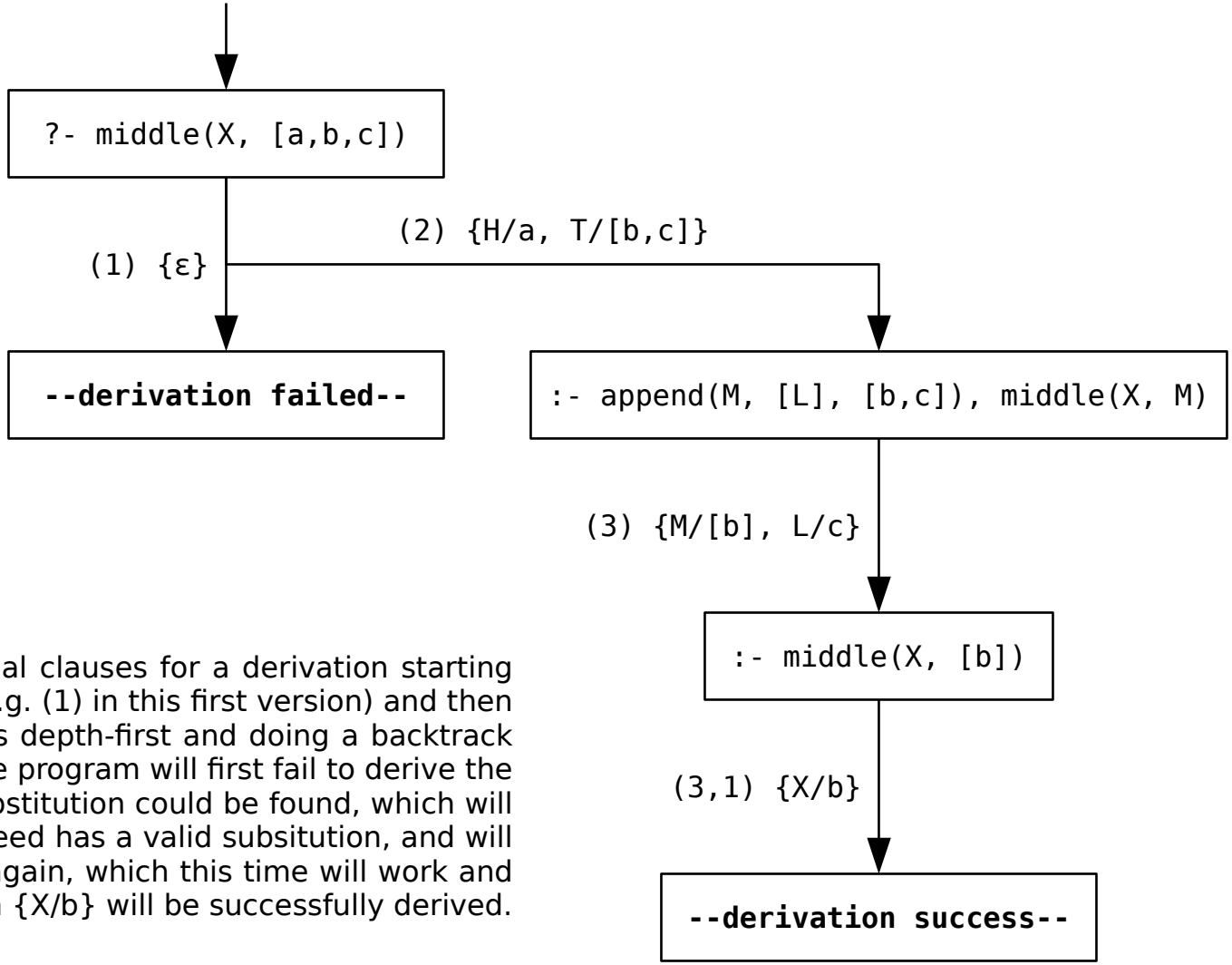
Program Version 1:

```
(1) middle(X, [X]).
(2) middle(X, [H|T]) :-
(3)   append(M, [L], T),
(4)   middle(X, M).
```

