Derivation Tree (6A)

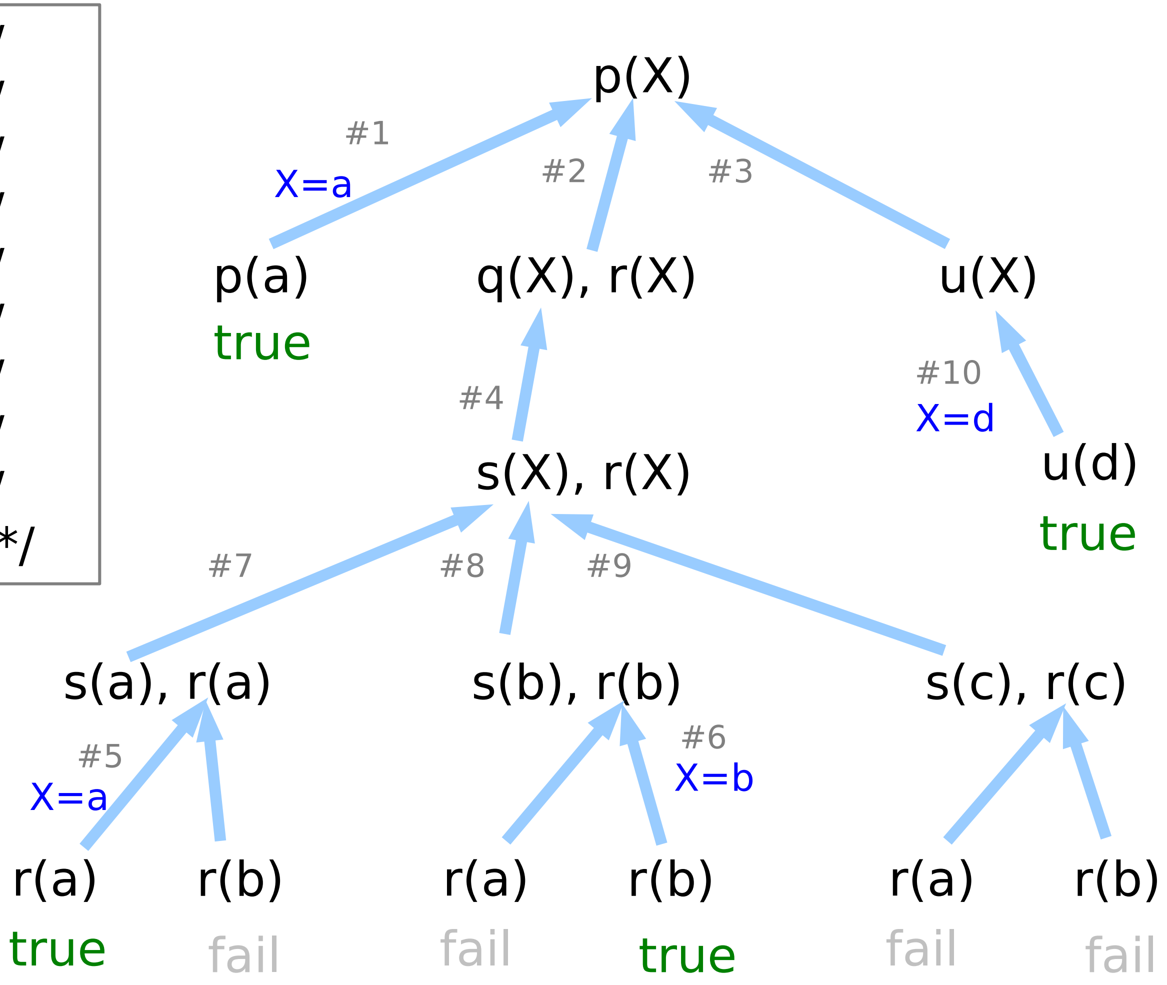
Young W. Lim
1/28/14

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

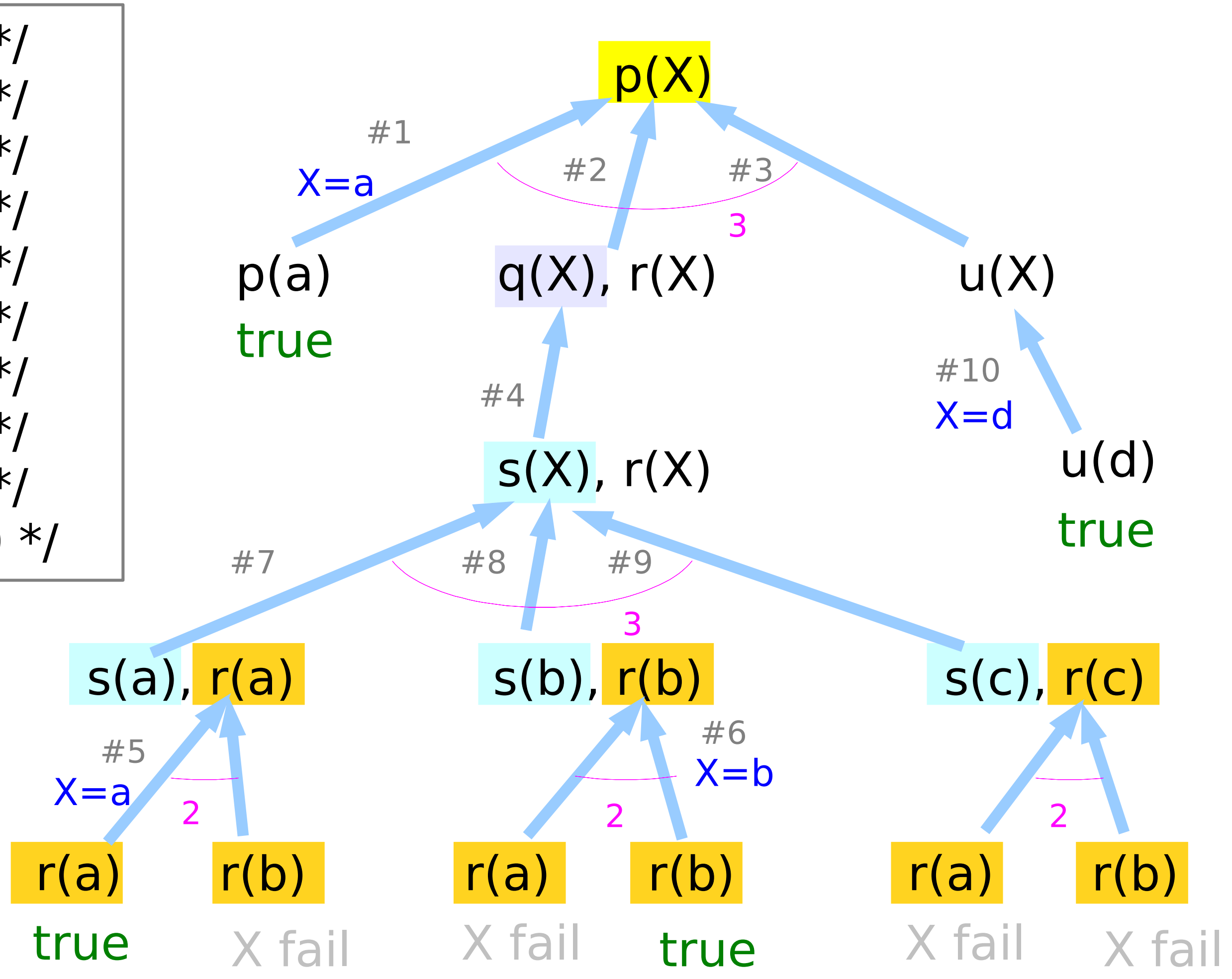
Derivation Tree Examples

```
p(a).                    /* #1 */
p(X) :- q(X), r(X).      /* #2 */
p(X) :- u(X).            /* #3 */
q(X) :- s(X).            /* #4 */
r(a).                    /* #5 */
r(b).                    /* #6 */
s(a).                    /* #7 */
s(b).                    /* #8 */
s(c).                    /* #9 */
u(d).                    /* #10 */
```



Derivation Tree Examples

```
     p(a).                      /* #1 */
