Appending lists in SCHEME and in Prolog

(define append (lambda (X Y)
    (cond ((null? X) Y)
          (#t (cons (car X) (append (cdr X) Y)))))
))


\[
\begin{align*}
& [X | W] \\
& \quad [X | Y] \\
& \quad \quad X \\
& \quad \quad Y \\
& \quad \quad Z \\
& \quad \quad W
\end{align*}
\]


?- append([a,b], [c], L).

rule [2]

\[ X_1 \rightarrow a \quad Y_1 \rightarrow [b] \quad Z_1 \rightarrow [c] \quad L \rightarrow [a \mid W_1] \]

append([b], [c], W_1).  rule [2]

\[ X_2 \rightarrow b \quad Y_2 \rightarrow [] \quad Z_2 \rightarrow [c] \quad W_1 \rightarrow [b \mid W_2] \]

append([], [c], W_2).  rule [1]

\[ U_3 \rightarrow [c] \quad W_2 \rightarrow [c] \]

<empty goal list>  SUCCESS

What happened to L?

L was bound to \([a \mid W_1]\)

\[ W_1 \text{ was bound to } [b \mid W_2] \]

\[ W_2 \text{ was bound to } [c] \]

So L was bound to \([a \mid [b \mid [c]]] = [a, b, c]\)
Input parameters may become output parameters


?- append(V, [b, c], [a, b, c]). rule [2]
    V → [a | Y₁] Z₁ → [b, c] X₁ → a W₁ → [b, c]
    append(Y₁, [b, c], [b, c]). rule [1]
    Y₁ → [] U₂ → [b, c]
so V was bound to [a | Y₁]
and Y₁ was bound to []
so V was ultimately bound to [a | []] = [a]

Exercise: Verify that the goal

    append(V, [b, c], [b, c, a]).

will FAIL.
More input parameters become output parameters


?- append(L, M, [a,b,c]). rule [1]
   L → []  M = U₁ → [a,b,c]

After the ‘‘Yes’’ response type ‘‘;’’ for more answers:

?- append(L, M, [a,b,c]). use rule [2] this time
   L → [a | Y₁]  M → Z₁  X₁ → a  W₁ → [b,c]

append(Y₁, Z₁, [b,c]). rule [1]
   Y₁ → []  Z₁ = U₂ → [b,c]

so in the second answer

   L is bound to [a | Y₁]  and  Y₁ is bound to []
   M and Z₁ are bound to [b,c]

   thus we have  L = [a | []] = [a],  &  M = [b,c]

What are the other answers that Prolog will find?