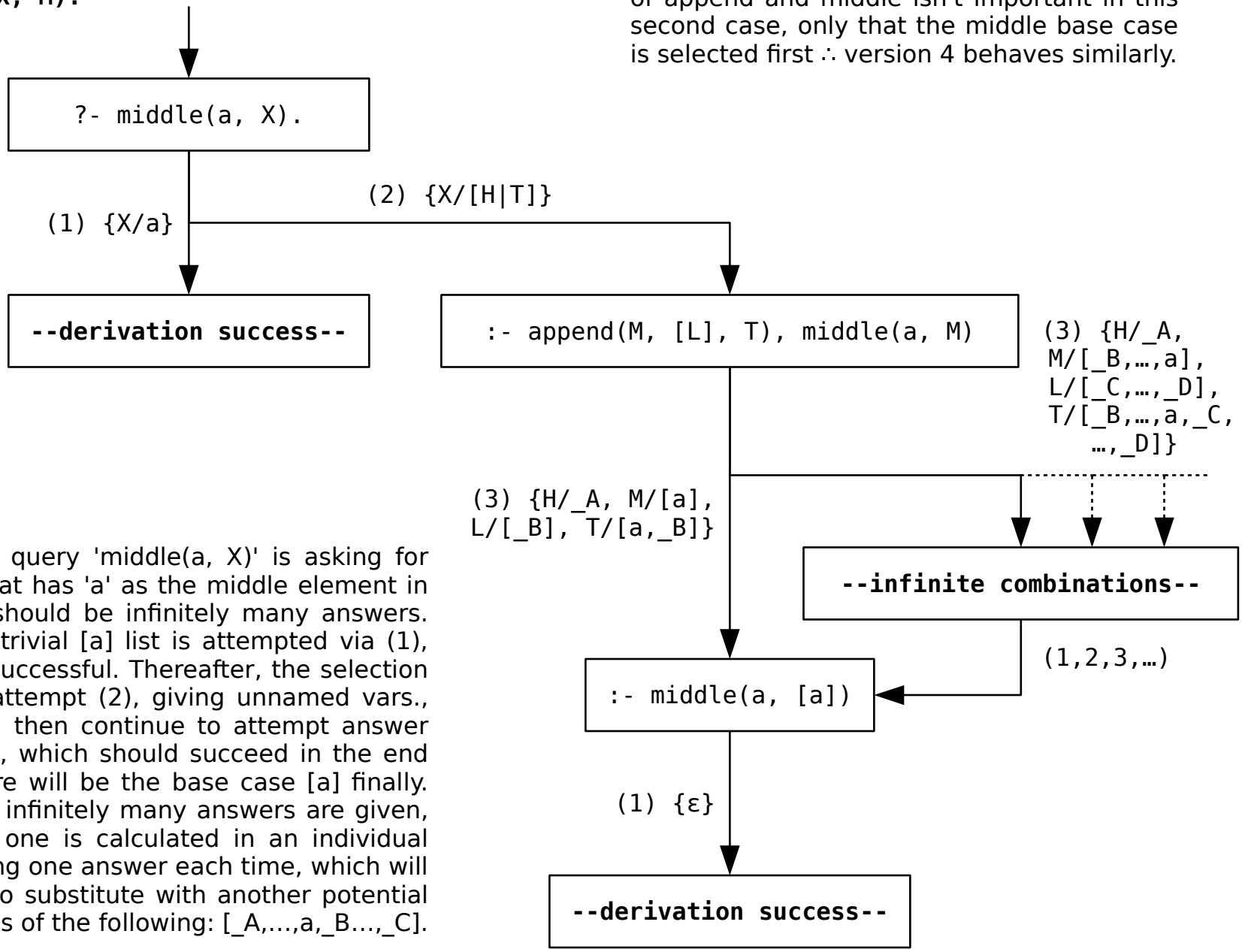


Since prolog selects potential clauses for a derivation starting from the top-most clause (e.g. (1) in this first version) and then traverses potential solutions depth-first and doing a backtrack if needed. Our version of the program will first fail to derive the (1) clause since no valid substitution could be found, which will then attempt (2), which indeed has a valid subsitution, and will continue to then apply (1) again, which this time will work and the solution/goal subsitution {X/b} will be successfully derived.