3    p(X) :- q(X), r(X).        /* #2 */
     p(X) :- u(X).              /* #3 */
1    q(X) :- s(X).              /* #4 */
2    r(a).                      /* #5 */
     r(b).                      /* #6 */
     s(a).                      /* #7 */
3    s(b).                      /* #8 */
     s(c).                      /* #9 */
     u(d).                      /* #10 */
```

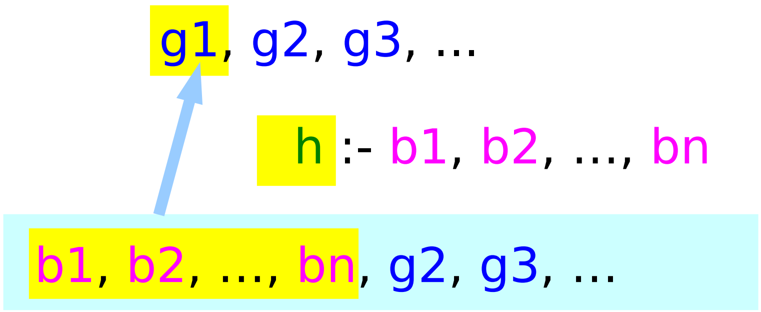


Replacing a selected subgoal

each node
the current goal
a sequence of subgoals

edges
the choices available for
replacing a selected subgoal

when g1 **unifies** with h



g1, g2, g3, ...

h :- b1, b2, ..., bn

b1, b2, ..., bn, g2, g3, ...

