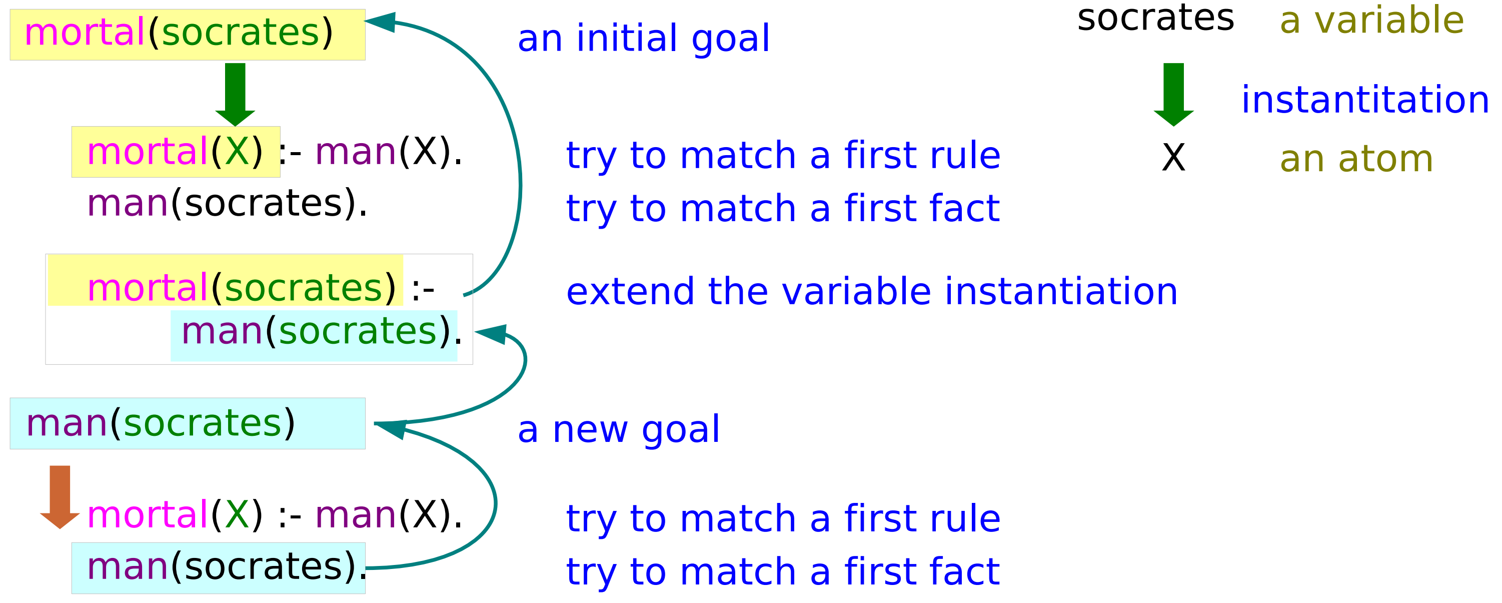
Introduction (1A)

Young Won Lim
9/17/13

Goal Execution

```
mortal(X) :- man(X).        a rule

man(socrates).              a fact

?- mortal(socrates).        a query
```



```
mortal(socrates)                        an initial goal

    mortal(X) :- man(X).                try to match a first rule
    man(socrates).                      try to match a first fact

    mortal(socrates) :-                 extend the variable instantiation
            man(socrates).

man(socrates)                           a new goal

    mortal(X) :- man(X).                try to match a first rule
    man(socrates).                      try to match a first fact
```

Select

Select (Element, List, Rest),

(List − Element) ➡ Rest

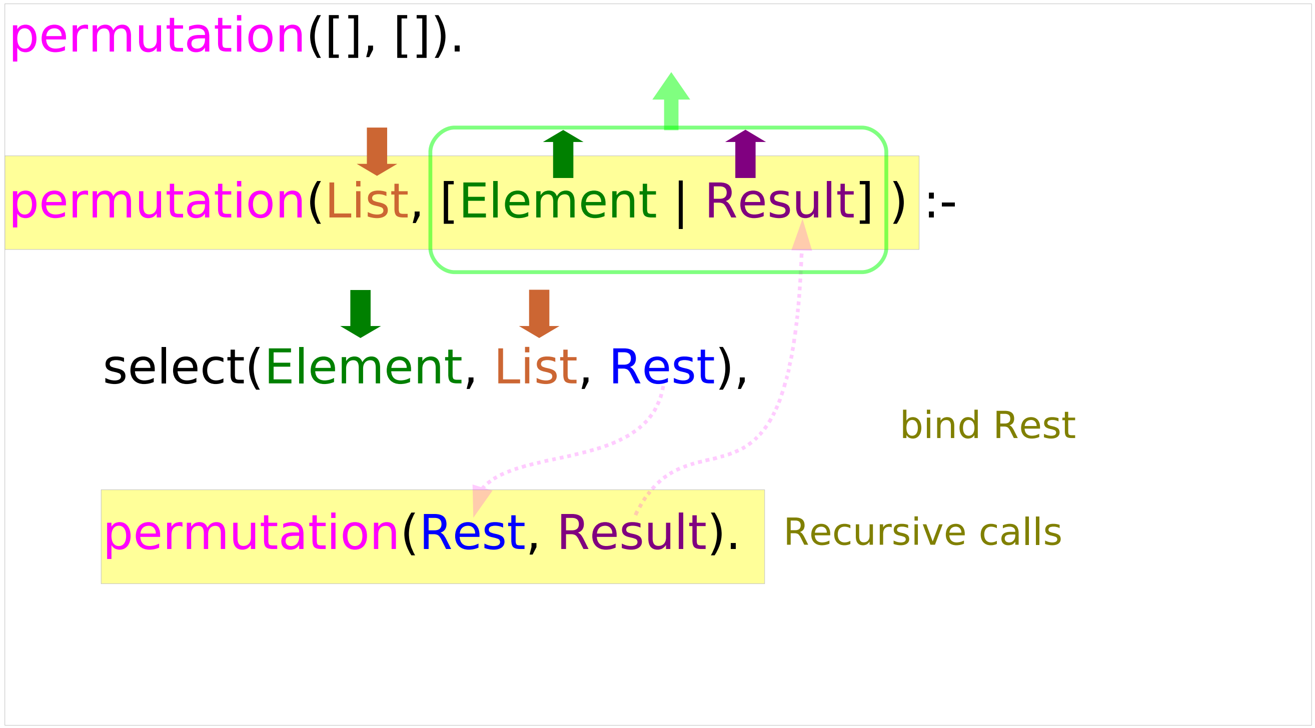
If **Element** is in **List**,
then remove **Element** from **List**
and return **Rest**

```
permutation([], []).

permutation(List, [Element | Result]) :-

    select(Element, List, Rest),

    permutation(Rest, Result).
```

bind Element

[Head | Tail]

