1 Accumulators

```
% reverse(L1,L2) holds iff L2 is the list L1 in reverse order
% Pre:   L1 is instantiated.

(1) reverse([],[]).
(2) reverse([A|X],Z) :- reverse(X,Y), append(Y,[A],Z).
```

Notice the use of `append` here. This solution is "slow" — quadratic in the length of the input list. See Figure 1 for the Prolog search tree of `reverse([1, 2, 3], L)`.

Using an accumulator variable can improve efficiency of our predicate.

```
(0) reverse(X,Z) :- reverse_acc(X,[],Z).

(1) reverse_acc([],Y,Y).
(2) reverse_acc([A|X],Y,Z) :- reverse_acc(X,[A|Y],Z).
```

Here `Y` is an *accumulator* variable. This solution is "fast" — linear in the length of the input list.

See Figure 2 for the Prolog search tree of `reverse([1, 2, 3], L)` with this solution.

And here is the trace of the same call:

```
?- reverse([1,2,3],L).
 Call: (7) reverse([1, 2, 3], _G293) ? creep
 Call: (8) reverse_acc([1, 2, 3], [], _G293) ? creep
 Call: (9) reverse_acc([2, 3], [1], _G293) ? creep
 Call: (10) reverse_acc([3], [2, 1], _G293) ? creep
 Call: (11) reverse_acc([], [3, 2, 1], _G293) ? creep
 Exit: (11) reverse_acc([], [3, 2, 1], [3, 2, 1]) ? creep
 Exit: (10) reverse_acc([3], [2, 1], [3, 2, 1]) ? creep
 Exit: (9) reverse_acc([2, 3], [1], [3, 2, 1]) ? creep
 Exit: (8) reverse_acc([1, 2, 3], [], [3, 2, 1]) ? creep
 Exit: (7) reverse([1, 2, 3], [3, 2, 1]) ? creep

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

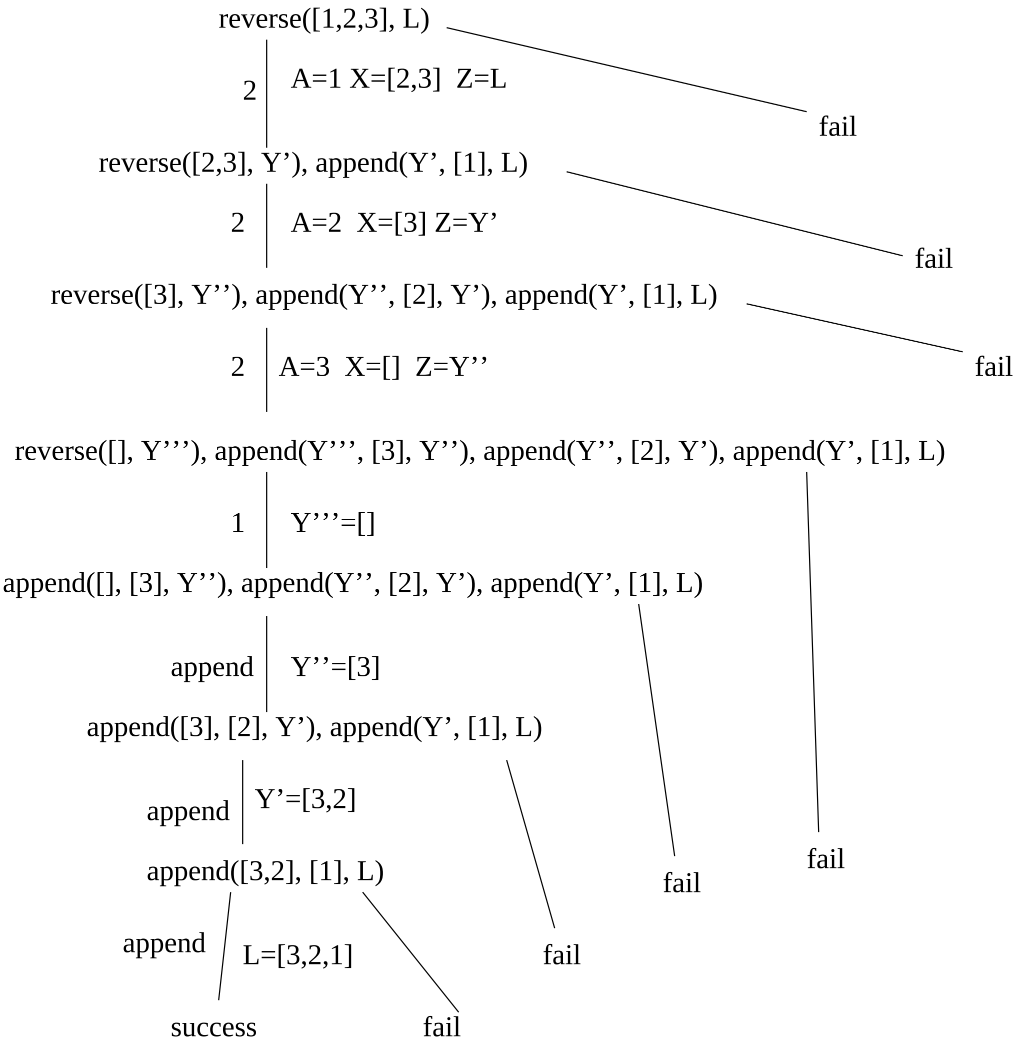

Figure 1: reverse([1,2,3], L)

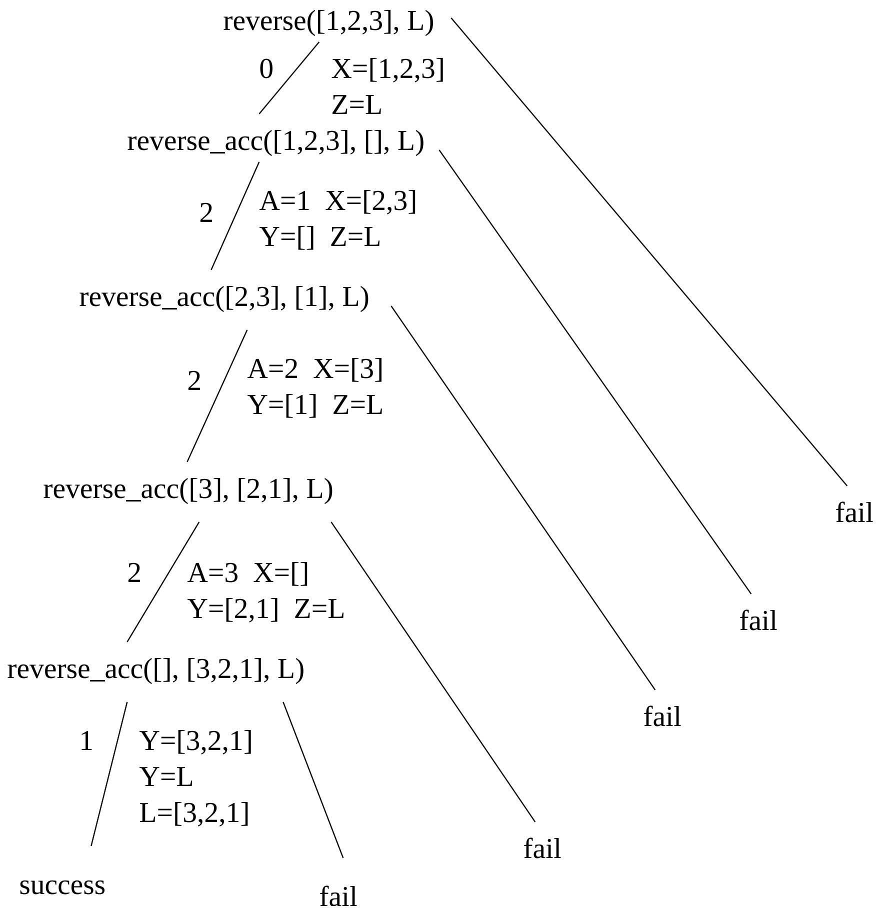

Figure 2: reverse([1,2,3], L) with an accumulator

Recall that we had a precondition that `L1` is instantiated. Why did we need that?