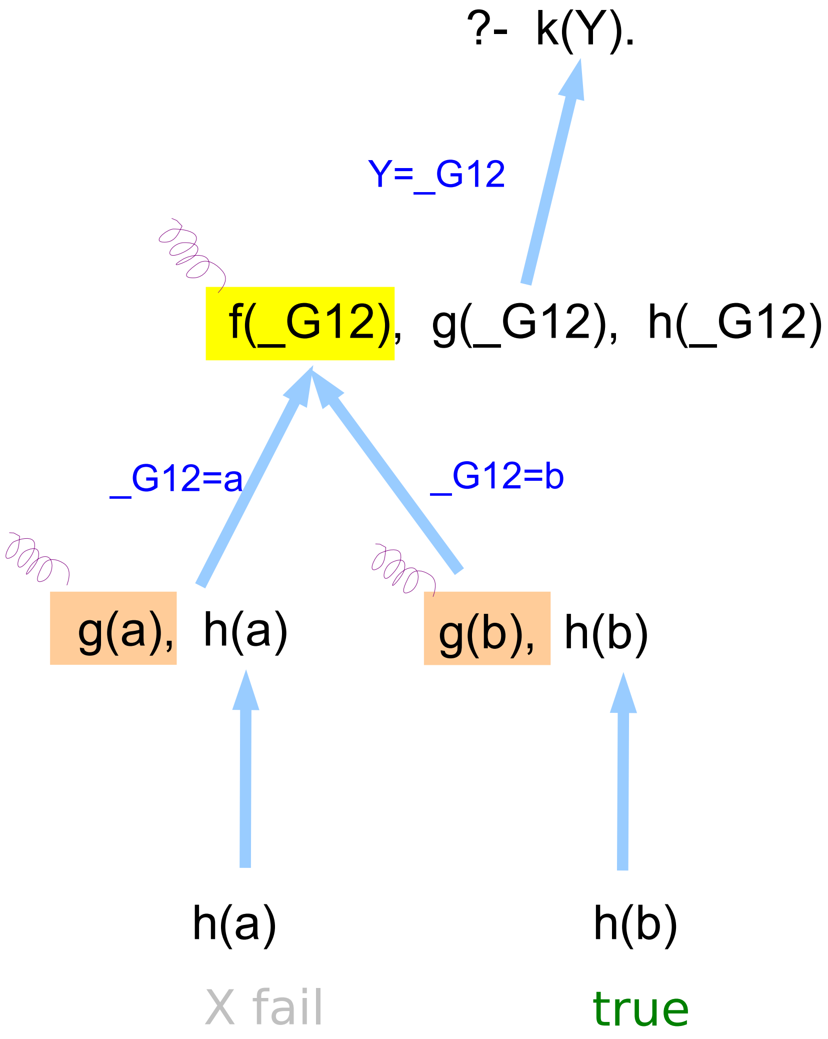
the logical variables in the body have been bound as a result of the **unification**
Prolog keeps track of **unifying substitutions**

- depth first traversal
- backtracking

Shared Variables

```
f(a).
f(b).

g(a).
g(b).

h(b).

k(X)  :-  f(X),  g(X),  h(X).
```

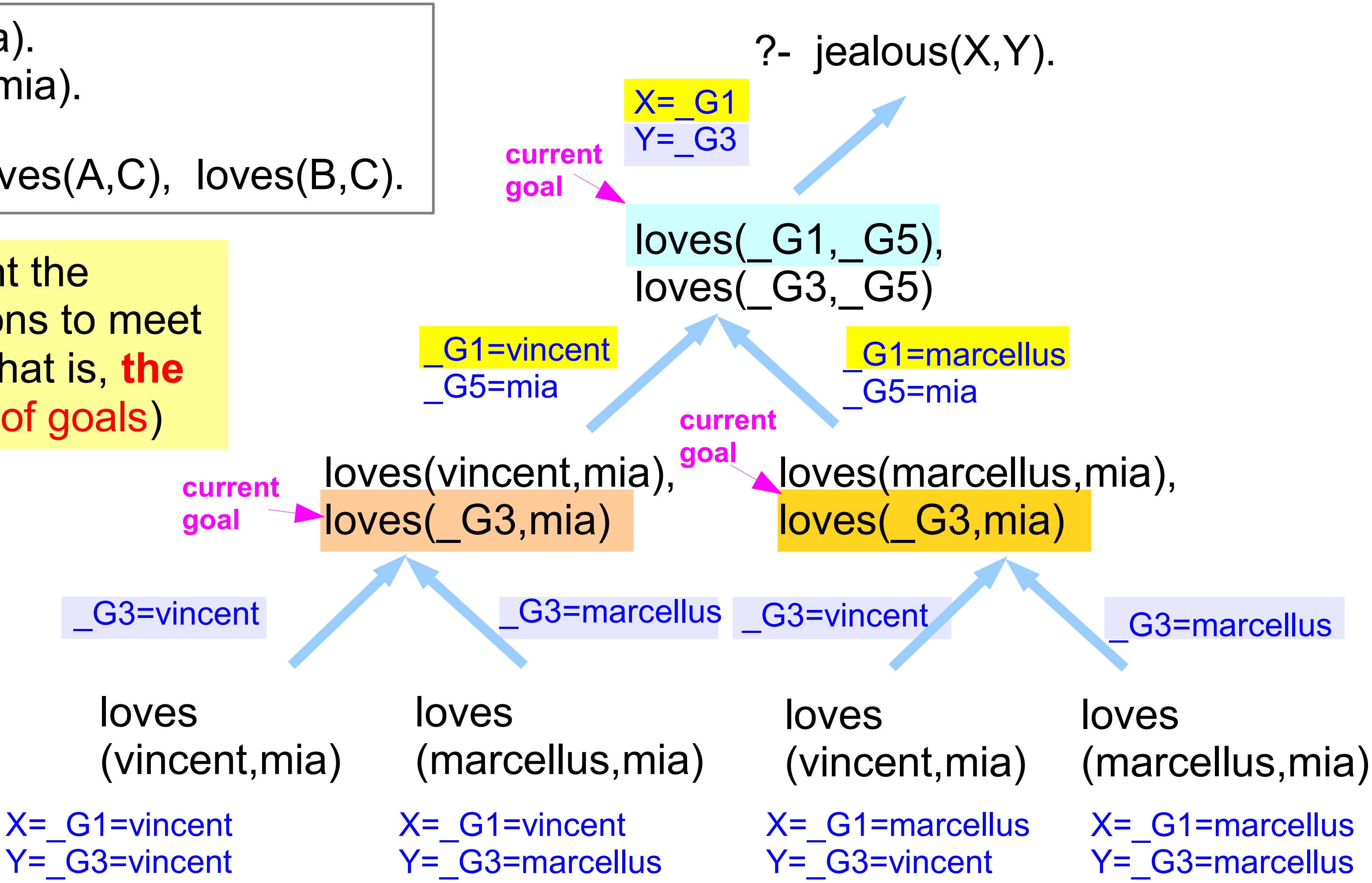
When Prolog unifies the variable in a query to a variable in a fact or rule, it generates a brand **new variable** (say _G12) to represent **the shared variables**.