Permutation – Recursive Call

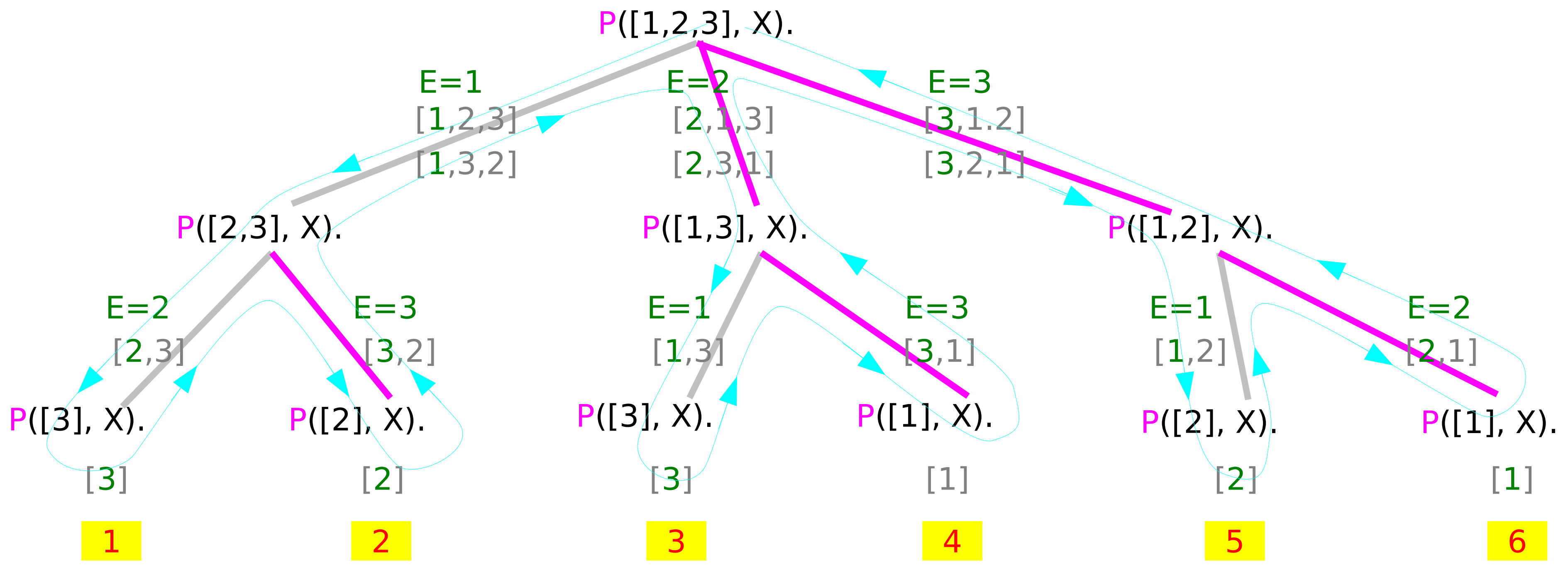


```
permutation([], []).

permutation(List, [Element | Result] ) :-

      select(Element, List, Rest),

      permutation(Rest, Result).
```

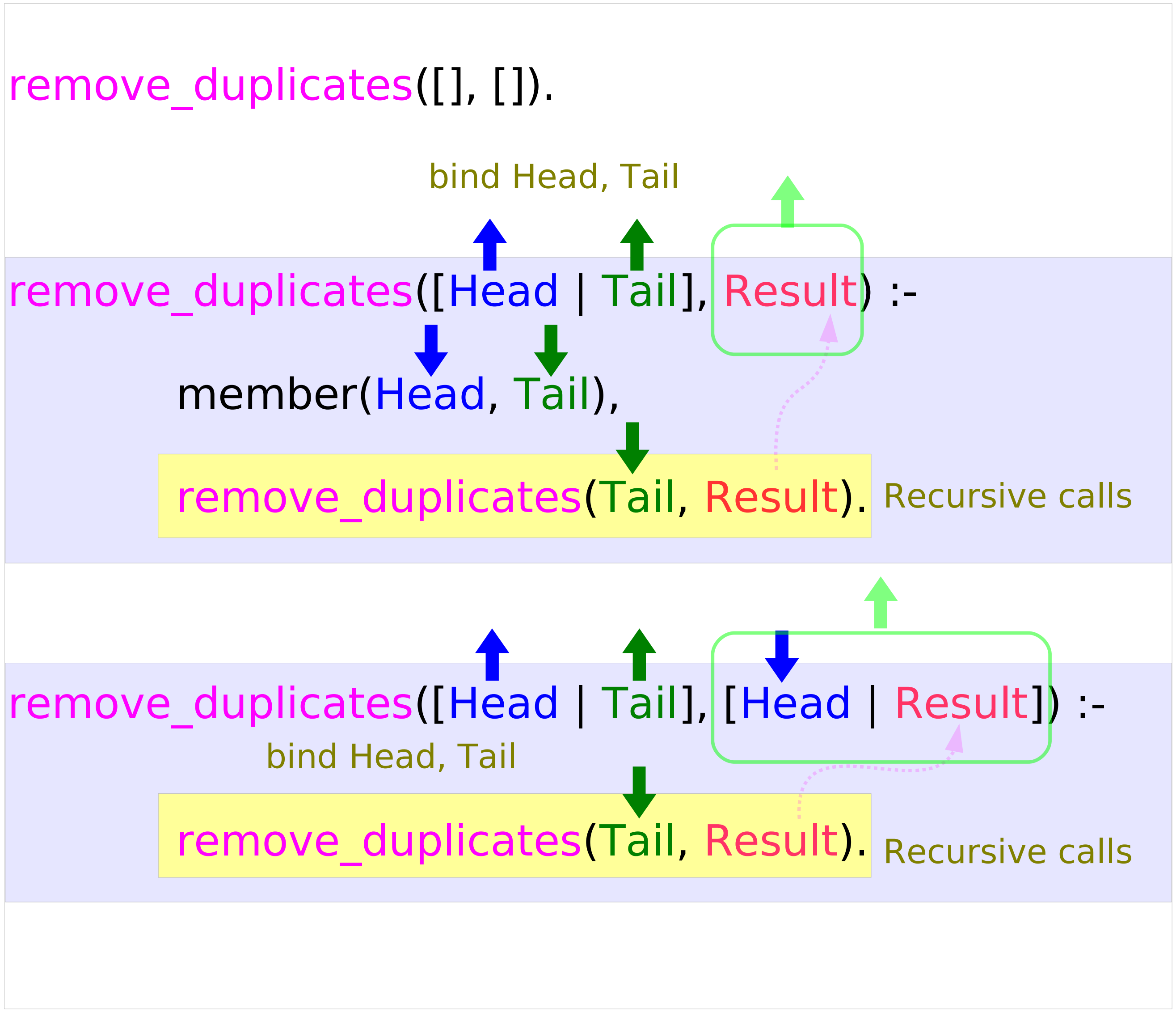
Permutation – Backtracking

```
Permutation([], []).

permutation(List, [Element | Result]) :-
       select(Element, List, Rest),
       permutation(Rest, Result).
```

```
X = [1, 2, 3] ;
X = [1, 3, 2] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [3, 1, 2] ;
X = [3, 2, 1] ;
No
```