```
?- reverse(L,[1,2,3]).

L = [3, 2, 1] ;
ERROR: (user://1:50):
        Out of global stack
```

Something goes wrong when `L1` is not instantiated. Let's look at (part of) the trace of `reverse(L,[1,2,3])`.

```
?- reverse(L,[1,2,3]).
   Call: (7) reverse(_G292, [1, 2, 3]) ? creep
   Call: (8) reverse_acc(_G292, [], [1, 2, 3]) ? creep
   Call: (9) reverse_acc(_G352, [_G351], [1, 2, 3]) ? creep
   Call: (10) reverse_acc(_G358, [_G357, _G351], [1, 2, 3]) ? creep
   Call: (11) reverse_acc(_G364, [_G363, _G357, _G351], [1, 2, 3]) ? creep
   Exit: (11) reverse_acc([], [1, 2, 3], [1, 2, 3]) ? creep
   Exit: (10) reverse_acc([1], [2, 3], [1, 2, 3]) ? creep
   Exit: (9) reverse_acc([2, 1], [3], [1, 2, 3]) ? creep
   Exit: (8) reverse_acc([3, 2, 1], [], [1, 2, 3]) ? creep
   Exit: (7) reverse([3, 2, 1], [1, 2, 3]) ? creep

L = [3, 2, 1] ;
   Redo: (11) reverse_acc(_G364, [_G363, _G357, _G351], [1, 2, 3]) ? creep
   Call: (12) reverse_acc(_G370, [_G369, _G363, _G357, _G351],
              [1, 2, 3]) ? creep
   Call: (13) reverse_acc(_G376, [_G375, _G369, _G363, _G357, _G351],
              [1, 2, 3]) ? creep
   Call: (14) reverse_acc(_G382, [_G381, _G375, _G369, _G363, _G357, _G351],
              [1, 2, 3]) ? creep
   Call: (15) reverse_acc(_G388,
              [_G387, _G381, _G375, _G369, _G363, _G357, _G351],
              [1, 2, 3]) ? creep
   Call: (16) reverse_acc(_G394,
              [_G393, _G387, _G381, _G375, _G369, _G363, _G357, _G351],
              [1, 2, 3]) ? creep
   Call: (17) reverse_acc(_G400,
              [_G399, _G393, _G387, _G381, _G375, _G369, _G363, _G357|...],
              [1, 2, 3]) ?
```

Thus, we need a precondition: the first list is instantiated to a list of known length, i.e., the tail is [], and not an uninstantiated variable.

The following implementation works with any input:

```
reverse(L,R) :- knownlength(L), reverse_acc(L,[],R).
reverse(L,R) :- \+knownlength(L), reverse_acc(R,[],L).

knownlength(L) :- \+var(L), L=[].
knownlength(L) :- \+var(L), L=[_|R], knownlength(R).

reverse_acc([],Y,Y).
reverse_acc([A|X],Y,Z) :- reverse_acc(X,[A|Y],Z).
```

Exercise: draw a search tree and examine the trace for `reverse(L,[1,2,3])` to see how it now works.