Program Version 1:

```
(1) middle(X, [X]).
(2) middle(X, [H|T]) :-
(3)   append(M, [L], T),
(4)   middle(X, M).
```

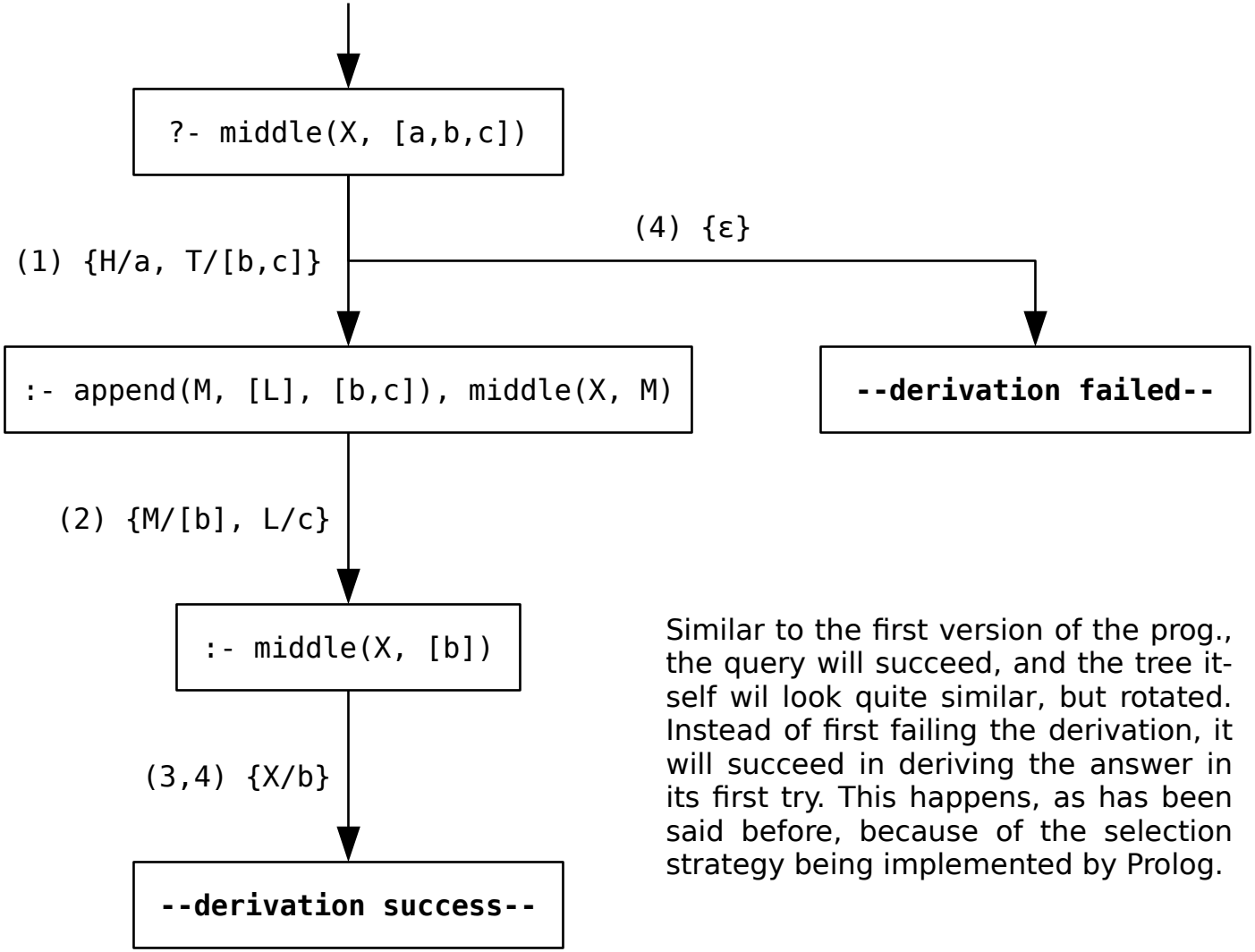
Since version 4 of the program is very similar it will also be handled here, the actual order of append and middle isn't important in this second case, only that the middle base case is selected first ∴ version 4 behaves similarly.



Since the query 'middle(a, X)' is asking for the list that has 'a' as the middle element in it, there should be infinitely many answers. First, the trivial [a] list is attempted via (1), which is successful. Thereafter, the selection rule will attempt (2), giving unnamed vars., which will then continue to attempt answer the query, which should succeed in the end since there will be the base case [a] finally. Note that infinitely many answers are given, but each one is calculated in an individual step, giving one answer each time, which will then try to substitute with another potential list, always of the following: [_A,…,a,_B…,_C].