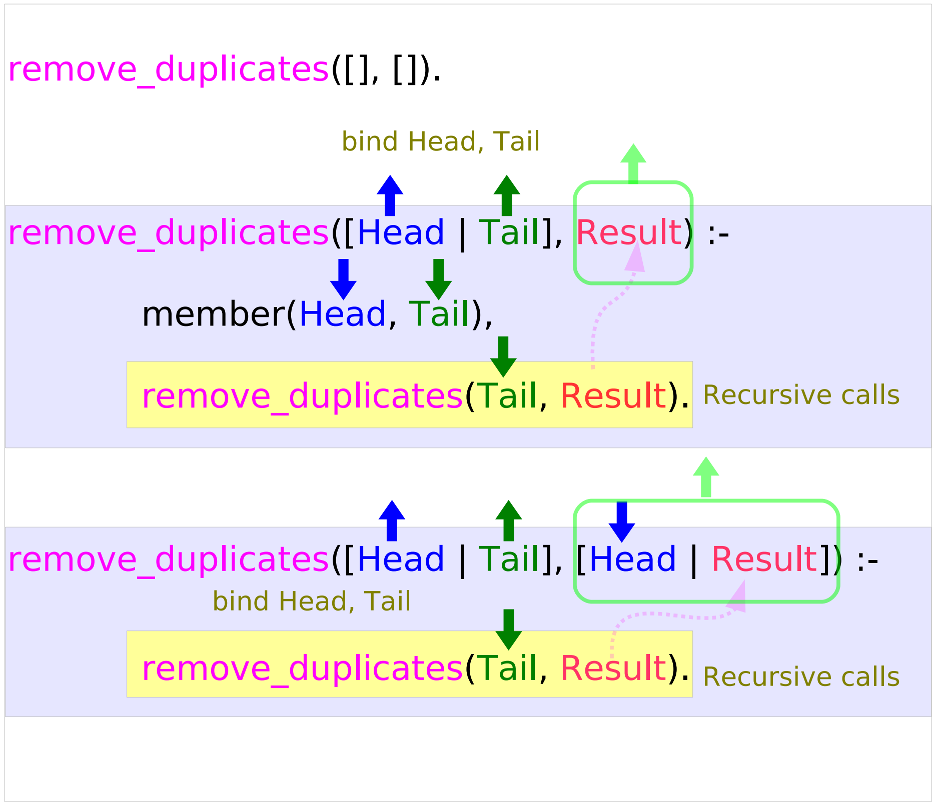


Remove Duplicates

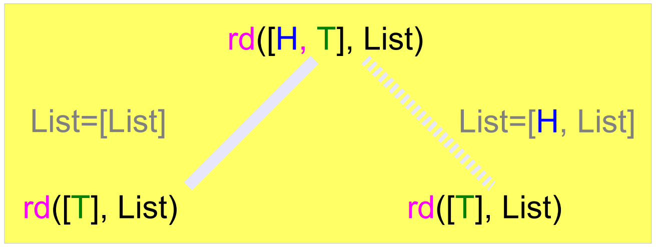


```
remove_duplicates([], []).

remove_duplicates([Head | Tail], Result) :-
        member(Head, Tail),
        remove_duplicates(Tail, Result).

remove_duplicates([Head | Tail], [Head | Result]) :-
        remove_duplicates(Tail, Result).
```

remove_duplicates – Backtracking



```
remove_duplicates([], []).

remove_duplicates([Head | Tail], Result) :-
        member(Head, Tail),
        remove_duplicates(Tail, Result).

remove_duplicates([Head | Tail], [Head | Result]) :-
        remove_duplicates(Tail, Result).
```



