CS 603: Programming Languages

Lecture 26

Spring 2004

Department of Computer Science
University of Alabama
Joel Jones

Ancestry Example

?- parent(X , ann), parent(X , joe).
X = mary;
X = mike
yes

?- grandparent(sue, Y).
Y = ann;
Y = joe
yes

program

```
mother(mary, ann).
mother(mary, joe).
mother(sue, mary).
father(mike, ann).
father(mike, joe).

parent( X ,Y ) :– mother( X ,Y ).
parent( X ,Y ) :– father( X ,Y ).
grandparent( X , Z ) :– parent( X ,Y ),parent(Y, Z ).
```

Tracing Queries

Example Query #1:

query ?- mother(X,joe). X=X,Y=joe binding

mother(mary,ann). /* fails */

mother(mary,joe). /* succeeds */

program

```
mother(mary, ann).
mother(mary, joe).
mother(sue, mary).
father(mike, ann).
father(mike, joe).

parent( X ,Y ) :– mother( X ,Y ).
parent( X ,Y ) :– father( X ,Y ).
```

Example Query #2:

?- parent(X,joe). X=X,Y=joe

parent(X,Y) :- mother(X,joe).

?- mother(X,joe). X=X,Y=joe

mother(mary,ann). /* fails */

mother(mary,joe). /* succeeds */

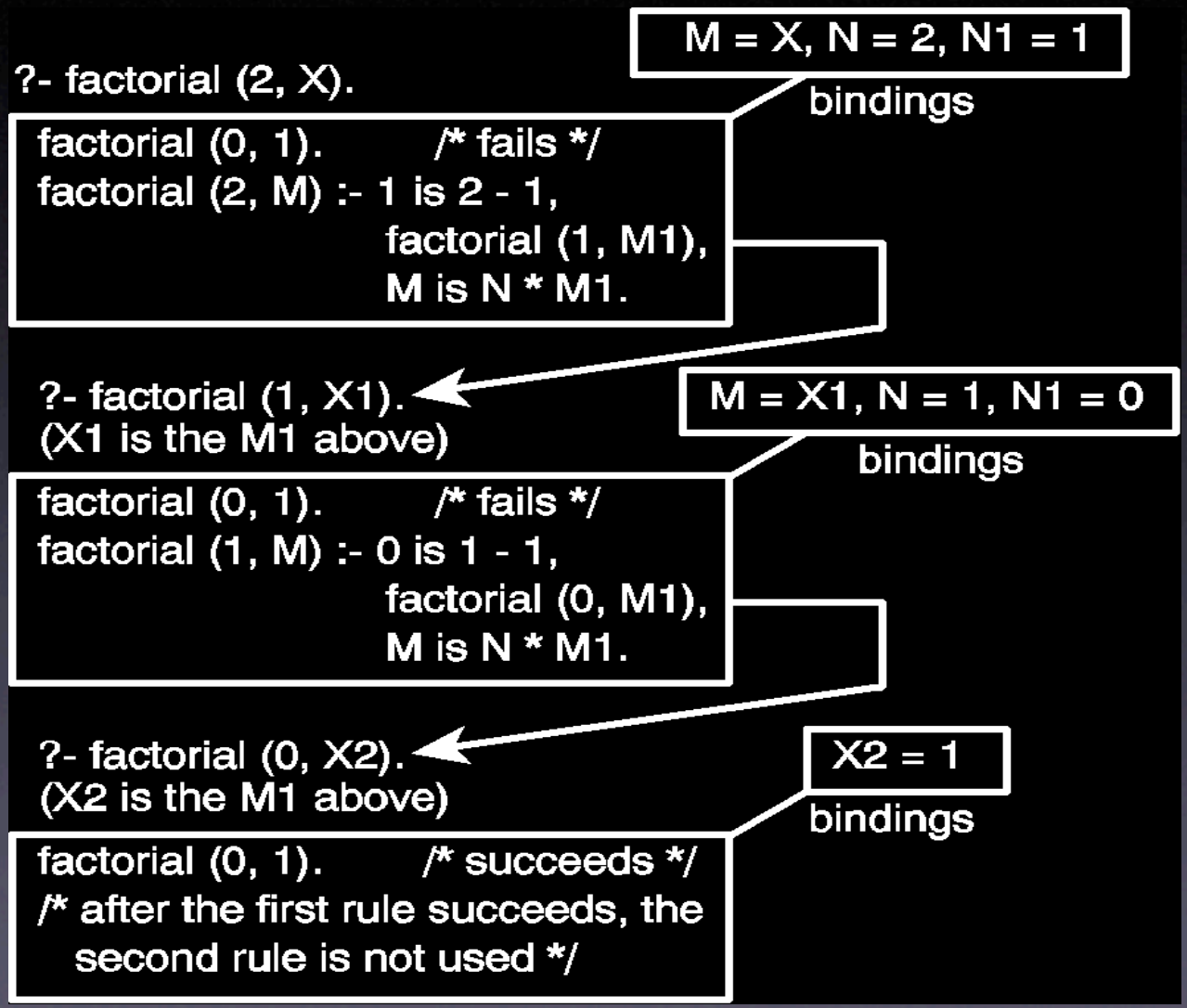
Tracing exercise

```
/* specification of factorial n! */
factorial(0,1).
factorial(N, M):– N1 is N – 1, factorial (N1, M1), M is N*M1.
```

