Prolog – Section 12.2

Programming Languages

Objective

Prolog lecture's primary objective:

- Introduce you to Prolog
- Show how to accomplish arithmetic, filtering and recursion in Prolog (Lab 1)
- Show how Prolog resolves queries (Lab 2)
- Represent facts and relations in Prolog, to solve problems (Lab 2 & 3)

Prolog Uses Horn Clauses

Named after Alfred Horn (1951)

Format:

```
Head :- Body.
father(X,Y):-parent(X,Y),male(X).
```

If no body, have a fact

```
father(ed, celia).
```

Make queries at the prompt

```
?-father(ed, celia).
```

Example Prolog Database

```
/* Database of  3 facts and 1 rule */
 rainy(seattle).
 rainy(rochester).
 cold(rochester).
 snowy(X):-rainy(X), cold(X).
```

Queries:

```
 ?-rainy(seattle).
 ?-cold(seattle).
 ?-snowy(rochester).
 ?-snowy(seattle).
 ?-snowy(C).
```

Example Prolog Database

```
/* Database of  3 facts and 1 rule */
 rainy(seattle).
 rainy(rochester).
 cold(rochester).
 snowy(X):-rainy(X), cold(X).
```

What do you get from ther queries?

```
 ?-rainy(seattle).          => true.
 ?-cold(seattle).           => false.
 ?-snowy(rochester).        => true.
 ?-snowy(seattle).          => false.
 ?-snowy(C).                => C = rochester.
```

Example Prolog Database

/* Database of 5 facts and 1 rule */
f(a).
f(b).
g(a).
g(b).
h(b).
k(X) :- f(X), g(X), h(X).

From: http://cse3521.artifice.cc/prolog-resolution.html

Example Prolog Query

```
/* Database of  5 facts and 1 rule */
f(a).
f(b).
g(a).
g(b).
h(b).
k(X) :- f(X), g(X), h(X).
```

Query on the above database:

```
?- k(Y).
```

Prolog Execution

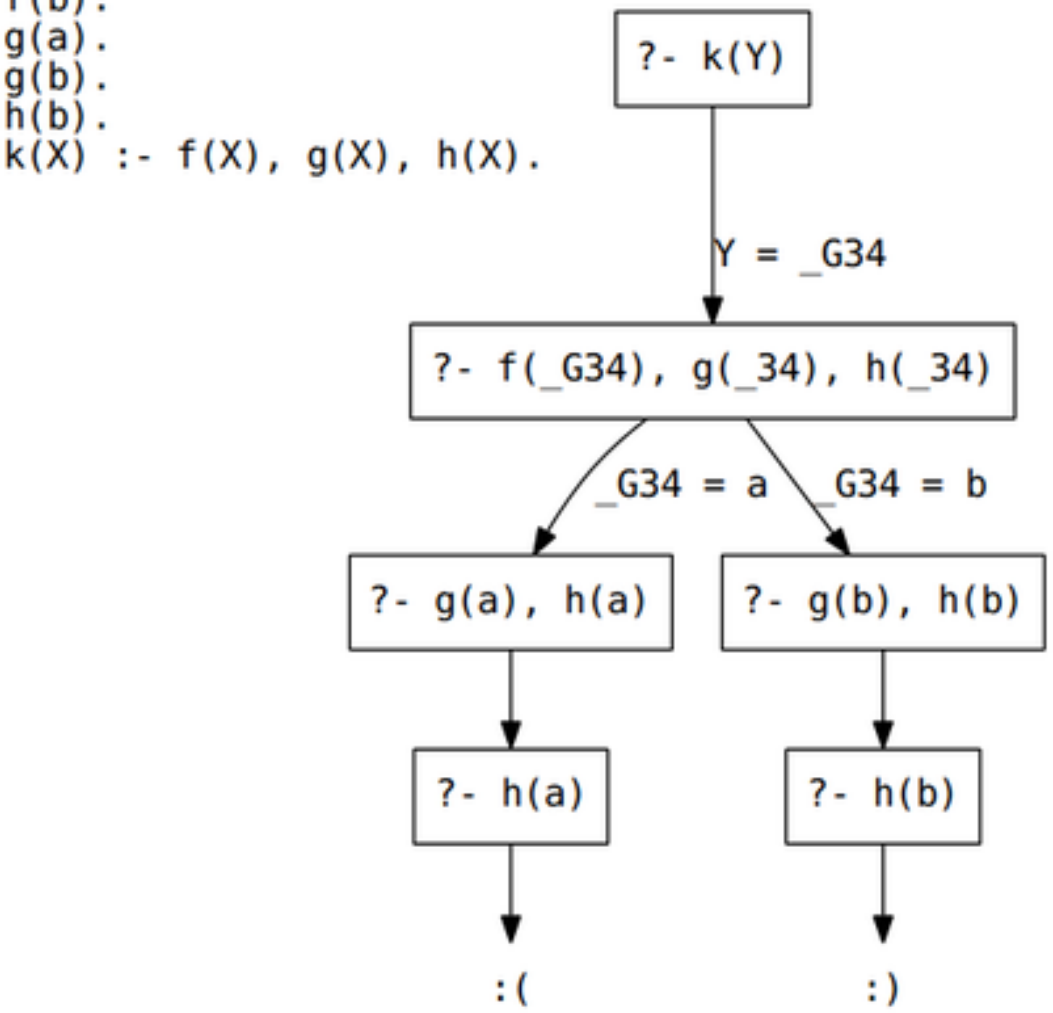
Database engine uses:

- unification – matching terms and substituting variables
- resolution – taking two clauses, and “resolving” them to return a single clause

Execution viewed as a search tree

Query:

?- k(Y).



Lab 1

Repeat the questions of the 1st Scheme lab

Prolog Execution

Database of facts and rules against which queries are written.

Load the database with:

```
?- consult(filename).
                % file extension .pl
                % don't need quotes
```

Ask query with

```
?- k(Y).
```

(Alternatively, when using the buffer, compiling the code the code handles the "consult" for you.)

Prolog Syntax

Head always a single term

Body can be multiple terms

‘,’ is conjunction (and)

‘;’ is disjunction (or)

group with parentheses

Example:

k(X) :- (f(X), g(X), h(X)); i(X).

variables start with a capital letter

constants and predicates start with a lower case letter

Prolog Syntax Continued

Body can contain:

‘_’ is special symbol which matches with anything except another _

```
?- mother(peggy,_).        % True if peggy is the mother of
                           % anyone
?- mother(X,_).            % X will get that value of any mother
```

Prolog Syntax

Conditional statement:

```
(condition ->
        do_when_true ; do_otherwise).
```

Comparisons

Numeric:
=, <, =<, >, >= (Notice '=' on left with < and right with >)

Alphabetic:
@=, @<, @=<, @>, @>=

Example:
?- likes(mary, pizza) @< likes(mary, plums).
true.

Prolog Syntax

'=' triggers unification

'is' triggers arithmetic evaluation

Unification – matches terms and substitutes variables (if one side is a variable, it will be assigned the value on the other side)

```
?- a = a.        % Two identical atoms unify
True

?- X = a.        % Unification instantiates a variable to an atom
   X=a

?- [a,b,c] = [X | Y].   % Unification using the '|' symbol  can
   X=a.                  % gives the head and tail
   Y=[b,c] .
```

Prolog Syntax

https://www.swi-prolog.org/pldoc/doc_for?object=manual

https://www.swi-prolog.org/pldoc/man?section=syntax

https://www.swi-prolog.org/Links.html

None of these are too helpful.

Lab 2

See how Prolog resolves queries

Represent facts and relations in Prolog, to solve problems

Debugging

Debugging is currently limited. Can only do tracing:

```
?- trace.
?- notrace.
```

From : https://www.swi-prolog.org/pldoc/man?section=practical

Debugging Ports

Debugging ports related to constraints are:

call – a new constraint is called and becomes active

exit – an active constraint exists: it has either been inserted in the store after trying all rules or has been removed from the constraint store

fail – an active constraint fails

redo – an active constraint starts looking for an alternative solution

wake - a suspended constraint is woken and becomes active

Tracing

When enabled, tracer steps through call, exit, fail, redo and simply writes out the port names. It supports the following debug commands:

creep – step to the next port

skip – skip to exit port of this call or wake port

break – enter a recursive Prolog top level

abort – exit to top level

fail – insert failure in execution

help – print the above available debug options

Lab 3

Continue to represent facts and relations in Prolog, to solve problems

Horn Clauses

When get results:

Press enter if done

Press space bar if you want more results

?- consult(ancestry).
?- mother(X, celia).
?- mother(peggy, Child).
?-mother(X,Y).

Prolog Syntax Continued

Body can contain:

‘\+’ is negation (not)

```
?- \+mother(peggy, celia).          % false.
?- \+mother(ed, celia).             % true.

% People who aren’t celia’s mother
?- \+mother(X, celia).              % false.

% People with a mother but not a father
?- mother(_,Y), \+father(_,Y).      % X=jim
```

Built-in Prolog Functions

- Built-in functions used in this code:
- asserta – place a fact temporarily into the db
- repeat – continues searching (provided ';' is entered)
- retractall – to remove facts which you have asserted (using asserta).

Prolog Construct

Cut (!) – a zero-argument predicate which always succeeds and causes searching to stop.