CSE 3302
Programming Languages

Logic Programming: Prolog (II)

Chengkai Li
Spring 2008

SWI-Prolog

Resources

- Download:
  http://www.swi-prolog.org/dl-stable.html
- Documentation:
  (You don't necessarily need to read. But good for reference when you have questions.)

  http://www.swi-prolog.org/dl-doc.html
  Menu "Help -> Online Manual" (HTML files in directory "doc")

Query Prompt

- query prompt

?- (Enter goals after "?-")
Example: ?- help(help).

- Load a file with clauses

```
?-[swi('myprogram/example.pl')].
```

Or

```
?-[swi('myprogram/example')].
```

(`myprogram` must be a subdirectory in the swi-prolog program directory)


```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```


User Interaction

```
?- parent(bob,sam).    (a query must end with . )

true                   (can be proved)
?- parent(bob,jill).

fail                   (cannot prove)
?- parent(bill,X),
|  father(X,sam)
|.                     (user can use multiple lines to write a query )

fail
?- parent(X, sam).
X = jill ;             (user typed ; to ask for more answers.)
X = bob
```

Debugging

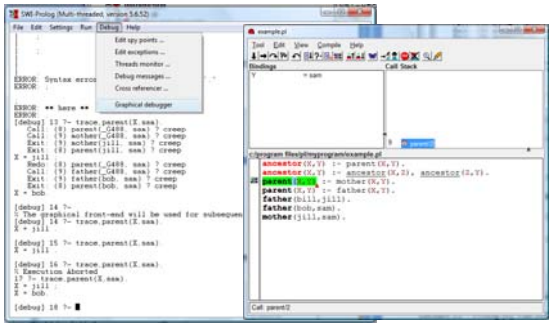
```
?- trace, parent(X, sam).
 Call: (8) parent(_G494, sam) ? creep
   Call: (9) mother(_G494, sam) ? creep
   Exit: (9) mother(jill, sam) ? creep
   Exit: (8) parent(jill, sam) ? creep

X = jill ;
   Redo: (8) parent(_G494, sam) ? creep
   Call: (9) father(_G494, sam) ? creep
   Exit: (9) father(bob, sam) ? creep
   Exit: (8) parent(bob, sam) ? creep

X = bob
```

More details in section 2.9 and 4.2.8 of the manual

Graphical Debugger



Prolog Syntax

Basic Syntax

```
<clause>      ::= <fact> | <rule>
<fact>        ::= <term> .
<rule>        ::= <term> :- <termlist> .
<termlist>    ::= <term> | <term> , <termlist>


<term>        ::= <variable> | <constant> | <compound-term>
<constant>          ::= <number> | <atom>
<compound-term>     ::= <atom> ( <termlist> )
```


Prolog syntax

- :- for ←
  , for and
- Uppercase: variable
  Lowercase: other names (constants, atom (i.e., name of predicate))
- Built-in predicates:

  read, write, nl (newline)

  =,is,<,>,=<,>=,/,*,+,-,mod,div
  (Note it is =<, not <=)

Arithmetic

- Arithmetic operation can use prefix or infix notations.
  +(3,4)
  3+4
- Value is not immediately evaluated.
  ?- write(3+5).
  ?- X is 3+5. (is a predicate that evaluates 3+5)
  x=7.
  ?- 3+4 = 4+3. (these are two different terms)
  fail.
  ?- X is 3+4, Y is 4+3, X = Y. (unification)
  X=7,
  Y=7.

Unification

- The semantics of = is determined by unification, i.e., = forces unification.

?- me = me.
true.
?- me = you.
fail.
?- me = X.
X = me.
?- f(a,X) = f(Y,b).
X = b,
Y = a.
?- f(X) = g(X).
fail.

(See unification algorithm in Page 556)

1. Constant unifies only with itself.

2. Variable unifies with anything, and becomes instantiated to that thing.

3. Two predicates unifies if they have the same name and the same number of arguments, and their arguments unify.

Unification for List Operations

```
?- [H|T]=[1,2,3].
H = 1,
T = [2,3]

?- [H1,H2|T]=[1,2,3].
H1 = 1,
H2 = 2,
T = [3]

?- [H1,H2,H3|T]=[1,2,3,4,5].
H1 = 1,
H2 = 2,
H3 = 3,
T = [4,5]
```


List Operations

- Concatenation:

```
?- X = [0,1|[2,3,4]].
X = [0,1,2,3,4]
```

- Get elements, or tail :

```
?- [H1,H2|[3,4]] = [0,1|[2,3,4]]
```

What do we get?
fail.