List = [b, c, a] **; (alternative ➡ backtracking)**

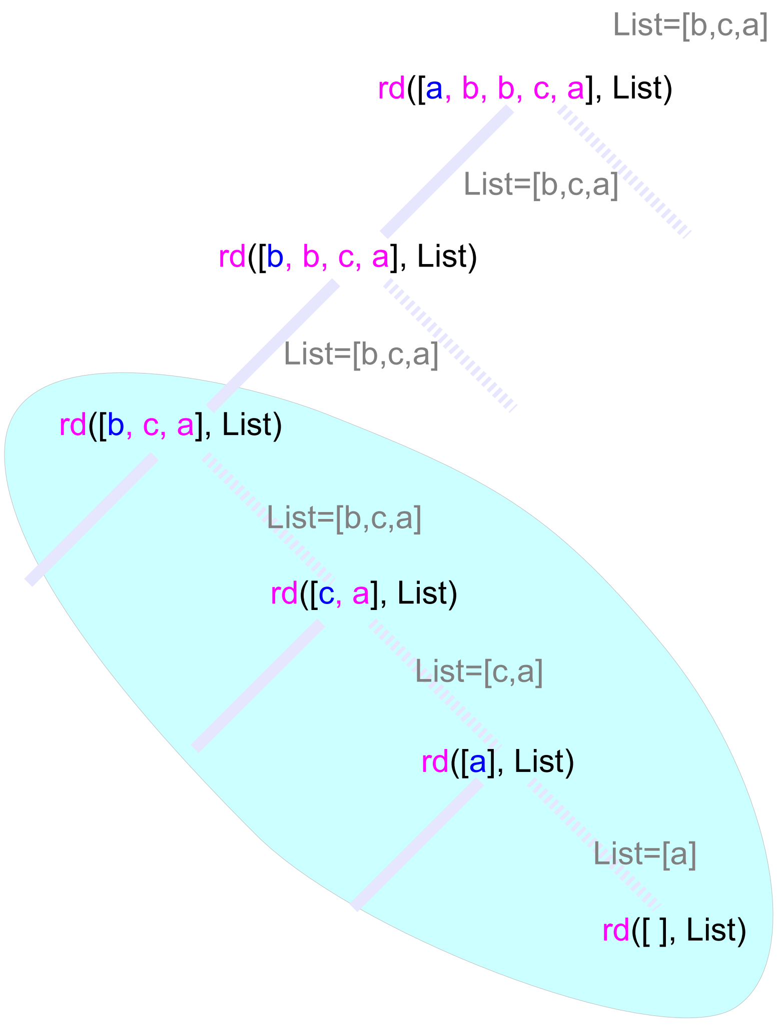
List = [b, b, c, a] **;**

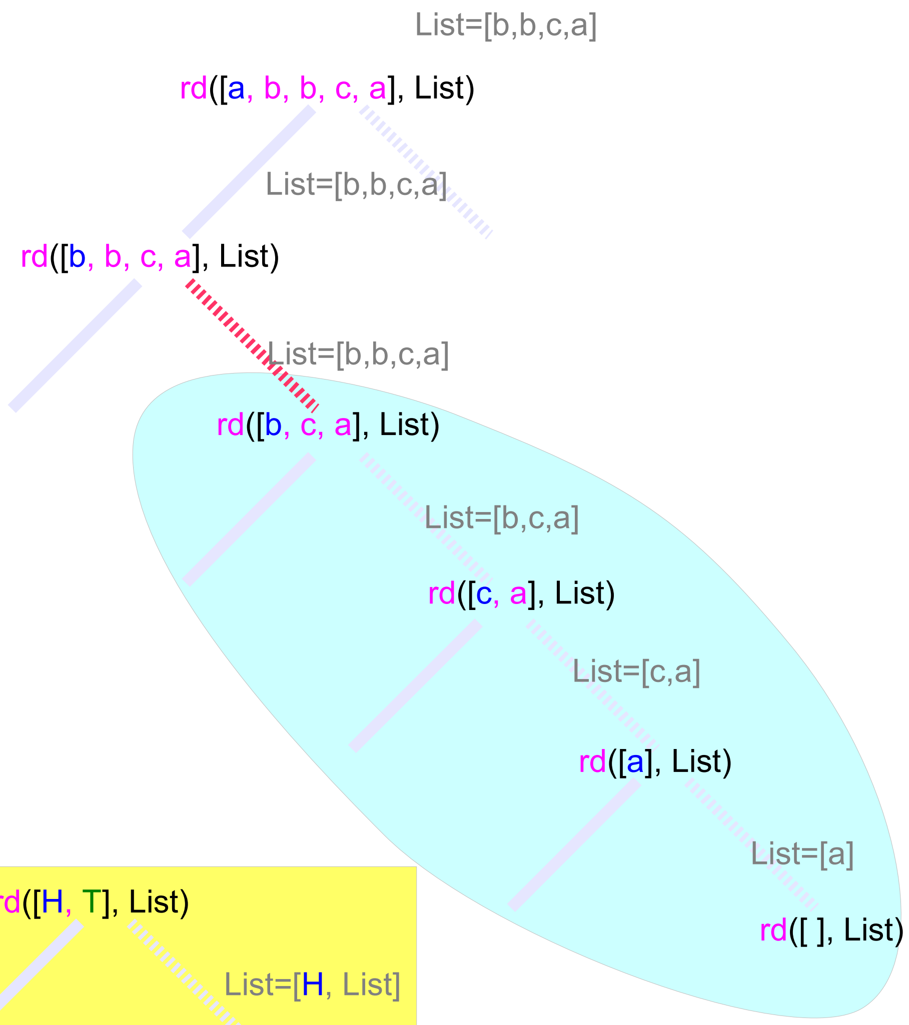
List = [a, b, c, a] **;**

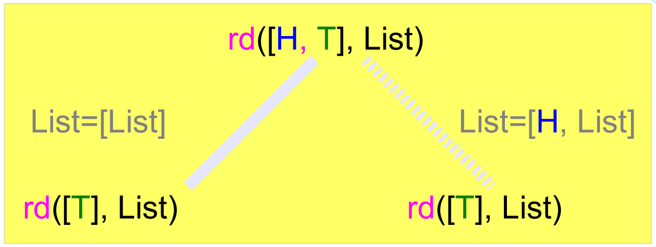
List = [a, b, b, c, a] **;**

No

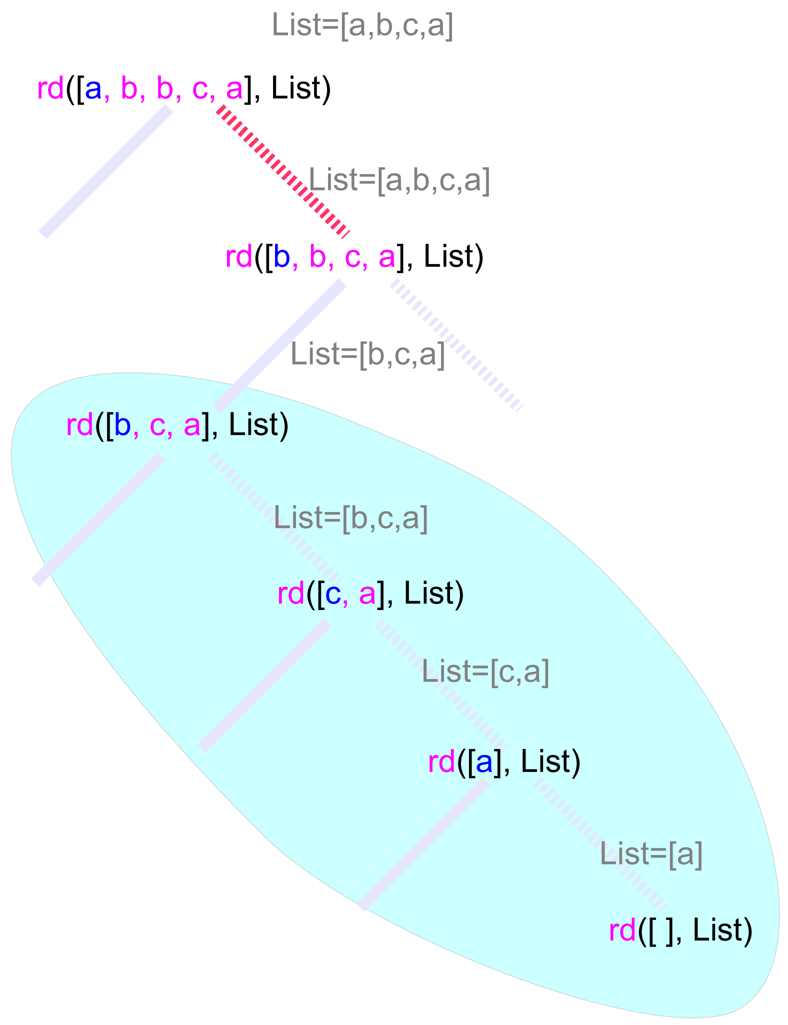
Backtracking (1)