Current Goals

```
loves(vincent,mia).
loves(marcellus,mia).

jealous(A,B):-  loves(A,C),  loves(B,C).
```

the edges represent the variable instantiations to meet the **current goal** (that is, **the first one** in the list of goals)



Writing Recursive Rules

```
child(bridget,caroline).
child(caroline,donna).

descend(X,Y)  :-  child(X,Y).
descend(X,Y)  :-  child(X,Z), child(Z,Y).
```

a non-recursive rule

a recursive rule

```
child(anne,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y)  :-  child(X,Z_1),
                  child(Z_1,Z_2),
                  child(Z_2,Y).

descend(X,Y)  :-  child(X,Z_1),
                  child(Z_1,Z_2),
                  child(Z_2,Z_3),
                  child(Z_3,Y).
```

a non-recursive rule

```
child(anne,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y)  :-  child(X,Y).
descend(X,Y)  :-  child(X,Z), descend(Z,Y).
```

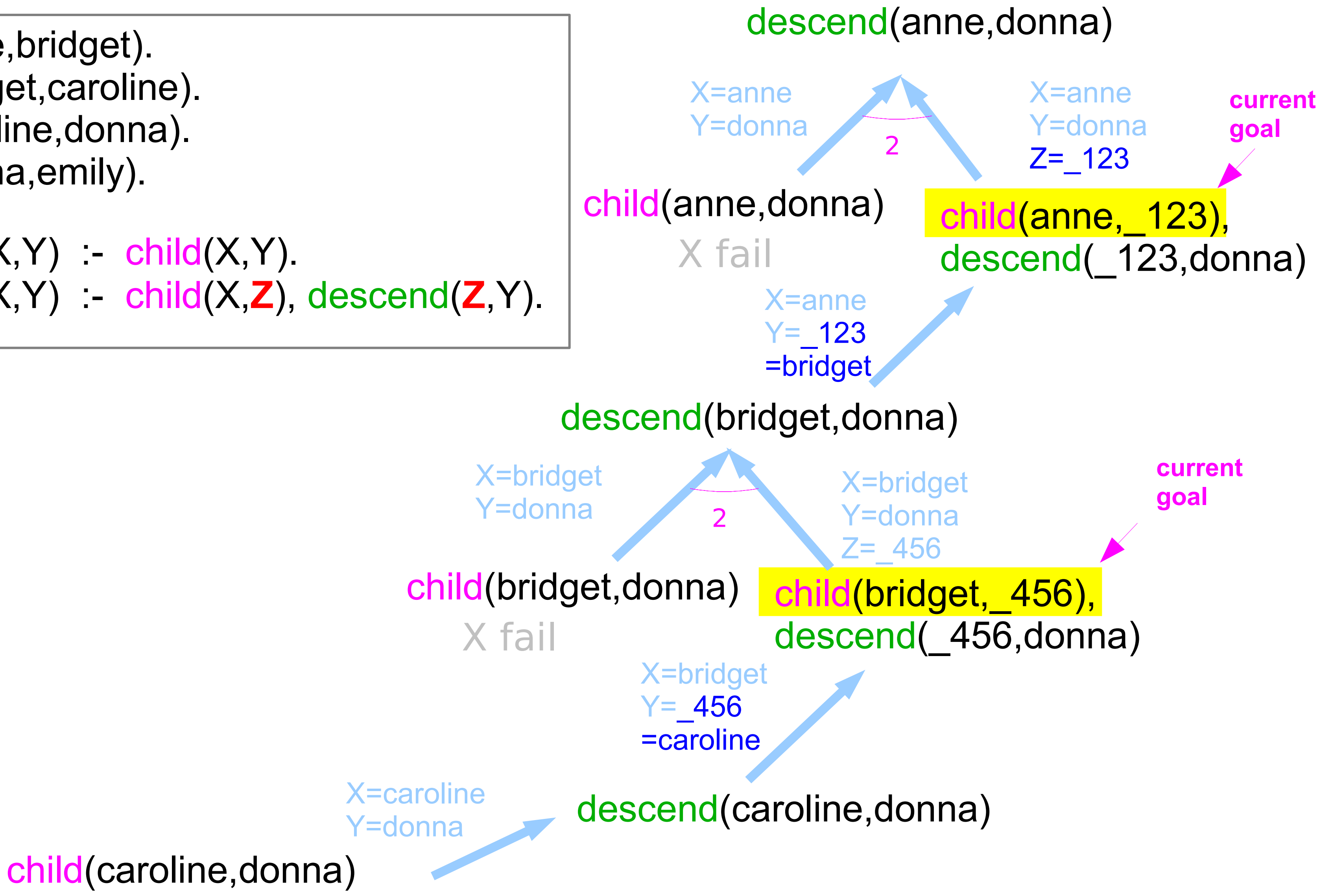
a problem

a smaller problem

The first rule first

```
child(anne,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y)  :-  child(X,Y).
descend(X,Y)  :-  child(X,Z), descend(Z,Y).
```

