
INTRODUCTION TO LOGIC PROGRAMMING

(Lecture 3)

- SRIVIDYA KONA

Built-in Arithmetic

Arithmetic terms:

- a number is an arithmetic term,
- if f is an n -ary arithmetic functor and X_1, \dots, X_n are arithmetic terms then $f(X_1, \dots, X_n)$ is an arithmetic term.
- Arithmetic functors: $+$, $-$, $*$, $/$ (float quotient), $//$ (integer quotient), mod , and more.
- Examples:
 - $(3*X+Y)/Z$ is evaluated if X , Y and Z are arithmetic terms, otherwise it will raise an error.
 - $a+3*X$ raises an error (because a is not an arithmetic term).

Built-in Arithmetic

Built-in arithmetic predicates:

- the usual $<$, $>$, $=<$, $>=$, $:=$ (arithmetic equal), \neq (arithmetic not equal). Both arguments are evaluated and their results are compared
- Z is X . % 'is' operator
 where X (which must be an arithmetic term) is evaluated and result is unified with Z .
- *Examples:* let X and Y be bound to 3 and 4, respectively, and Z be a free variable:
 - $Y < X+1$, X is $Y+1$, $X := Y$ fail (the system will backtrack).
 - $Y < a+1$, X is $Z+1$, $X := f(a)$ error (abort).

'\+' operator (Negation as Failure)

*“p holds if a and b hold, OR
if a does not hold and c holds”*

$p \text{ :- } a, b.$

$p \text{ :- } \text{\textbackslash+ } a, c.$

If-then-else

Built-in predicate for the if-then-else construct

if A then B else C is written as

(A -> B ; C).

Example:

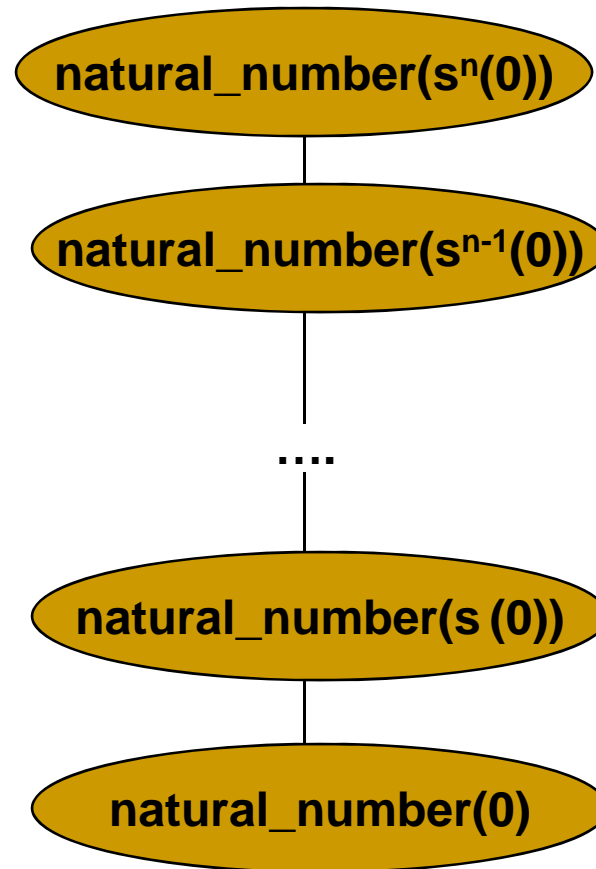
maxmin(X, Y, Max, Min) :-

(X =< Y -> Max = Y, Min = X
; Max = X, Min = Y).

Successor notation – natural numbers

- Representation of numbers in a logical manner using structure/functor
- How is the set of natural numbers defined? (using Peano arithmetic)
 - 0 (zero) is a natural number,
 - if X is natural number, then so is $X+1$.
- $X+1$ can be represented by a functor $s(X)$
- $\text{natural}(X)$ is true if X is a natural number.
 $\text{natural}(0)$.
 $\text{natural}(s(X)) \text{ :- natural}(X)$.