Program Version 2:

```
(1) middle(X, [H|T]) :-
(2)   append(M, [L], T),
(3)   middle(X, M).
(4) middle(X, [X]).
```

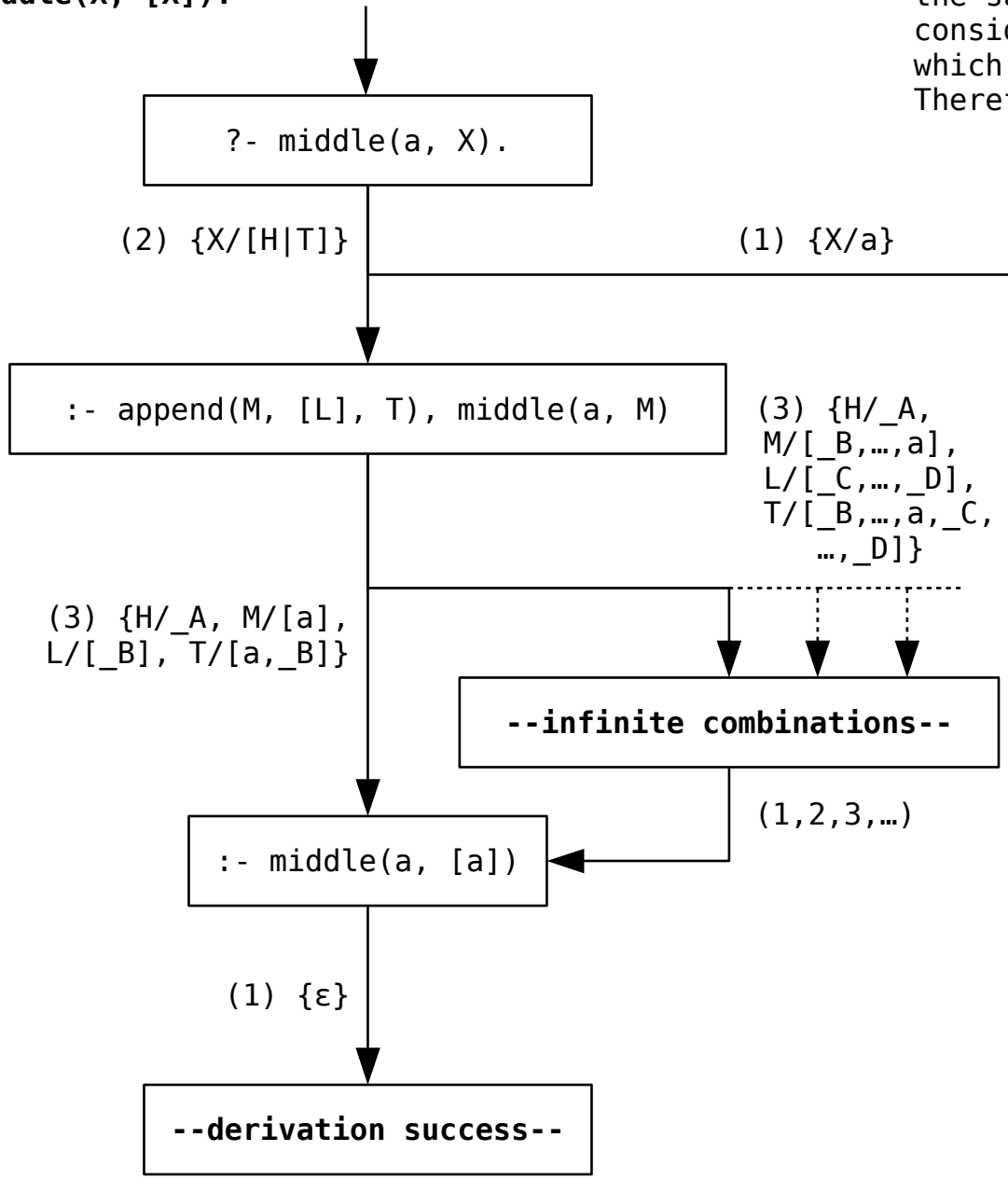


Similar to the first version of the prog., the query will succeed, and the tree itself wil look quite similar, but rotated. Instead of first failing the derivation, it will succeed in deriving the answer in its first try. This happens, as has been said before, because of the selection strategy being implemented by Prolog.