2



A Number System

The number representation using only 4 symbols: 0, **succ**, **(,)**

succ(X) : the number X + 1

```
           number  2          number  2                     number  4
?-  add(succ(succ(0)), succ(succ(0)), succ(succ(succ(succ(0))))).
Yes
```

```
           number  2       number  1
?-  add(succ(succ(0)), succ(0), Y).
Y  =  succ(succ(succ(0)))
```

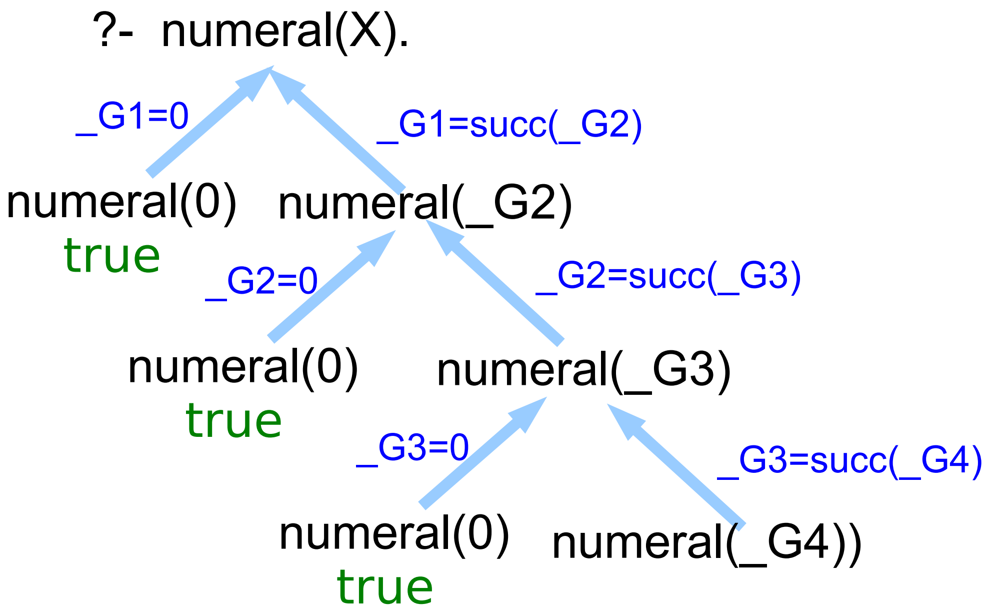
```
add(0, Y, Y).
add(succ(X), Y, succ(Z))  :- add(X, Y, Z).
```

0 + Y = Y

X+1 + Y = Z+1 ⬅ X + Y = Z

Alternative Solutions

```
numeral(0).

numeral(succ(X)) :-
      numeral(X).
```



?- numeral(X).

_G1=0

_G1=succ(_G2)

numeral(0)
true

numeral(_G2)

_G2=0

_G2=succ(_G3)

numeral(0)
true

numeral(_G3)

_G3=0

_G3=succ(_G4)

numeral(0)
true

numeral(_G4))

```
X = 0 ;
X = succ(0) ;
X = succ(succ(0)) ;
X = succ(succ(succ(0))) ;
X = succ(succ(succ(succ(0)))) ;
```

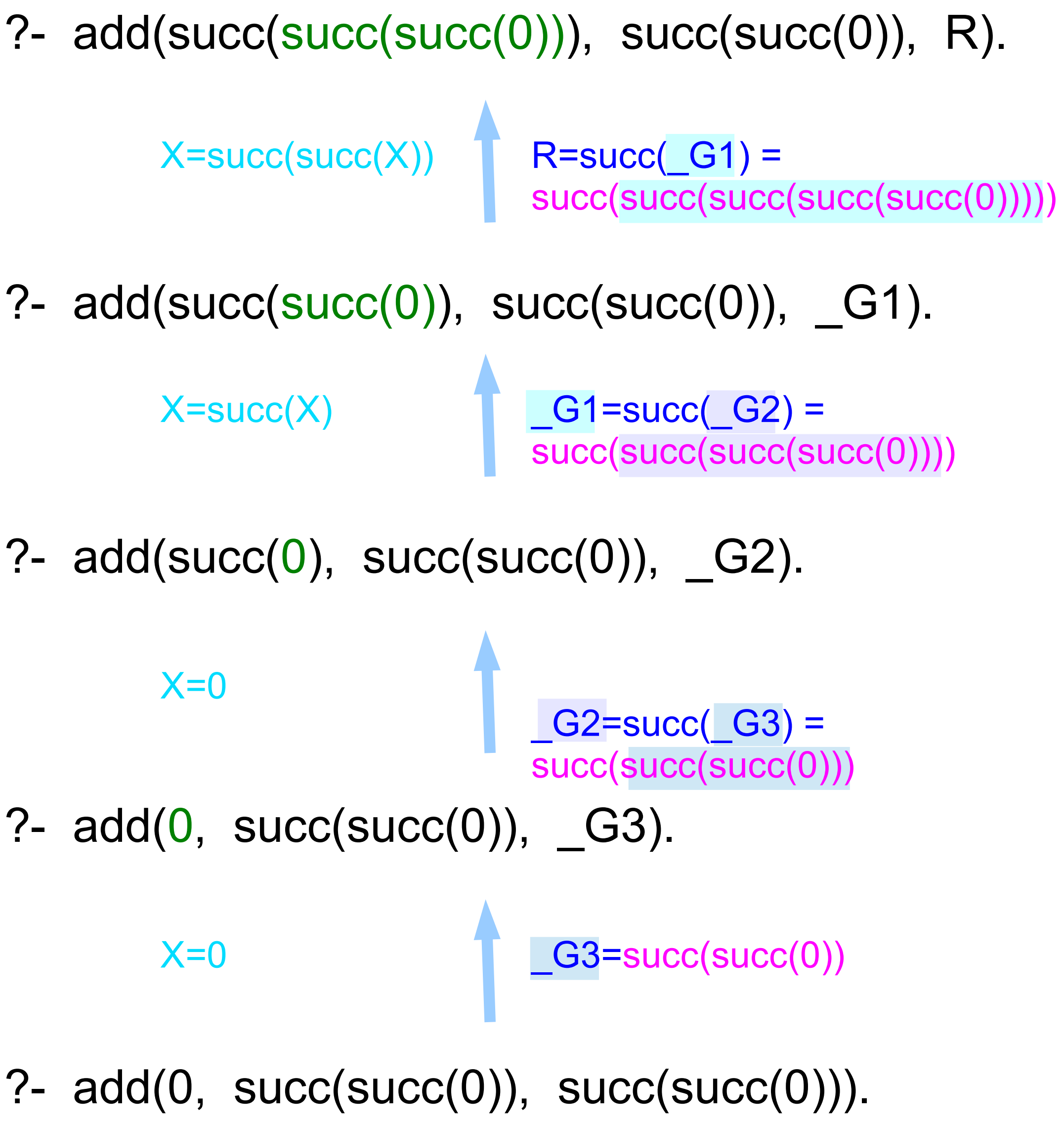
```
numeral(X).

?- numeral(_G1).

   numeral(succ(_G2)) :- numeral(_G2).

numeral(succ(X)) :- numeral(X).
```