```
?- [H1,H2|[3,4]] = [0,[1,2],3,4]
```

What do we get?
H1=0,
H2=[1,2].

Define List Operation Predicates

- cons(X,Y,L) :- L = [X|Y].

?- cons (0,[1,2,3],A).
?- cons (X,Y,[1,2,3]).

- Rewrite cons:

cons(X,Y, [X|Y]).

Define List Operation Predicates

- append(X,Y,Z) :- X = [ ], Y=Z.
  append(X,Y,Z) :- X = [A|B], Z=[A|W], append(B,Y,W).

- Another definition
  append([ ],Y,Y).
  append([A|B], Y, [A|W]) :- append(B,Y,W).

?- append(X, Y, [1,2]).

- reverse([],[]).
  reverse([H|T], L) :- reverse(T,L1), append(L1, [H], L).

Prolog's Search Strategy

Resolution and Unification

- Order matters:
  - The order to resolve subgoals.
  - The order to use clauses to resolve subgoals.
- Thus programmers must know the orders used by the language implementations, in order to write efficient or even correct program. (Search Strategies)

Prolog's Strategy

- Depth-first search
  - The order to resolve subgoals. (left to right)
  - The order to use clauses to resolve subgoals. (top to bottom)
- **Backtrack:** try another clause when it fails.

Example 1

- Facts:

```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Queries:

```
?- ancestor(bill,sam).
```


Example 1

- Facts:

```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Queries:

```
        :- ancestor(bill,sam).
ancestor(X1,Y1) :- ancestor(X1,Z1), parent(Z1,Y1).
```

X1= bill, Y1=sam

```
:- ancestor(bill,Z1), parent(Z1,sam).
```


Example 1

- Facts:

```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Queries:

```
        :- ancestor(bill,Z1), parent(Z1,sam).
ancestor(X2,Y2) :- ancestor(X2,Z2), parent(Z2,Y2).
```

X2= bill, Y2=Z1

```
:- ancestor(bill,Z2), parent(Z2,Z1), parent(Z1,sam).
```

Example 1

- Facts:

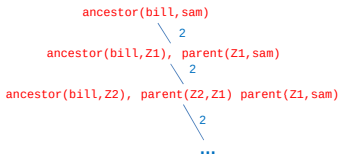
```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Queries:

```
:- ancestor(bill,Z2), parent(Z2,Z1), parent(Z1,sam).
ancestor(X3,Y3) :- ancestor(X3,Z3), parent(Z3,Y3).
```

...

Example 1



Resulting in an infinite loop.
Original order was bad

Example 2

- Facts:

```
ancestor(X,Y) :- parent(X,Y).                    1
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).     2
parent(X,Y) :- mother(X,Y).                      3
parent(X,Y) :- father(X,Y).                      4
father(bill,jill).                               5
mother(jill,sam).                                6
father(bob,sam).                                 7
```

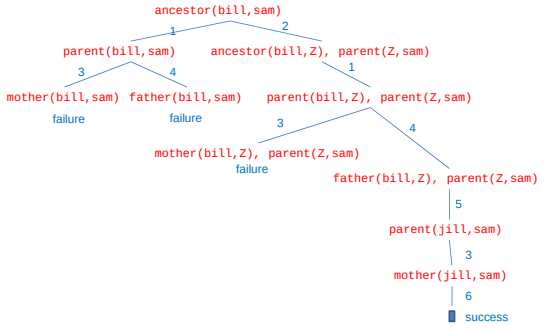
- Queries:

```
?- ancestor(bill,sam).
```

What will happen?

Note that we change the order of the first two clauses in facts.

Example 2



Example 3

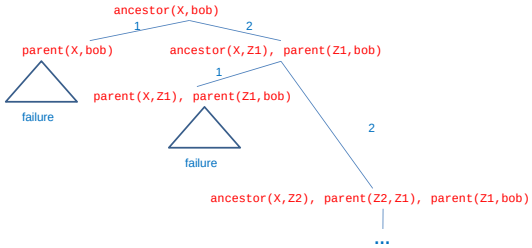
- Facts:

```
ancestor(X,Y) :- parent(X,Y).                    1
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).     2
parent(X,Y) :- mother(X,Y).                      3
parent(X,Y) :- father(X,Y).                      4
father(bill,jill).                               5
mother(jill,sam).                                6
father(bob,sam).                                 7
```

- Queries:

```
?- ancestor(X,bob).
```


Example 3



Resulting in an infinite loop.
Original order was bad

Example 4

- Facts:

```
ancestor(X,Y) :- parent(X,Y).                    1
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).     2
parent(X,Y) :- mother(X,Y).                      3
parent(X,Y) :- father(X,Y).                      4
father(bill,jill).                               5
mother(jill,sam).                                6
father(bob,sam).                                 7
```

- Queries:

```
?- ancestor(X,bob).
```

What will happen?

Note that we change the order of the two subgoals in clause (2).