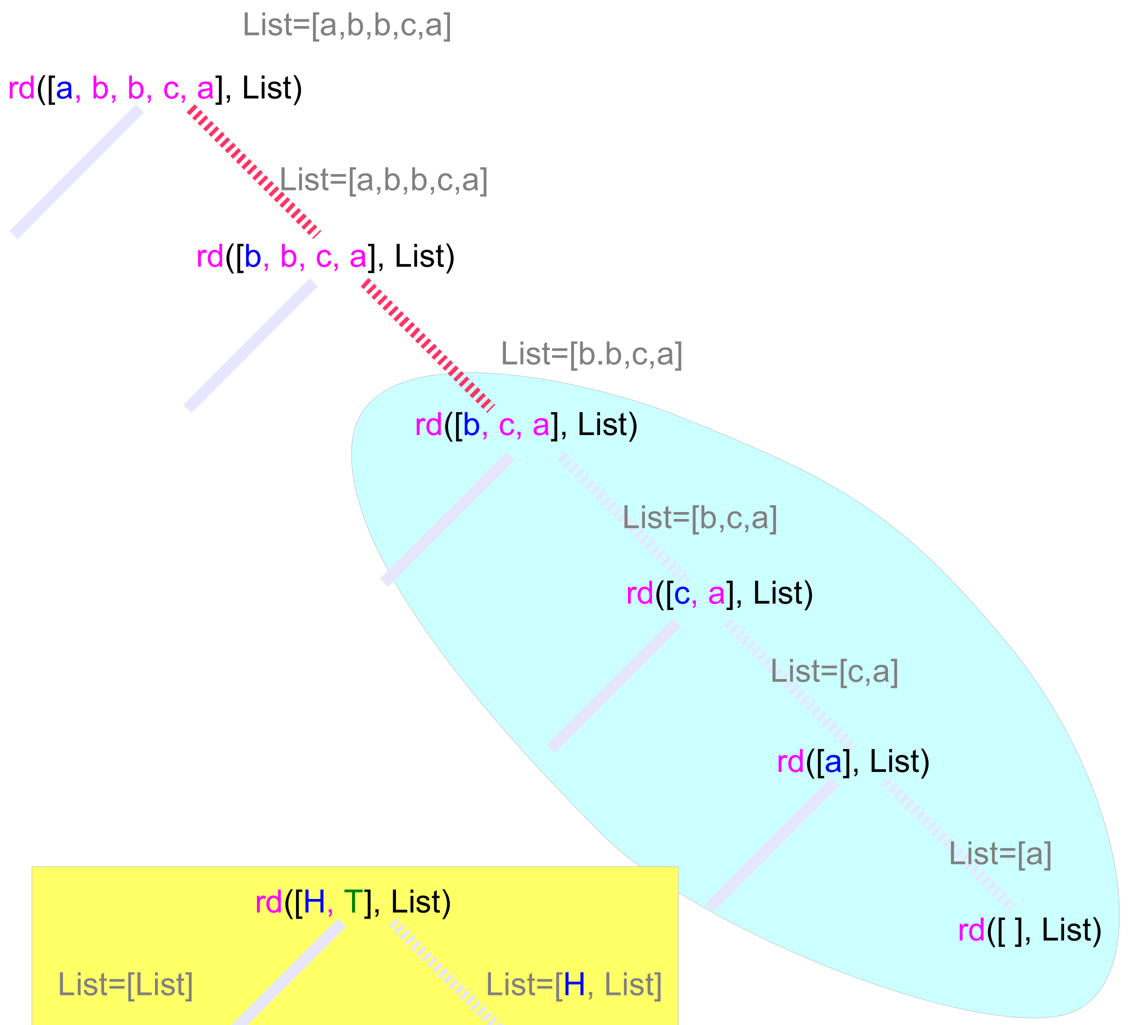


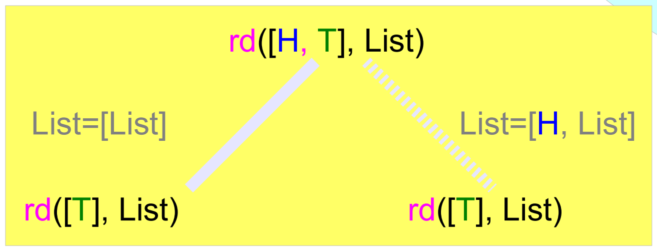




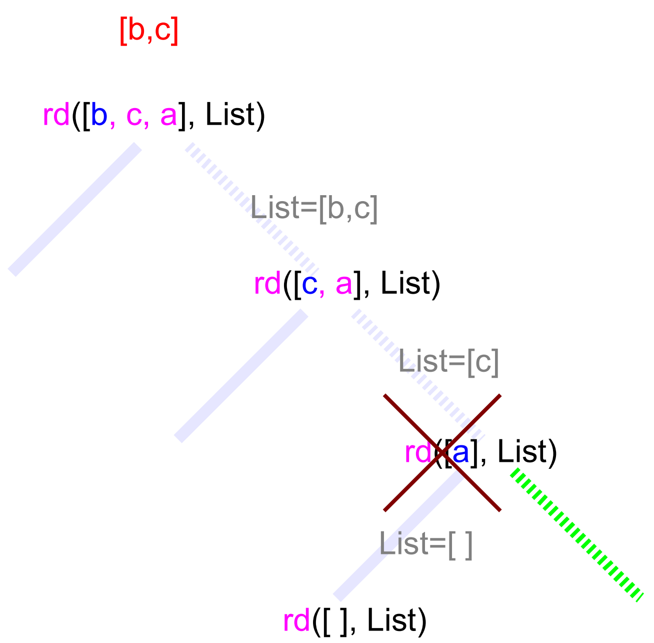
Backtracking (2)