Instantiating with a complex term

```
add(0, Y, Y).
add(succ(X), Y, succ(Z))
        :- add(X, Y, Z).
```

R was instantiated to succ(_G1)

But that means that R is not a completely uninstantiated variable anymore. It is now a complex term, that has a (uninstantiated) variable as its argument.



?- add(succ(succ(succ(0))), succ(succ(0)), R).

X=succ(succ(X))

R=succ(_G1) = succ(succ(succ(succ(succ(0)))))

?- add(succ(succ(0)), succ(succ(0)), _G1).

X=succ(X)

_G1=succ(_G2) = succ(succ(succ(succ(0))))

?- add(succ(0), succ(succ(0)), _G2).

X=0

_G2=succ(_G3) = succ(succ(succ(0)))

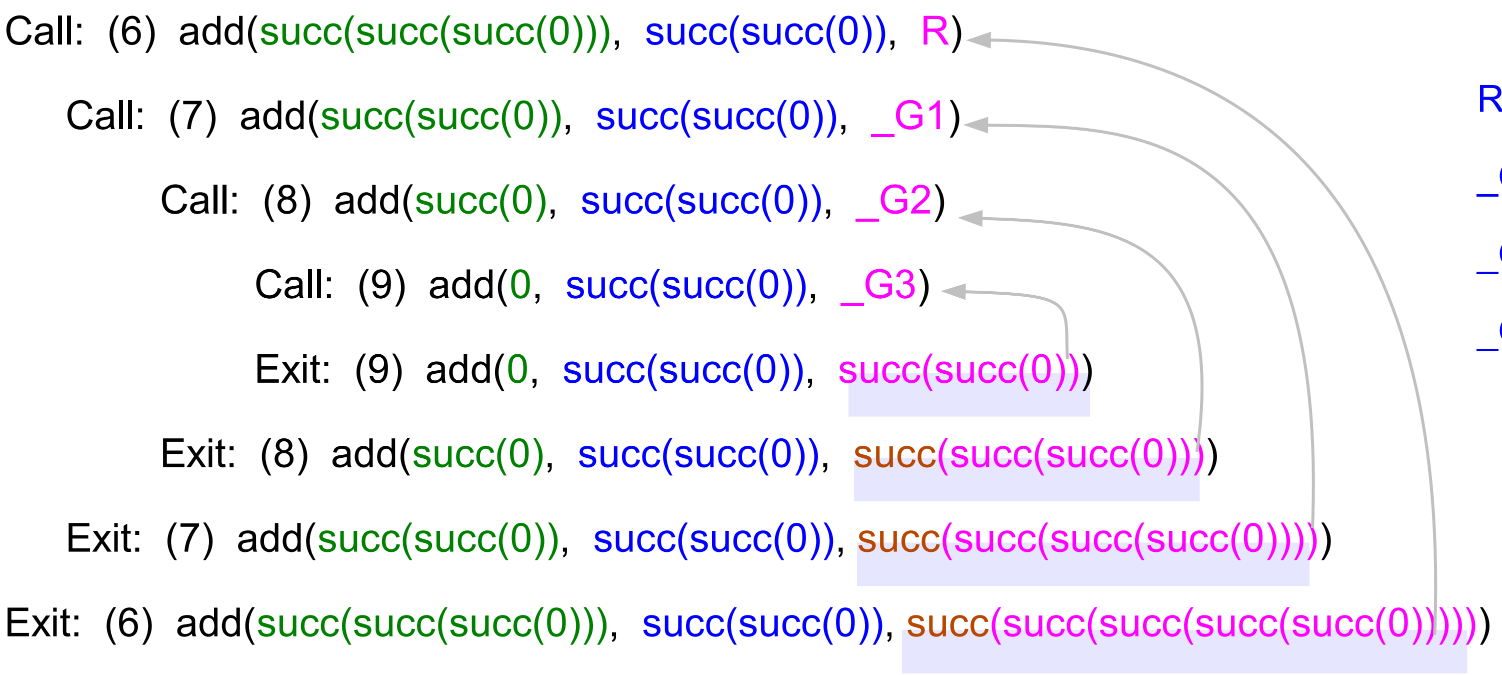
?- add(0, succ(succ(0)), _G3).

