


CSE 3302
Programming Languages

Logic Programming:
Prolog (II)


Chengkai Li
Fall 2007

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 1



SWI-Prolog


Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 2



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")
?-help(help).

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 3




Query Prompt

- query prompt
?- (Enter goals after "?-")
Example: ?- help(help).
- Load a file with facts
?-[swi('myprogram/test.pl')].
or
?-[swi('myprogram/test')].

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

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 4



User Interaction

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


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

No (cannot prove)
?- parent(bill,X),
| parent(X,sam)
|.

(user can use multiple lines to write a query, using |)

No
?- parent(X, sam).
X = jill ; (user typed ; to ask for more answers.)
X = bob

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 5



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

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007 6



Prolog Syntax

Lecture 22 – Prolog (II), Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

7

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> )

```

Lecture 22 – Prolog (II), Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

8

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 <=)

Lecture 21 – Prolog, Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

9

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)
- `?- 3+4 = 4+3.` (these are two different terms)
No.
`?- X is 3+4, Y is 4+3, X = Y.` (unification)
`X=7,`
`Y=7`

Lecture 22 – Prolog (II), Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

10

Unification



- The semantics of = is determined by unification, i.e., = forces unification.
(See unification algorithm in Page 556)

```

?- me = me.
Yes
?- me = you.
No
?- me = X.
X = me
?- f(a,X) = f(Y,b).
X = b,
Y = a
?- f(X) = g(X).
No

```

Lecture 22 – Prolog (II), Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

11

List



```

?- [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]

```

Lecture 22 – Prolog (II), Fall 2007

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007

12

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?

Define List Operation Predicates



- $\text{cons}(X,Y,L) :- L = [X|Y]$.

?- $\text{cons}(0,[1,2,3],A)$.

?- $\text{cons}(X,Y,[1,2,3])$.

- Rewrite cons:

$\text{cons}(X,Y, [X|Y])$.

Define List Operation Predicates



- $\text{append}(X,Y,Z) :- X = [], Y=Z$.

$\text{append}(X,Y,Z) :- X = [A|B], Z=[A|W], \text{append}(B,Y,W)$.

- Another definition

$\text{append}([],Y,Y)$.

$\text{append}([A|B], Y, [A|W]) :- \text{append}(B,Y,W)$.

?- $\text{append}(X, Y, [1,2])$.

- $\text{reverse}([],[])$.

$\text{reverse}([H|T], L) :- \text{reverse}(T,L1), \text{append}(L1, [H], L)$.

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.

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 19

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).
```

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 20

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).
```

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 21

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).
```

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 22

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).
                        ...
```

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 23


Example 1

```
ancestor(bill,sam)
  /
 2
ancestor(bill,Z1), parent(Z1,sam)
  /
 2
ancestor(bill,Z2), parent(Z2,Z1), parent(Z1,sam)
  /
 2
...
```

Resulting in an infinite loop.
Original order was bad

Lecture 22 – Prolog (II), Fall 2007 CSE3302 Programming Languages, UT-Arlington ©Chengkai LI, 2007 24

Example 2

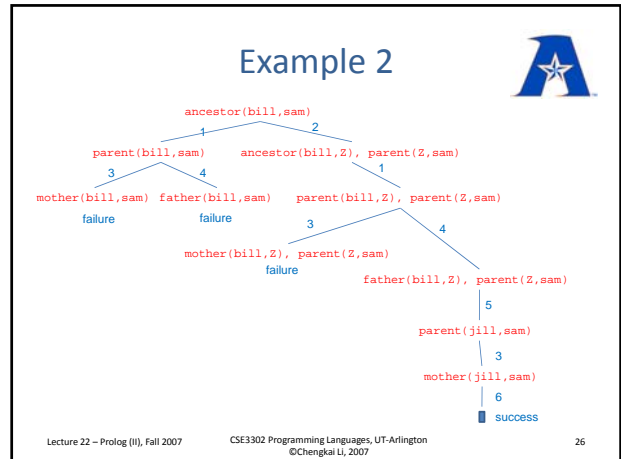


- Facts:
 - 1 ancestor(X,Y) :- parent(X,Y).
 - 2 ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
 - 3 parent(X,Y) :- mother(X,Y).
 - 4 parent(X,Y) :- father(X,Y).
 - 5 father(bill,jill).
 - 6 mother(jill,sam).
 - 7 father(bob,sam).
- Queries:



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

What will happen?
 Note that we change the order of the first two clauses in facts.

Lecture 22 – Prolog (II), Fall 2007
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007
25



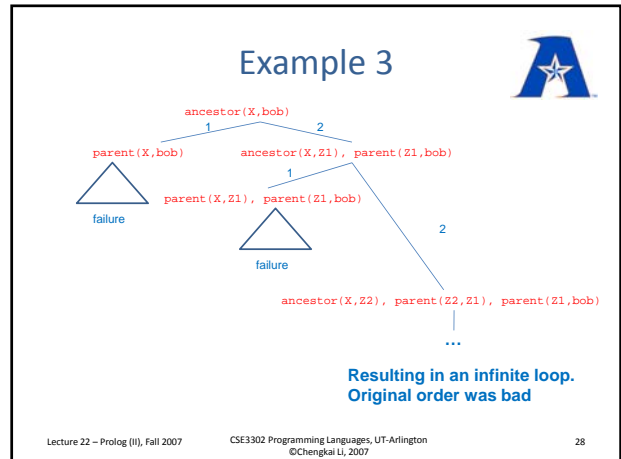
Example 3




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


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

Lecture 22 – Prolog (II), Fall 2007
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007
27



Example 4




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


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

What will happen?
 Note that we change the order of the two subgoals in clause (2).

Lecture 22 – Prolog (II), Fall 2007
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007
29



Loops and Control:

fail and cut (!)

Lecture 22 – Prolog (II), Fall 2007
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2007
30

fail



- Loops:
Enforce backtrack even when an answer is found (using built-in predicate **fail**)

Example



```
printpieces(L) :- append(X,Y,L),
                 write(X),
                 write(Y),
                 nl,
                 fail.
```

```
?- printpieces([1,2]).
[[1,2]
 [1][2]
 [1,2][
 No
```

Example

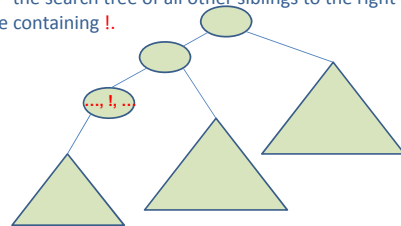


- num(0).
num(X) :- num(Y), X is Y+1.
?- num(X).
- writenum(I,J) :- num(X),
 I =< X,
 X =< J,
 write(X),
 nl,
 fail.
?- writenum(1,10).

cut



- Cut (using built-in predicate **!**) branches in the search tree (to avoid infinite loop).
- “prunes” the search tree of all other siblings to the right of the node containing **!**.



Example



- writenum(I,J) :- num(X),
 I =< X,
 X =< J,
 write(X),
 nl,
 X = J, !,
 fail.
?- writenum(1,10).