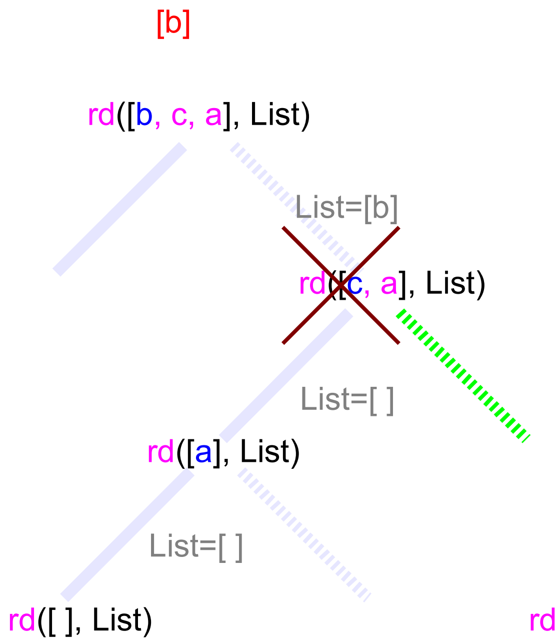


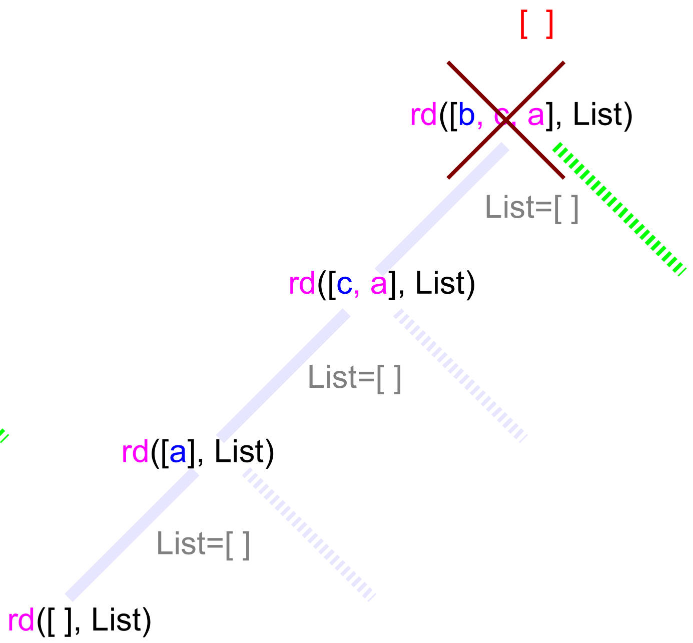




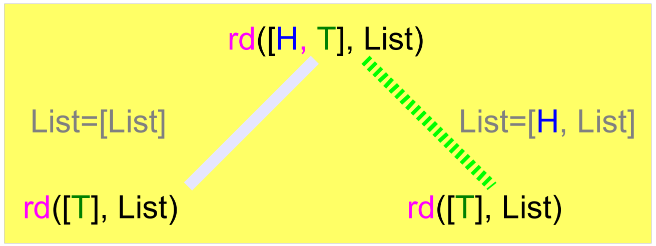
Backtracking (3)



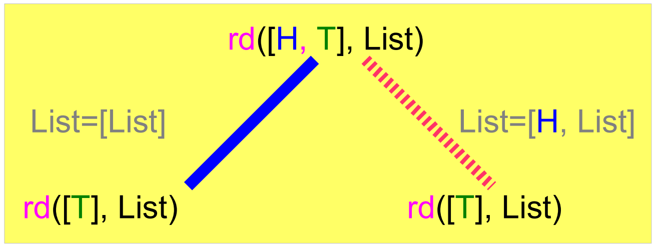




In the first solution, the first branch were rejected and the second were selected. No more branches are left to try for the alternative solutions.



In the first solution, the first branch were selected and the second are left to try for the alternative solutions.



During backtracking, however, also all other branches of the search tree will be visited.
Even if the first rule would match, sometimes the second one will be picked instead and the duplicate head will remain in the list.

Cut

! : cut, the predefined predicate

can be anywhere in a rule's body
can be a part of a sequence of subgoals in a query

The subgoal **!** is always succeed

backtracking into subgoals
    placed before the cut
    inside the same rule body
is not possible anymore

Whenever a cut is encountered in a rule's body,
all choices made between
    the time that rule's head has been matched
        with the parent goal
    and the time the cut is passed
are final, i.e. any choicepoints are being discarded.

A Cut Example

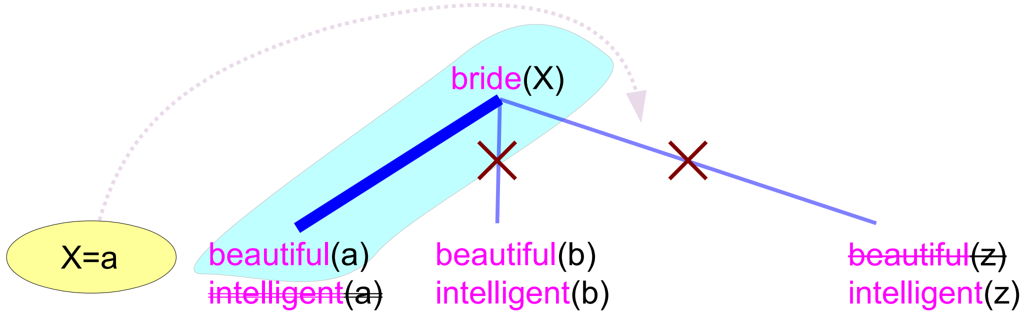
```
bride(Girl) :-
      beautiful(Girl), !,
      intelligent(Girl).
```

Bound variable before the cut are ***final***
Cannot change the first choice

Considered as True
regardless of the possible failure
of the subgoals after the cut

```
?- bride(X).
No
```

