Today

- Prolog

Assignments

- HW-7, R-7 due Thurs
- HW-8, R-8 out
- Proj 2 out
Prolog

Prolog stands for PROgramming in LOGic

- Prolog is a “logic-based” programming language
- Syntax is close to first-order logic (FOL)

The first Prolog system was developed circa 1972

- Colmerauer, Roussel, Kowalski
- Univ. of Aix-Marsielle and Univ. of Edinburgh
- Born out of work in Automated Theorem Proving
- As well as Natural Language Understanding
- Because of this, considered an “AI” language
Some features of Prolog

• Extremely simple syntax

• Declarative (what not how)
  – Haskell leans declarative as well

• A very different way to think about programming
  – Use “rules” or “constraints”
  – Hard to pick up at first ...

• Efficient at certain problems
  – Prolog runtime is fairly small ...
  – Prolog itself is generally fast (wasn’t always)

• No data types!

Prolog programming today

Not widely used today to build applications ...

• However, many languages that are used are “prolog” like
  – SQL (queries)
  – Datalog (Prolog like)
  – Answer Set Programming (constraints)
  – Erlang (concurrent/parallel computing ... mixes Prolog and functional)
**SWI Prolog**

We’ll be using SWI Prolog ...

- [http://www.swi-prolog.org](http://www.swi-prolog.org)
- freely available, open-source implementation
- supports most operating systems (mac, linux, windows)
- available on ada (run `swipl` at the command prompt)
- lots of documentation online

**Like with Haskell**

- you are encouraged to download, install, and run on your own machine
- if you need help, come see me

**Starting SWI prolog on ada**

```
$ swipl
Welcome to SWI-Prolog ...

For help, use ?- help(Topic) ...

?-
```

- The ?- symbol is the **query prompt**
Prolog syntax

Prolog programs are built from “terms”

- A term is either:
  - a `constant`
  - a `variable`
  - or a `structure`

Constants are either “numbers” or “atoms”

- numbers are integers or floating point values
- atoms represent everything else

Atoms

- Atoms usually begin with lowercase letters
  - can contain letters, digits, and underscores (\_)
- Atoms in single quotes denote strings
  - any character can occur between single quotes
  - different from a list of char (denoted by double quotes)
- Some atoms contain only symbols (e.g., +)
- The `true` and `false` atoms are special
Examples of atoms

x
washington
a123BC_d

• note these are different than in most other languages!!!
• these are not variables!

'Washington'
'this is a long atom with spaces'
'\one back slash'
'this statement isn’t true'

• the last one prints “isn’t”
• ’isn’t’ does the same thing
Variables

Variables begin with either ...

- uppercase characters
- an underscore (don’t care / wildcard)

Variables can contain letters, digits, underscores ...

X
A
_1234
My_Var
Xs
Structures

Structures normally take the form:

\[ p(t_1, t_2, \ldots, t_n) \]

- \( p \) is an atom and called a **functor**
- each arg \( t_i \) is a **term**
- the whole thing is a **relation**
- the number of terms \( n \) is the **arity**

**Examples**

- \( \text{mother(necessity,invention)} \)
- \( \text{empoyed_by(X,gonzaga_university)} \)
- \( \text{employee(‘John’,‘Smith’,55567,28,x5555)} \)
- \( ‘a \text{ weird atom!?’}(x,y,\text{Var}) \)
- \( \text{node(‘a’,node(‘b’,nil,nil),nil)} \)

- an atom is a structure with no arguments (arity = 0)
Facts and Rules

A knowledge base (KB) consists of facts and rules

• all statements end in periods ...

/* facts */
mother(necessity,invention).
mother(june,wally).
father(experience,wisdom).
father(ward,wally).
human(june).
human(wally).
human(ward).

/* rules */
parent(X,Y) :- mother(X,Y). % parent if mother
parent(X,Y) :- father(X,Y). % parent if father
human_parent(X,Y) :- parent(X,Y), human(X). % "and"

• :- is an infix functor read “if”
• e.g., “X is the parent of Y if X is the mother of Y”
• and “X is a human parent of Y if X is the parent of Y and X is human”

Load these into swi by “consulting” the file:
• e.g. ?- consult('the_file.pl').
• or: ?- ['the_file.pl'].
• or: ?- [the_file].
• To exit swi: ?- halt.
?- consult('family.pl').
% family compiled 0.00 sec, 1,816 bytes

?- [family] % alternative load
% family compiled 0.00 sec, 1,816 bytes

?- parent(X,Y). % this is called a "query"
X = necessity,
Y = invention ;
X = june,
Y = wally ;
... etc ... .

- the ; tells prolog to search for more results
- you can hit enter to stop searching
Prolog is agnostic about inputs and outputs

?- parent(X,invention).  % X is an "output"
X = necessity.

?- parent(necessity,Y).  % Y is an "output"
Y = invention.

?- parent(X,Y).  % X & Y are "outputs"
X = invention,
Y = wisdom.

?- parent(necessity,invention). % both inputs
true.

• Some rules require certain “binding patterns” ... more later