X=0

_G3=succ(succ(0))

?- add(0, succ(succ(0)), succ(succ(0))).

Instantiations On Exiting



Call: (6) add(succ(succ(succ(0))), succ(succ(0)), R)

Call: (7) add(succ(succ(0)), succ(succ(0)), _G1)

Call: (8) add(succ(0), succ(succ(0)), _G2)

Call: (9) add(0, succ(succ(0)), _G3)

Exit: (9) add(0, succ(succ(0)), succ(succ(0)))

Exit: (8) add(succ(0), succ(succ(0)), succ(succ(succ(0))))

Exit: (7) add(succ(succ(0)), succ(succ(0)), succ(succ(succ(succ(0)))))

Exit: (6) add(succ(succ(succ(0))), succ(succ(0)), succ(succ(succ(succ(succ(0))))))

R=succ(_G1)

_G1=succ(_G2)

_G2=succ(_G3)

_G3=succ(succ(0))

_G3

_G2

_G1

R

R=succ(_G1)
= succ(succ(_G2))
= succ(succ(succ(_G3)))
= succ(succ(succ(succ(succ(0)))))

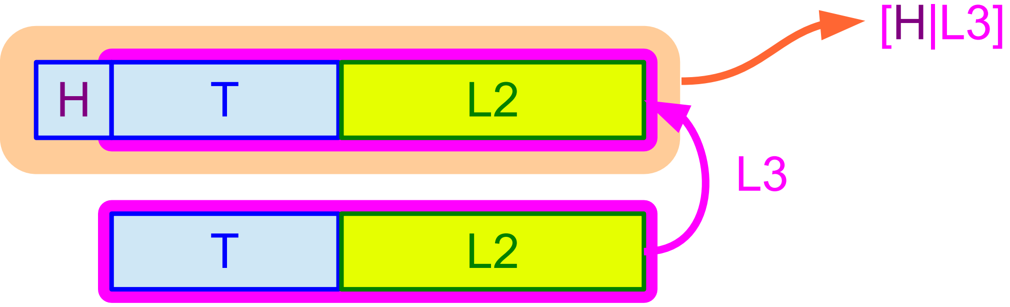
Recursive Definition and Base Case

```
?-  append([a,b,c], [1,2,3], [a,b,c,1,2,3]).
yes

?-  append([a,[foo,gibble],c], [1,2,[[],b]], [a,[foo,gibble],c,1,2,[[],b]]).
yes

?-  append([a,b,c], [1,2,3], L3).
L3  =  [a,b,c,1,2,3]
yes
```

```
append([], L, L).

append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
```

base case

recursive definition



Instantiation during calling and exiting

```
append([], L, L).

append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
```

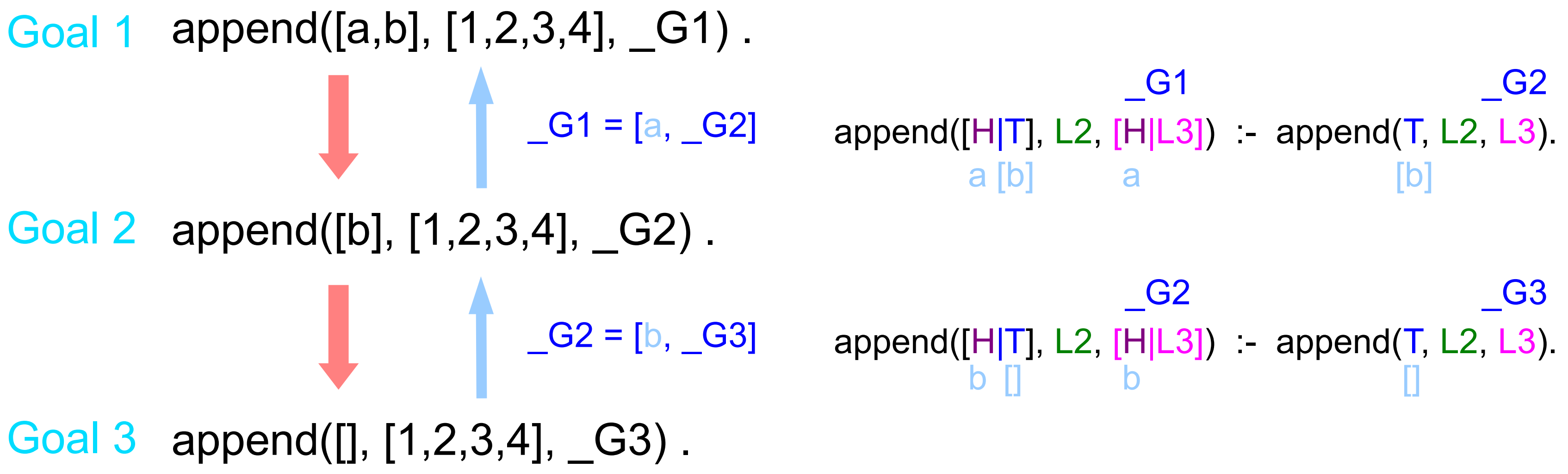
[H|T] is instantiated when the call is made

[H|L3] is instantiated during exiting the call

```
?-  append([a,b], [1,2,3,4], X) .

append([a,  b],  [1,  2,  3, 4],  _G12)
  append([b],  [1,  2,  3, 4],  _G34)
    append([],  [1,  2,  3, 4],  _G56)
    append([],  [1,  2,  3, 4],  [1,  2,  3, 4])
  append([b],  [1,  2,  3, 4],  [b,  1,  2,  3, 4])
append([a,  b],  [1,  2,  3, 4],  [a,  b, 1,  2,  3, 4])

X  =  [a,  b,  c,  1,  2,  3,  4]
yes
```

```
_G1        =  [a| _G1]
           =  [a| [b | _G2]]
           =  [a| [b | [c | _G3]]]
```

Finding subgoals



Goal 1 append([a,b], [1,2,3,4], _G1) .