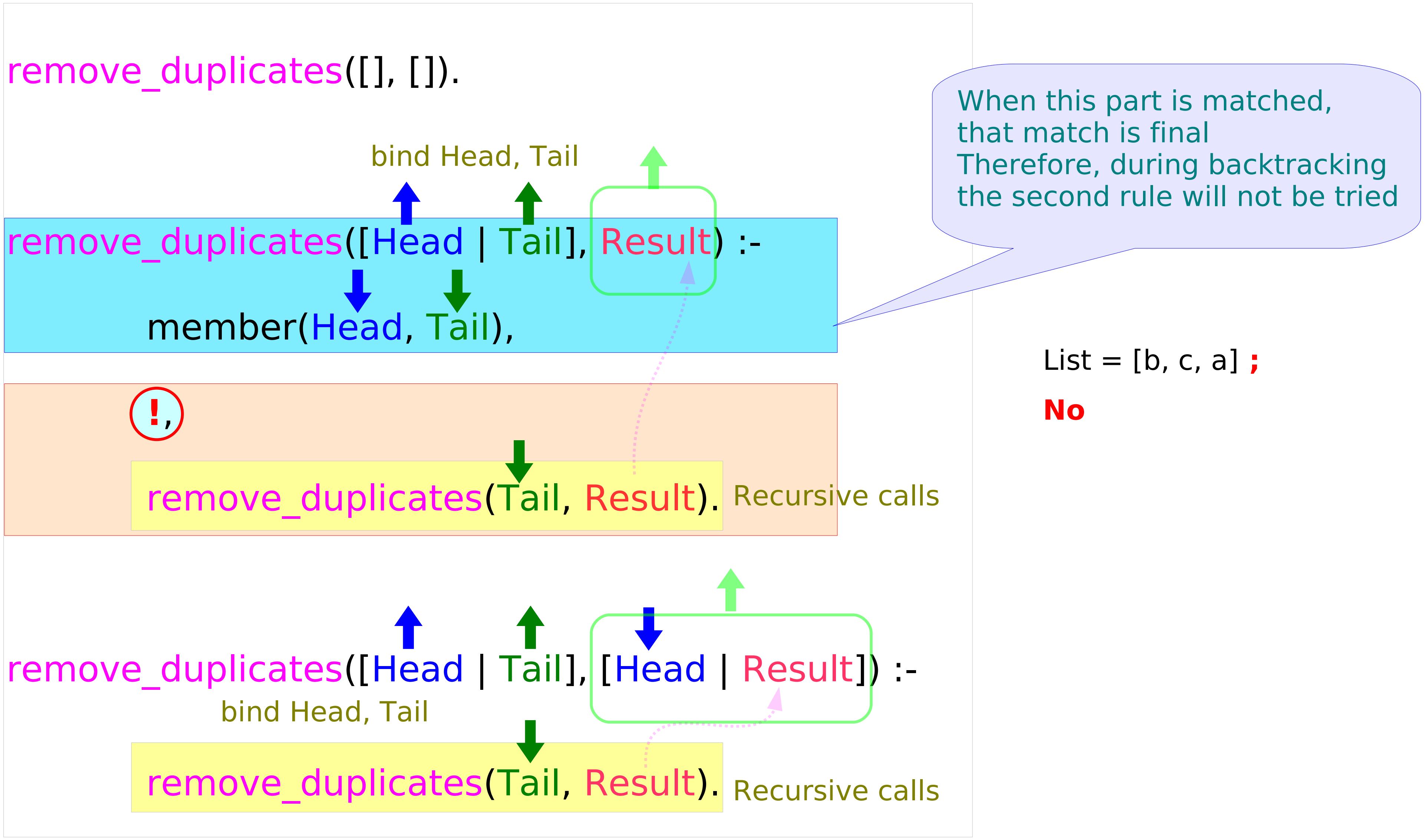
The first choice X=a is final
cannot try other alternatives



No match
after the cut
Result: **No**

Without the cut
Result: **X=b**

remove_duplicates with a cut



```
remove_duplicates([], []).

remove_duplicates([Head | Tail], Result) :-
        member(Head, Tail),
        !,
        remove_duplicates(Tail, Result).

remove_duplicates([Head | Tail], [Head | Result]) :-
        remove_duplicates(Tail, Result).
```

When this part is matched,
that match is final
Therefore, during backtracking
the second rule will not be tried

```
List = [b, c, a] ;
No
```

Member

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

bind Head & Tail

•Returns Yes if X is equal to Head

```
member(X, [H | T]) :-
```

bind Head & Tail

•Else try the recursive calls

```
    member(X, T)
```

Recursive calls

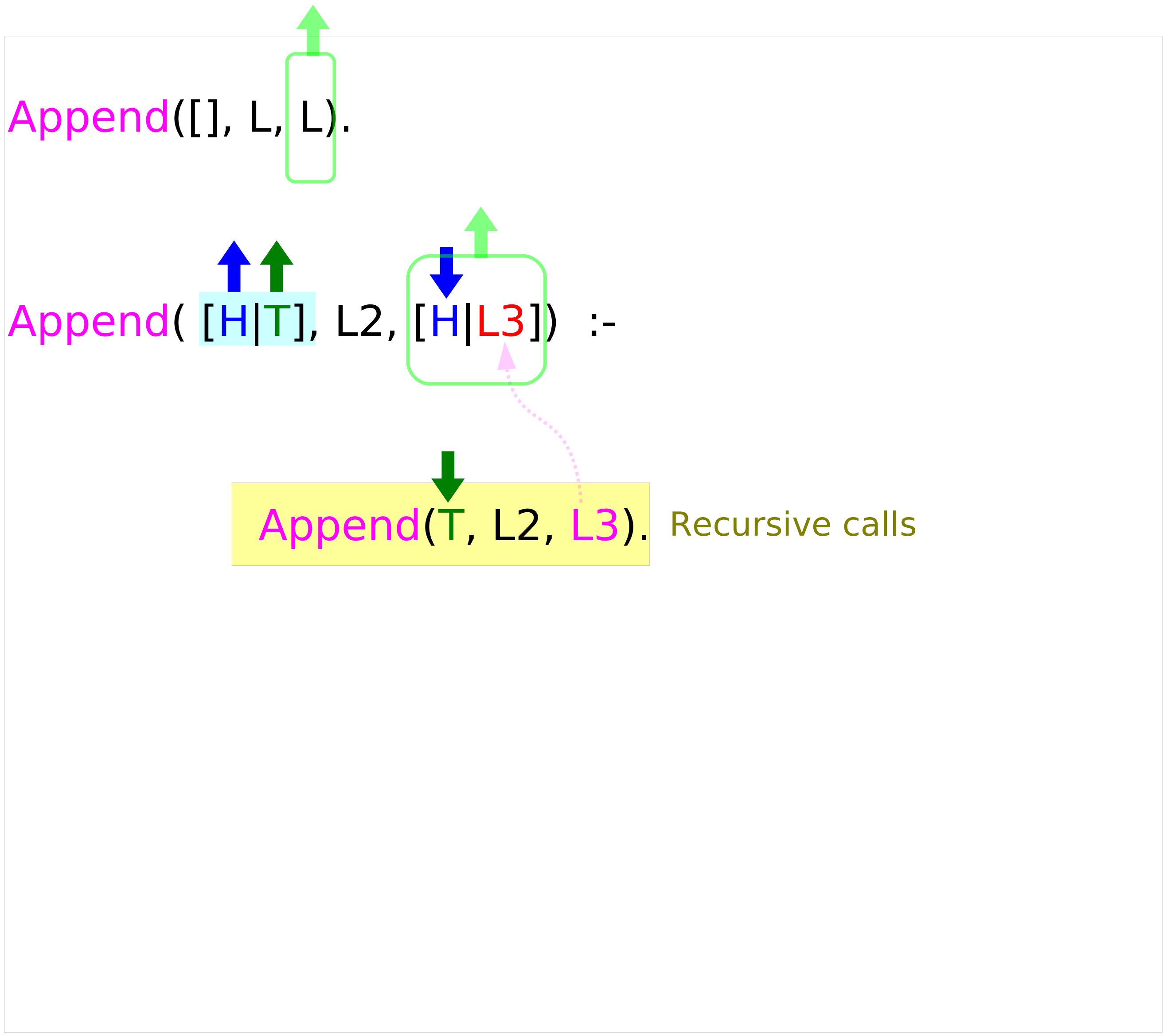
Recursing Down Lists

```
a2b([], []).

a2b([a | Ta], [b | Tb]) :-
    a2b(Ta, Tb)
```

