CS206 Lecture 11

Prolog Built-in Functions

G. Sivakumar
Computer Science Department
IIT Bombay
siva@iitb.ac.in
http://www.cse.iitb.ac.in/~siva

Thu, Jan 23, 2003

Plan for Lecture 11

- More Prolog Examples (Lists and Numbers)
- Built-in Functions

Numbers

Simple **inductive** definitions.

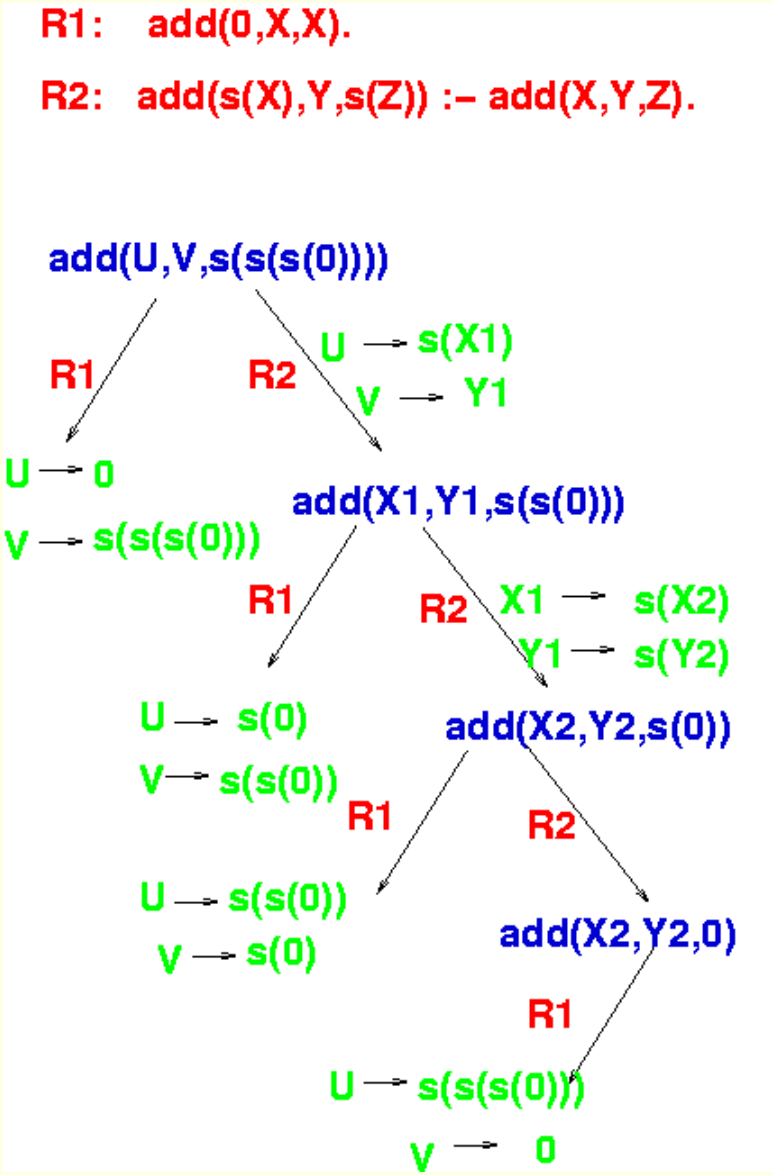
```
natnum(0).
natnum(s(X)) :- natnum(X).

add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,X,0).
mult(s(X),Y,Z) :- mult(X,Y,Z1), add(Y,Z1,Z).

power(X,0,s(0)).
power(X,s(Y),Ans) :- power(X,Y,A), mult(A,X,Ans).
```

Execution Trace

Goal with **multiple** answers enumerated.

```
| ?- add(X,Y,s(s(s(0)))).

X = 0
Y = s(s(s(0))) ? ;

X = s(0)
Y = s(s(0)) ? ;

X = s(s(0))
Y = s(0) ? ;

X = s(s(s(0)))
Y = 0 ? ;

no

| ?-
```

Prolog's Compuation Tree



Another Example

```
| ?- mult(X,Y,s(s(0))).

X = s(0)
Y = s(s(0)) ? ;

X = s(s(0))
Y = s(0) ? ;

Fatal Error: global stack overflow
(size: 8193 Kb, environment variable used: GLOBALSZ)
```

What happened?