Program Version 2:

```
(1) middle(X, [H|T]) :-
(2)   append(M, [L], T),
(3)   middle(X, M).
(4) middle(X, [X]).
```

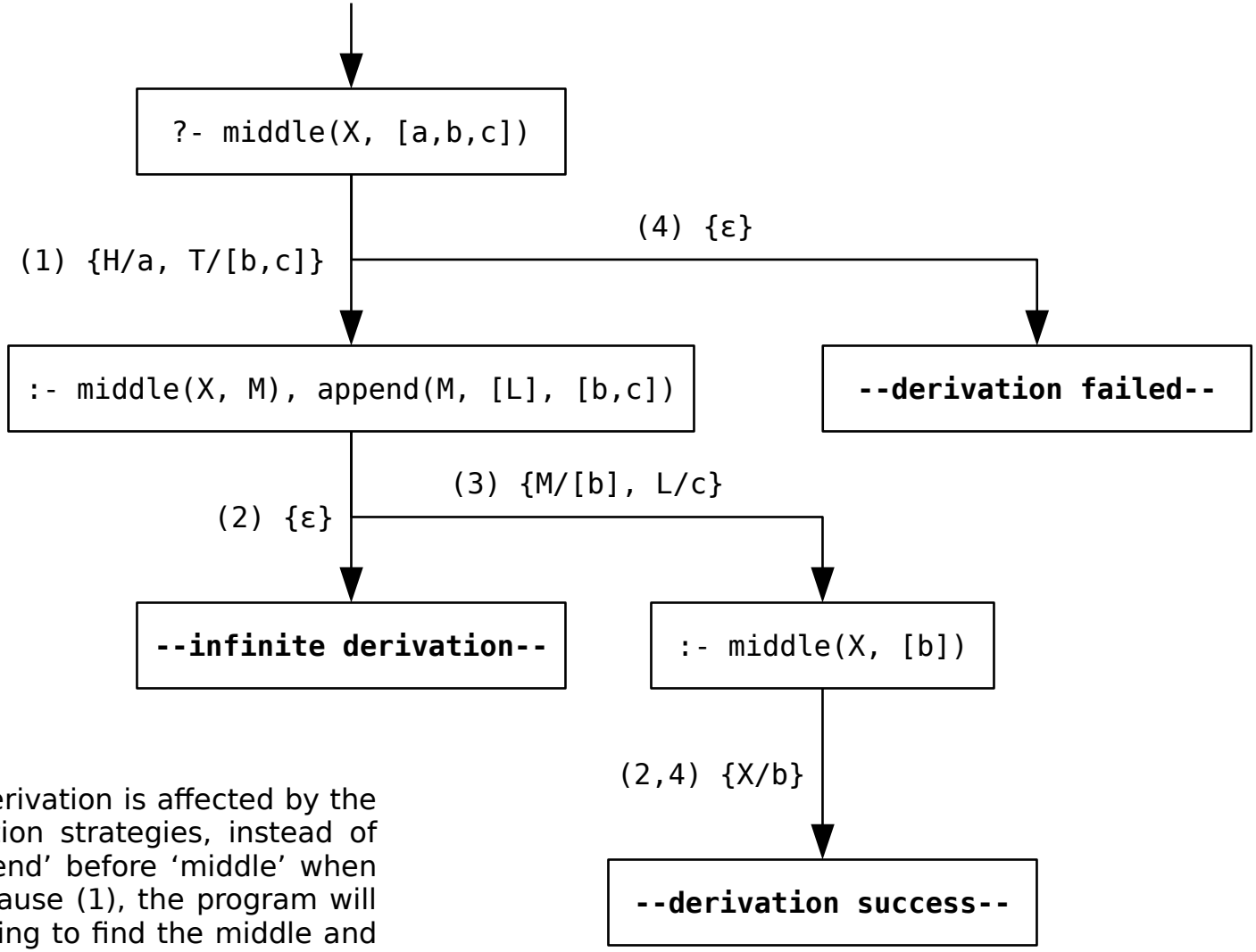
Here however, version 3 of the program which is quite similar, will not get the same behavior, since it will never consider the anonymous variable subst. which would otherwise have happened. Therefore, version 3 won't terminate.



Unlike the first version of the program, this will not consider the [a] expansion, first, which will result in it never being considered as a potential solution for X since there will be, as in version 1, an infinite number of combinations for the substitution of H and T. However, the program will still respond with answers but will always be: [_A,...,a,_B,...,_C], and trivial solution [a] won't be found...