Successor notation – natural numbers



Successor notation – plus

$\text{plus}(X, Y, Z) :- Z \text{ is } X + Y$

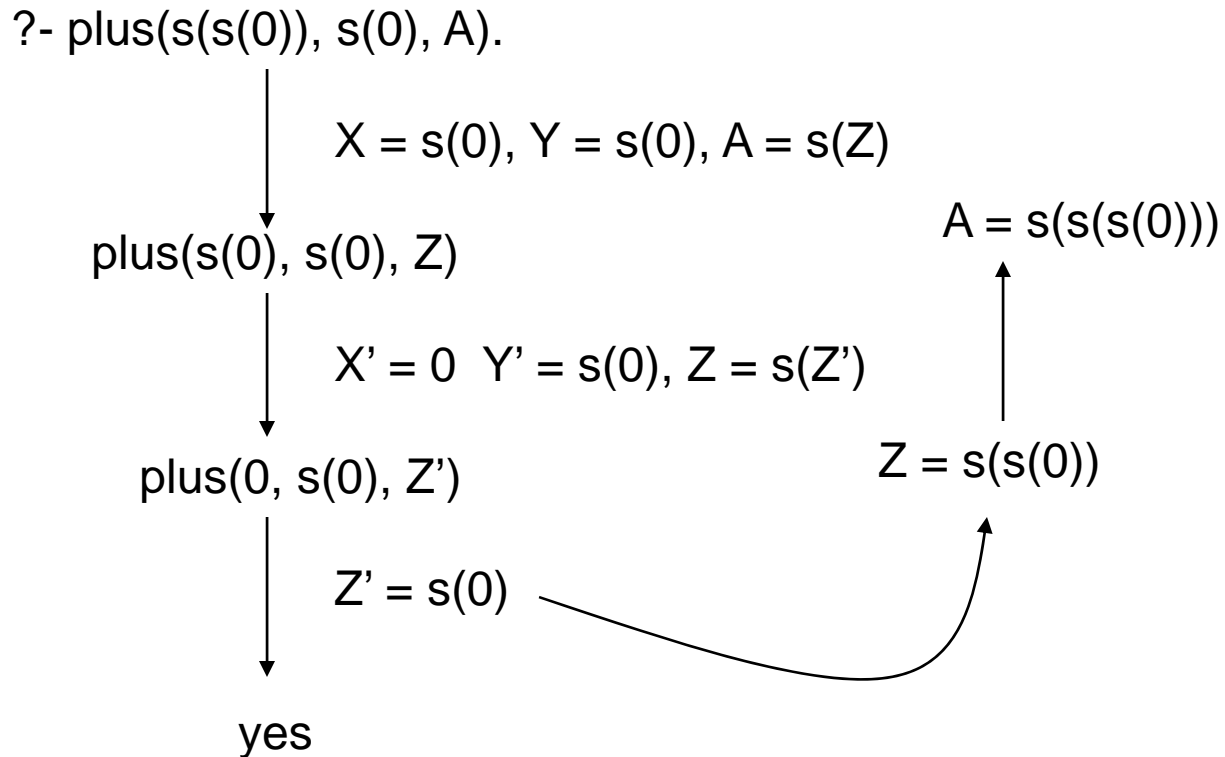
- Only works in one direction (X and Y are arithmetic terms).
- We have lost the recursive structure of the numbers.
- Axioms for plus from first principles (using successor notation):
 - $0 + Y = Y$
 - $(X + 1) + Y = (X + Y) + 1$

$\text{plus}(X, Y, Z)$ is true if the sum of the natural number X and the natural number Y is the natural number Z .

$\text{plus}(0, X, X)$.

$\text{plus}(s(X), Y, s(Z)) :- \text{plus}(X, Y, Z)$.

Successor notation – plus



Successor notation – times

- Axioms for times from first principles (using successor notation):
 - $0 * Y = 0$
 - $(X+1) * Y = (X * Y) + Y$
- $\text{times}(X, Y, Z)$ is true if the product of the natural number X and the natural number Y is the natural number Z .

$\text{times}(0, _Y, 0)$.

$\text{times}(s(X), Y, s(Z)) \text{ :- } \text{times}(X, Y, XY), \text{ plus}(Y, XY, Z)$.

Successor notation - factorial

- Factorial using Prolog arithmetic:

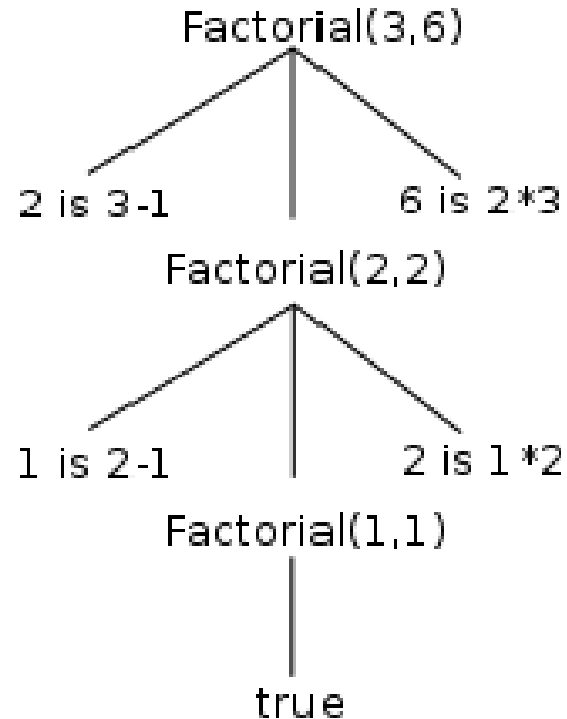
`factorial(0,1).`

`factorial(N,F) :- N > 0, N1 is N-1,
factorial(N1,F1),
F is F1*N.`

- Factorial using successor notation:

`factorial(0, s(0)).`

`factorial(s(N),F) :- factorial(N,F1),
times(s(N),F1,F).`



Successor notation - fibonacci

- Fibonacci using Prolog arithmetic:

```
fib(0,1).
```

```
fib(1,1).
```

```
fib(N, R) :- N1 is N-1, N2 is N-2,  
            fib(N1, R1), fib(N2, R2),  
            R is R1+R2.
```

- Fibonacci using successor notation:

```
fib(0,s(0)).
```

```
fib(s(0),s(0)).
```

```
fib(s(s(X)), Z) :- fib(X, X1),  
                  fib(s(X), X2),  
                  plus(X1, X2, Z).
```

'select' predicate

- `select(X, L, NewList)` – `NewList` is obtained by removing one occurrence of `X` from `L`. If `X` is not found, it fails.

`select(X, [X|T], T).`

`select(X, [Y|T], [Y|R]) :- select(X, T, R).`

- `delete(X, L, NewList)` – `NewList` is obtained by removing all occurrence of `X` from `L`. If `X` is not found `NewList` is same as `L`

`del(_, [], []).`

`del(X, [X|T], L) :- del(X,T,L).`

`del(X, [H|T1], [H|T2]) :- X \= H, del(X,T1,T2).`

'permutation' predicate

- `permutation(X, Xp)` - `Xp` is the permutation of list `X`

`permutation([],[]).`

`permutation(L, [H|T]) :- select(H, L, R),
permutation(R, T).`

- Generates all possible permutations of list `X`

Permutation Sort

- `permsort(X, Y)` – `Y` is the ordered permutation of `X`

```
permsort(X, Y) :- permutation(X, Y),  
                  ordered(Y).
```

```
ordered([ ]).
```

```
ordered([X]).
```

```
ordered([A, B | T]) :- A =< B, ordered([B|T]).
```

- Naïve sorting program, uses generate-and-test paradigm

Insertion Sort

- `insertsort(X, Y)` – `Y` is the sorted permutation of `X`
(sorted using insertion sort)
- The first element is removed from the list, and remaining list is recursively sorted. Then the first element is inserted preserving the sorted order of the list

```
insertsort([ ], [ ]).
```

```
insertsort([H|T], Y) :- insertsort(T, Z), insert(H, Z, Y).
```

```
insert(X, [ ], [X]).
```

```
insert(X, [Y|T], [Y|Z]) :- X > Y, insert(X, T, Z).
```

```
insert(X, [Y|T], [X,Y|T]) :- X =< Y.
```

Quicksort

- `qsort(L, R)` – R is the sorted permutation of L
(using quick sort)
- List is split into two by choosing a pivot element
 - one list containing elements smaller than the chosen pivot
 - other list containing elements larger than the chosen pivot
- Then the split lists are recursively sort and their results are appended

```
qsort([ ], [ ]).
```

```
qsort([X|L], R) :- partition(L, X, Littles, Bigs),  
                  qsort(Littles, Ls),  
                  qsort(Bigs, Bs),  
                  append(Ls, [X|Bs], R).
```

```
partition([ ], _X, [ ], [ ]).
```

```
partition([X|Xs], Y, [X|Ls], Bs) :- X =< Y, partition(Xs, Y, Ls, Bs).
```

```
partition([X|Xs], Y, Ls, [X|Bs]) :- X > Y, partition(Xs, Y, Ls, Bs).
```

SEND + MORE = MONEY Puzzle

- Each letter represents a unique digit from 0 to 9.
- Two letters cannot represent the same digit.
- What digit each letter represents to satisfy the simple equation below?

Solution:

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

$$\begin{array}{r} 9 5 6 7 \\ + 1 0 8 5 \\ \hline 1 0 6 5 2 \end{array}$$

SEND + MORE = MONEY Puzzle

```
solve([S,E,N,D,M,O,R,Y]) :- M is 1,  
    select(D, [0,2,3,4,5,6,7,8,9], R1),  
    select(E, R1, R2),  
    Y is (D+E) mod 10,  
    C1 is (D+E) // 10,  
    select(Y, R2, R3),  
    select(N, R3, R4),  
    select(R, R4, R5),  
    E is (N+R+C1) mod 10,  
    C2 is (N+R+C1) // 10,  
    select(O, R5, R6),  
    N is (E+O+C2) mod 10,  
    C3 is (E+O+C2) // 10,  
    select(S, R6, R7),  
    O is (S+M+C3) mod 10,  
    M is (S+M+C3) // 10.
```

	S	E	N	D
+	M	O	R	E
<hr/>				
M	O	N	E	Y

Conclusions

- Imperative Vs Declarative Programming
- Logic & History of Logic Programming
- Basic Prolog Syntax and Semantics
- Unification
- Database Programming
- Recursion & Lists
- Accumulator trick for tail-recursive programs
- Arithmetic functors and predicates
- Successor Notation
- Sorting & Cryptarithmic puzzle