Infinite Computation Trees

R1: add(0,X,X).
R2: add(s(X),Y,s(Z)) :– add(X,Y,Z).
R3: mult(0,Y,0).
R4: mult(s(X),Y,Z) :– mult(X,Y,N), add(Y,N,Z).

mult(U,V,s(s(0)))

R4 | U –> s(X1)

mult(X1,V,N1), add(V,N1,s(s(0)))

R3 X1 –> 0, N1 –> 0 | R4 X1 –> s(X2)

add(V,0,s(s(0)))

R2

R2

U → s(0)

V → s(s(0))

mult(X2,V,N2), add(V,N2,N1), add(V,N1,s(s(0)))

R3 X2 –> 0, N2 –> 0 | R4

add(V,0,N1),add(V,N1,s(s(0)))

Infinite subtree for this goal!

Only 1 solution. Rest Fail!!

Never reach here if DFS!

Back to Satan-Cantor Puzzle

How to implement the following **fair enumerations**?

```
| ?- pairs(Ans).

Ans = [0,0] ? ;
Ans = [0,s(0)] ? ;
Ans = [s(0),0] ? ;
Ans = [0,s(s(0))] ? ;
Ans = [s(0),s(0)] ? ;
Ans = [s(s(0)),0] ? ;
Ans = [0,s(s(s(0)))] ? ;
Ans = [s(0),s(s(0))] ? ;
Ans = [s(s(0)),s(0)] ?

yes
```

Triples

```
| ?- triples(Ans).

Ans = [0,0,0] ? ;
Ans = [0,0,s(0)] ? ;
Ans = [0,s(0),0] ? ;
Ans = [s(0),0,0] ? ;
Ans = [0,0,s(s(0))] ? ;
Ans = [0,s(0),s(0)] ? ;
Ans = [0,s(s(0)),0] ? ;
Ans = [s(0),0,s(0)] ? ;
Ans = [s(0),s(0),0] ? ;
Ans = [s(s(0)),0,0] ? ;
Ans = [0,0,s(s(s(0)))] ?

yes
```

Solution

One rule is enough!

```
pairs([X,Y]) :- natnum(N), add(X, Y, N).

triples([X,Y,Z]) :- natnum(N),
                    add(X,N1,N),
                    add(Y,Z,N1).
```

Trace these goals and understand their computational trees.

Bounded Enumeration

Part of next assignment is to implement the following.

```
| ?- enum_from_to(s(0),s(s(s(0))),Ans).

Ans = s(0) ? ;
Ans = s(s(0)) ? ;
Ans = s(s(s(0))) ? ;

no
| ?- pairsupto(s(s(0)),Ans).

Ans = [0,0] ? ;
Ans = [0,s(0)] ? ;
Ans = [s(0),0] ? ;
Ans = [0,s(s(0))] ? ;
Ans = [s(0),s(0)] ? ;
Ans = [s(s(0)),0] ? ;

no
```

Must Stop after enumerating all answers.

Some List Operations

Appending and Reversal.

```
app([],X,X).
app([U | V], X, [U | W]) :- app(V,X,W).

rev([],[]).
rev([A|X], Ans) :- rev(X, A1),
                   app(A1, [A], Ans).
```

Permutation

```
| ?- perm([a,b,c],Ans).

Ans = [a,b,c] ? ;

Ans = [b,a,c] ? ;

Ans = [b,c,a] ? ;

Ans = [a,c,b] ? ;

Ans = [c,a,b] ? ;

Ans = [c,b,a] ? ;

no
```

One solution given in next slide. It does not halt when invoked as perm(Ans,[a,b,c]).
Fix that!