_G1 = [a, _G2]

```
                 _G1                _G2
append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
        a [b]      a                 [b]
```

Goal 2 append([b], [1,2,3,4], _G2) .

_G2 = [b, _G3]

```
                 _G2                _G3
append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
        b  []      b                 []
```

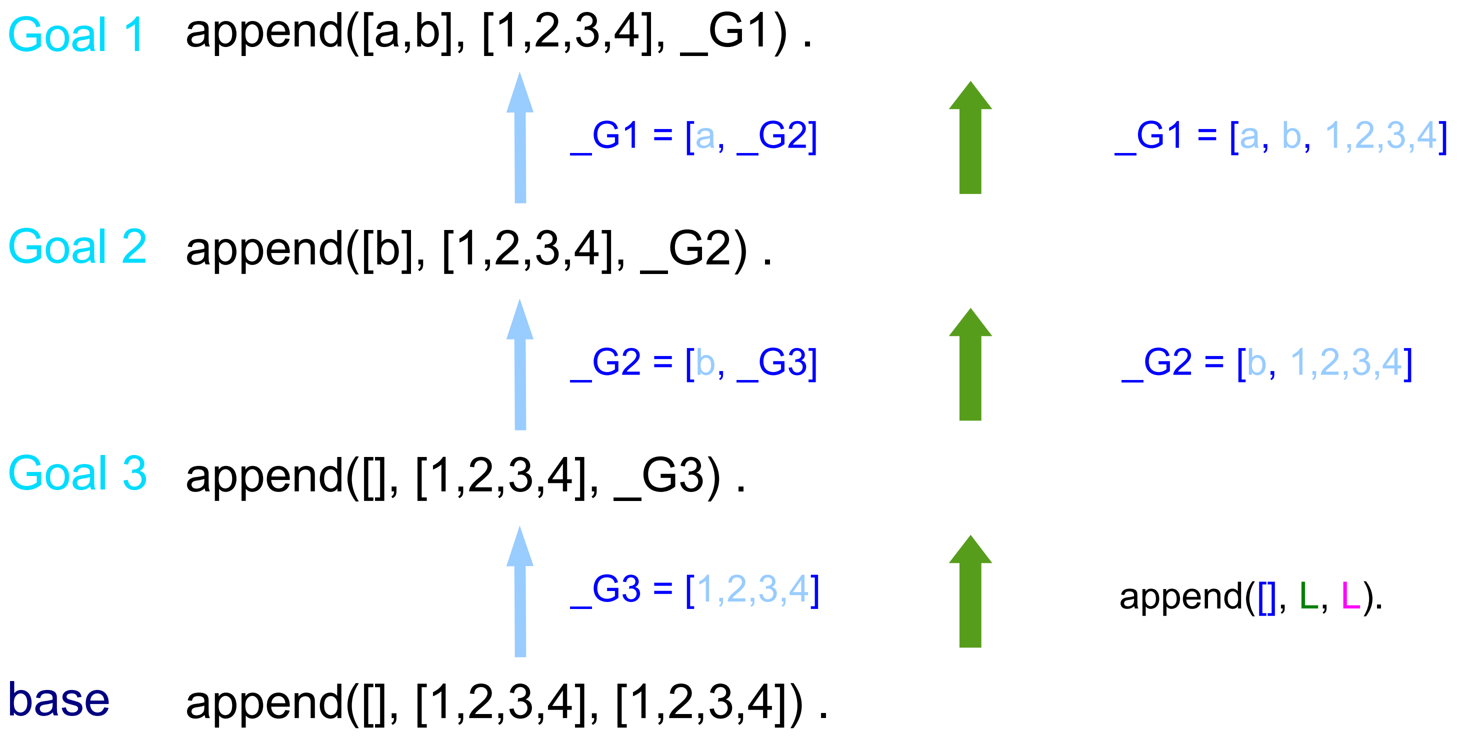
Goal 3 append([], [1,2,3,4], _G3) .

```
append([], L, L).

append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
```

[H|T] is instantiated when the call is made

[H|L3] is instantiated during exiting the call

Answering subgoals



Goal 1 append([a,b], [1,2,3,4], _G1) .

_G1 = [a, _G2]

_G1 = [a, b, 1,2,3,4]

Goal 2 append([b], [1,2,3,4], _G2) .

_G2 = [b, _G3]

_G2 = [b, 1,2,3,4]

Goal 3 append([], [1,2,3,4], _G3) .

_G3 = [1,2,3,4]

append([], L, L).

base append([], [1,2,3,4], [1,2,3,4]) .

```
append([], L, L).

append([H|T], L2, [H|L3])  :-  append(T, L2, L3).
```

[H|T] is instantiated when the call is made

[H|L3] is instantiated during exiting the call

References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html