Pair Up:

- Trace factorial(2,X)

Solution


M = X, N = 2, N1 = 1
bindings
?- factorial (2, X).
factorial (0, 1). /* fails */
factorial (2, M) :- 1 is 2 - 1,
factorial (1, M1),
M is N * M1.
?- factorial (1, X1).
(X1 is the M1 above)
M = X1, N = 1, N1 = 0
bindings
factorial (0, 1). /* fails */
factorial (1, M) :- 0 is 1 - 1,
factorial (0, M1),
M is N * M1.
?- factorial (0, X2).
(X2 is the M1 above)
X2 = 1
bindings
factorial (0, 1). /* succeeds */
/* after the first rule succeeds, the
second rule is not used */

Recursion in Prolog

- trivial, or boundary cases
- 'general' cases where the solution is constructed from solutions of (simpler) version of the original problem itself.
- What is the length of a list ?
- THINK:
  The length of a list, [e | Tail], is 1 + the length of Tail
- What is the boundary condition?
  - The list [ ] is the boundary. The length of [ ] is 0.

Recursion

- Where do we store the value of the length?—an accumulator

```
mylength( [ ], 0).
mylength( [X | Y], N):–mylength(Y, Nx), N is Nx+1.

? – mylength( [1, 7, 9], X ).
X = 3
? - mylength(jim, X ).
no
? - mylength(Jim, X ).
Jim = [ ]
X = 0
```

Recursion

```
mymember( X , [X | _ ] ).
mymember( X , [ _ | Z ] ) :-  mymember( X , Z ).
% equivalently:  However swipl will give a warning
% Singleton variables :Y W
mymember( X , [X | Y] ).
mymember( X , [W | Z ] ) :- mymember( X , Z ).

1?–mymember(a, [b, c, 6] ).
no
2? – mymember(a, [b, a, 6] ).
yes
3? – mymember( X , [b, c, 6] ).
X = b;
X = c;
X = 6;
no
```

Appending Lists I

- The Problem: Define a relation append(X,Y,Z) to mean that X appended to Y yields Z
-
- The Program:

```
append([],Y,Y).
append([H|X],Y, [H|Z]) :- append(X,Y,Z).
```

Watch it work:

```
 ?- [append].
 ?- append([1,2,3,4,5],[a,b,c,d],Z).
Z = [1,2,3,4,5,a,b,c,d]?;
no
 ?- append(X,Y,[1,2,3]).
X = [] Y = [1,2,3]?;
X = [1] Y = [2,3]?;
X = [1,2] Y = [3]?;
X = [1,2,3] Y = []?;
no
 ?-
 ?- append([1,2,3],Y,Z).
Z = [1,2,3|Y]
```

Order Dependency

- Previous Program

```
mylength( [ ], 0).
mylength( [X | Y], N):–mylength(Y, Nx), N is Nx+1.
```

- The Program:

```
llength([],0).

llength([X|Z],N) :- N is M + 1, llength(Z,M).
?- [length2].
?- llength([a,b,c,d],M).
uncaught exception: error(instantiation_error,(is)/2)
```