Program Version 3:

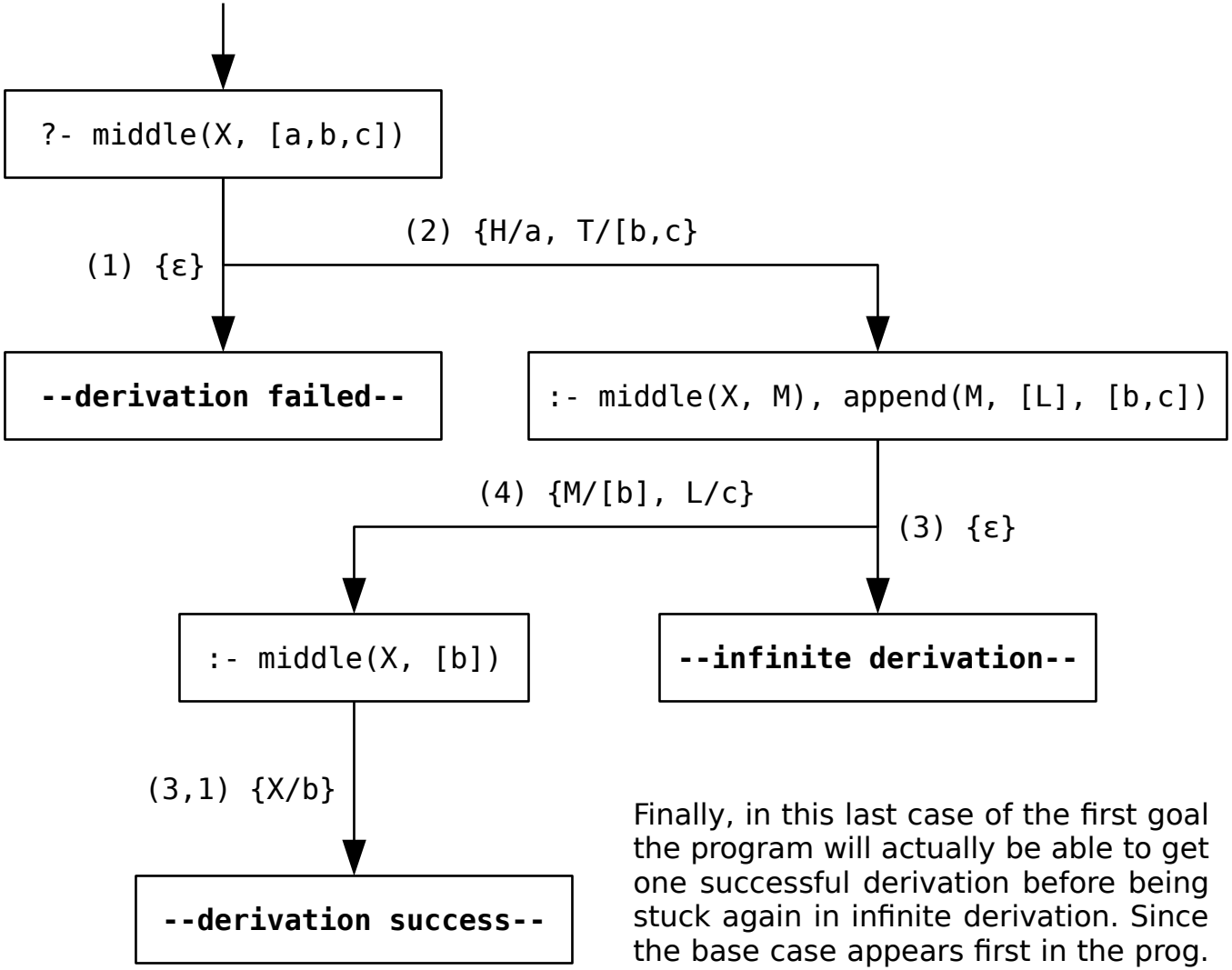
```
(1) middle(X, [H|T]) :-
(2)   middle(X, M),
(3)   append(M, [L], T).
(4) middle(X, [X]).
```



Again, the derivation is affected by the Prolog selection strategies, instead of picking ‘append’ before ‘middle’ when expanding clause (1), the program will find itself trying to find the middle and with no additional information gained. Look at program version 1 and 2, when the ‘append’ was before, allowing the derivation to actually resolve variables.

Program Version 4:

```
(1) middle(X, [X]).
(2) middle(X, [H|T]) :-
(3)   middle(X, M),
(4)   append(M, [L], T).
```



Finally, in this last case of the first goal the program will actually be able to get one successful derivation before being stuck again in infinite derivation. Since the base case appears first in the prog. it will be evaluated first, and ‘append’ will already have been substituted, the base case can then just plug values in.