useful) data structures
- Syntax:

```
[1,2,3,4]

[a,[1,2,3],42,'forty-two']

[a,b,c|Xs]
```

- (Lots) More next week

Exercises

- Using ***`simpsons.pl`***, write goal bodies for:
  - ***`classmate(X,Y)`***
  - ***`employer(X)`***
  - ***`parent(X,Y)`***
  - ***`grandparent(X,Y)`***
- More in tutorial problems handout