Recursive calls

```
?- a2b([a, a, a], [b, b, b]) .
Yes
?- a2b([a, 8, 9], [b, b, b]) .
No
```

Append



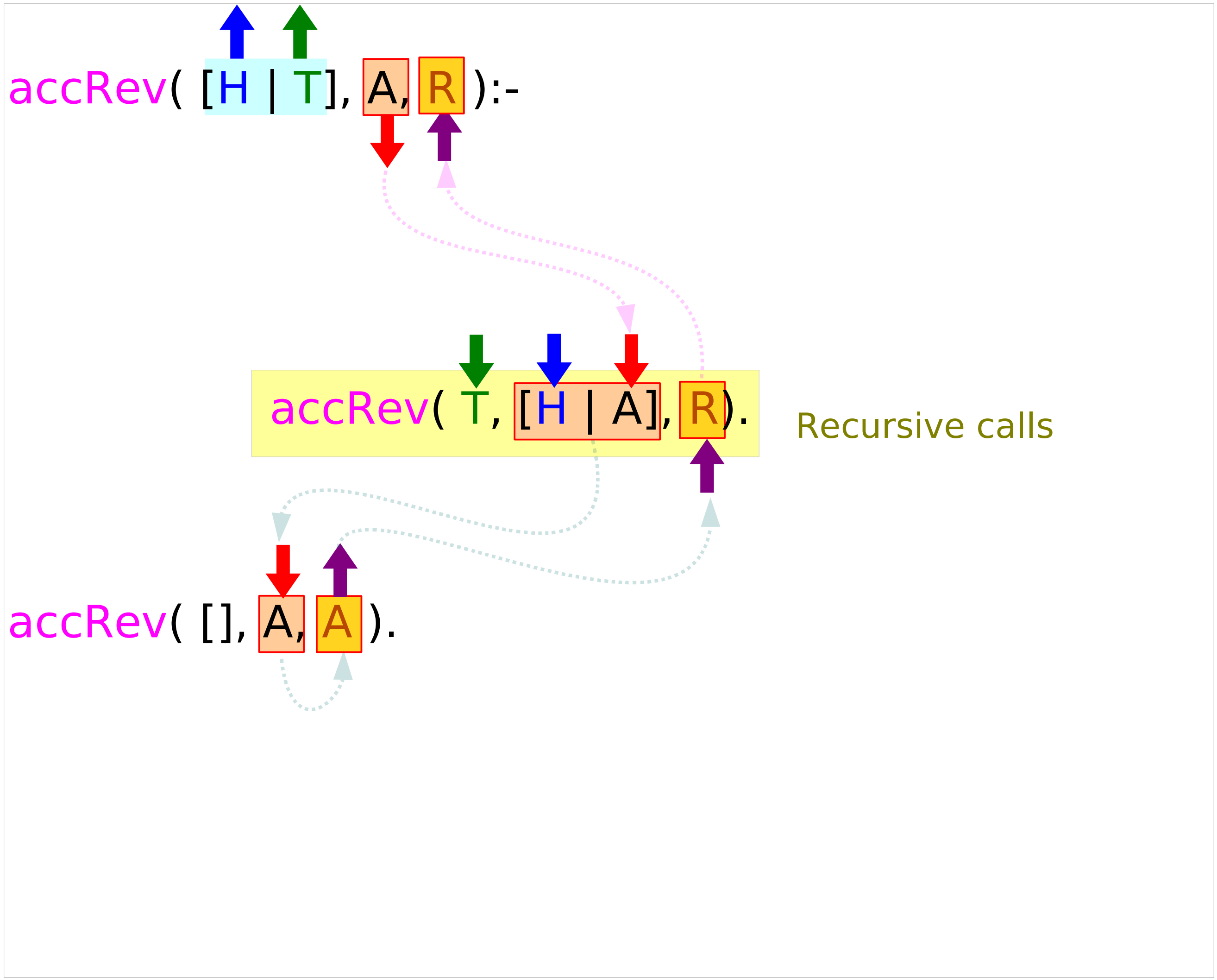
```
Append([], L, L).

Append( [H|T], L2, [H|L3])  :-

        Append(T, L2, L3).
```

Recursive calls

Naïve Reversing with Append

```
Naiverev ([], []).

Naiverev ([H|T], R):-

        Naiverev (T, RevT)   ,     Recursive calls

        Append(RevT, [H], R).
```

Reversing with an Accumulator



References

[1] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[2] http://www.learnprolognow.org/ Learn Prolog Now!