Permutation Solution

```
/*  erase(X,Y,Z)  means
 Z is Y with one occurrence of X removed */

erase(X,[X|Y],Y).
erase(X,[Y|Z],[Y|Z1]) :- erase(X,Z,Z1).

/* perm(L1,L2) means
   L2 is a permutation of L1 */
perm([],[]).
perm([A | R], X) :- perm(R,X1),
                    erase(A,X,X1).
```

Built-in Functions

Prolog provides built-in functions for several reasons although it is Turing-complete even without them.

1. Efficiency

   Why do arithmetic in *unary* notation, when we have fast arithmetic-logic units on computers?

2. Convenience

   Rather than write every function ourselves tediously, some standard ones are provided.

3. Expressive power

   Some extra (not pure logic) constructs are provided to make it easier to write programs.

Arithmetic Functions

Prolog does not interpret $+$, $*$ and so on specially except in special predicates.

Thus, if we write naively

```
add(X,Y,Z) :- Z = X + Y.
```

and give the query `add(0,0,Ans)` we will get back

```
Z = 0 + 0
```

as though $+$ was just any infix operator.

To force evaluation this must be written using `is`.

is Predicate

If we write

```
add(X,Y,Z) :- Z is X + Y.
```

Now + is actually evaluated and for the query `add(2,3,Z)` we do get

```
Z = 5
```

as the only answer.

Caution: Queries must use only numbers as first two arguments.

calling `add(X,2,5)` will give a system error!

Other Arithmetic Functions

1. X - Y (subtraction)
2. X * Y (multiplication)
3. X / Y (division)
4. X mod Y (remainder)
5. - X (unary minus)
6. floor, exp, square, sin, ...

They must be used with the *is* predicate as follows: `Z is X * Y`. X, Y must be arithmetic expressions evaluating to a number.

Comparison Operators

When X and Y are bound to numbers the following built in predicates work as expected.

Caution: Both X and Y must be ground

1. X < Y
2. X > Y
3. X =< Y
4. X >= Y
5. X =
   = Y (not equal)

Typing Operators

The following built in functions are also very useful.

- integer(X)

  tests if X is an integer. Similarly,

- real(X), float(X), number(X)
- var(X)

  checks if X is currently an unbound variable. Similarly,

- nonvar(X), atomic(X), structure(X)

Example

Counters using Built-in Functions.

```
/* simple counter. goes on for ever 0 1 2 3 4 ...  */

ctr(0).
ctr(X) :- ctr(Y), X is Y + 1.

/* bounded counter.  enumerates 0,1,2,3,...,N  */

bounded_ctr(X,Bound) :- ct_des(X1,Bound),
                        X is Bound - X1.
ct_des(N,N).
ct_des(X,N) :- N > 0, N1 is N - 1, ct_des(X,N1).
```

Insertion Sort of a List of Numbers

```
/* To sort a list, sort its tail, then
   insert its head in the right position. */

isort([Head|Tail],Result) :-
    isort(Tail,SortedTail),
    insert(Head,SortedTail,Result).
isort([],[]).
/* To insert an item into the correct position
   in a sorted list: Put it at the beginning
   if it should precede the first element;
   otherwise go down the list until a position
   is found where this is the case. */

insert(X,[Y|Tail],[X,Y|Tail]) :-
    X =< Y.
insert(X,[Y|Tail],[Y|Z]) :-
    X > Y,    insert(X,Tail,Z).
insert(X,[],[X]).
```

Problems for Next Lecture

Think about the following problems.

1. Coin-Change

   Given unlimited coins of some denominations (eg. 25,10,1), how to make change for some amount (58)?

2. Knight-Walk

   Given a starting square (i,j) on chessboard, how to generate all walks of length n.

3. P235 numbers

   Find all numbers (up to a bound) which have only 2, 3 or 5 as prime factors. Example such numbers: 36, 50.