Theory of Multi-Paradigm Programming Languages

Lecture 5-2

Chapter 3 ML Programming Language

Overview

Prof. Moonkun Lee, Ph.D.

Division of Computer Science and Engineering
Jeonbuk National University, Republic of Korea

Contents

- **제3장 ML**
  - 3.1 ML언어의 특징
  - 3.2 Basic
  - **3.3 Function**
  - 3.4 Input/Output
  - 3.5 More Functions
  - 3.6 사용자 정의 유형 (User Defined Type)
  - 3.7 More Data Structure
  - 3.8 Structure: ML Module System
  - 3.9 결어

3.3 FUNCTION

```
fun <identifier> ( <first pattern> ) = <first expression>
 |  <identifier> ( <second pattern> ) = <second expression>
 |   . . .
 |  <identifier> ( <last pattern> ) = <last expression>
```

```
1  fun square(x:int) = x*x;
2  val square = fn : int -> int
```

<프로그램 3-20> square Example

```
fun square x229) = (x:int)*x;
```

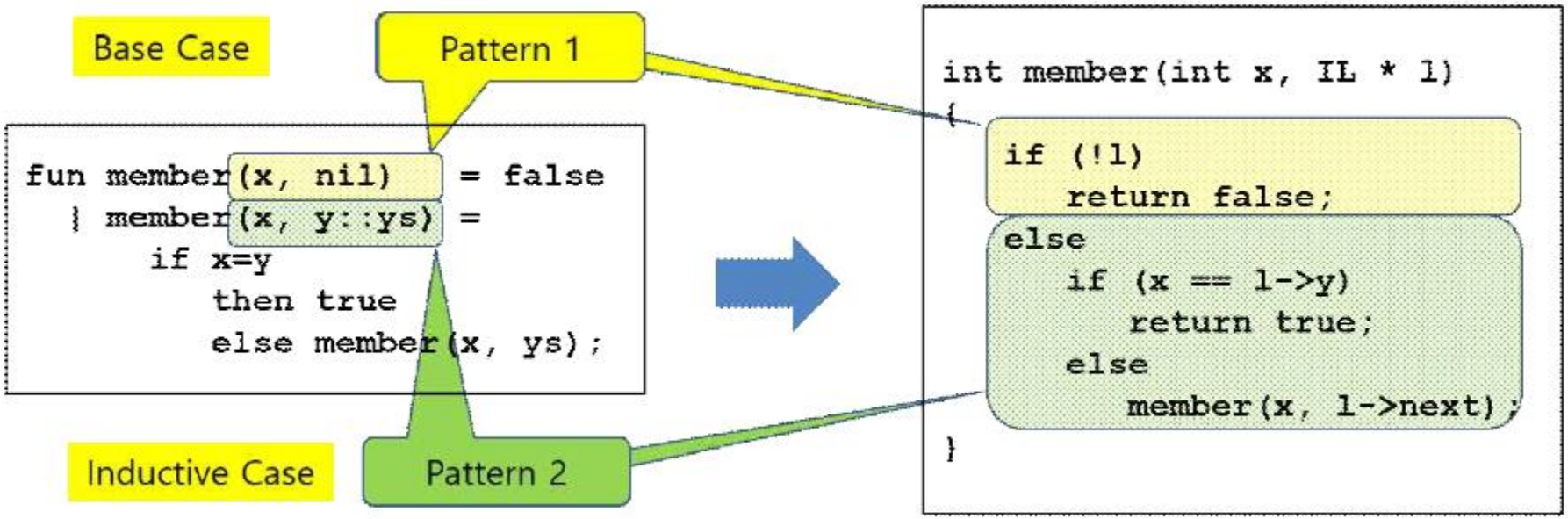
<프로그램 3-20-1> square Example

```
1  square(3);
2  val it = 9 : int
```

<프로그램 3-21> square Example

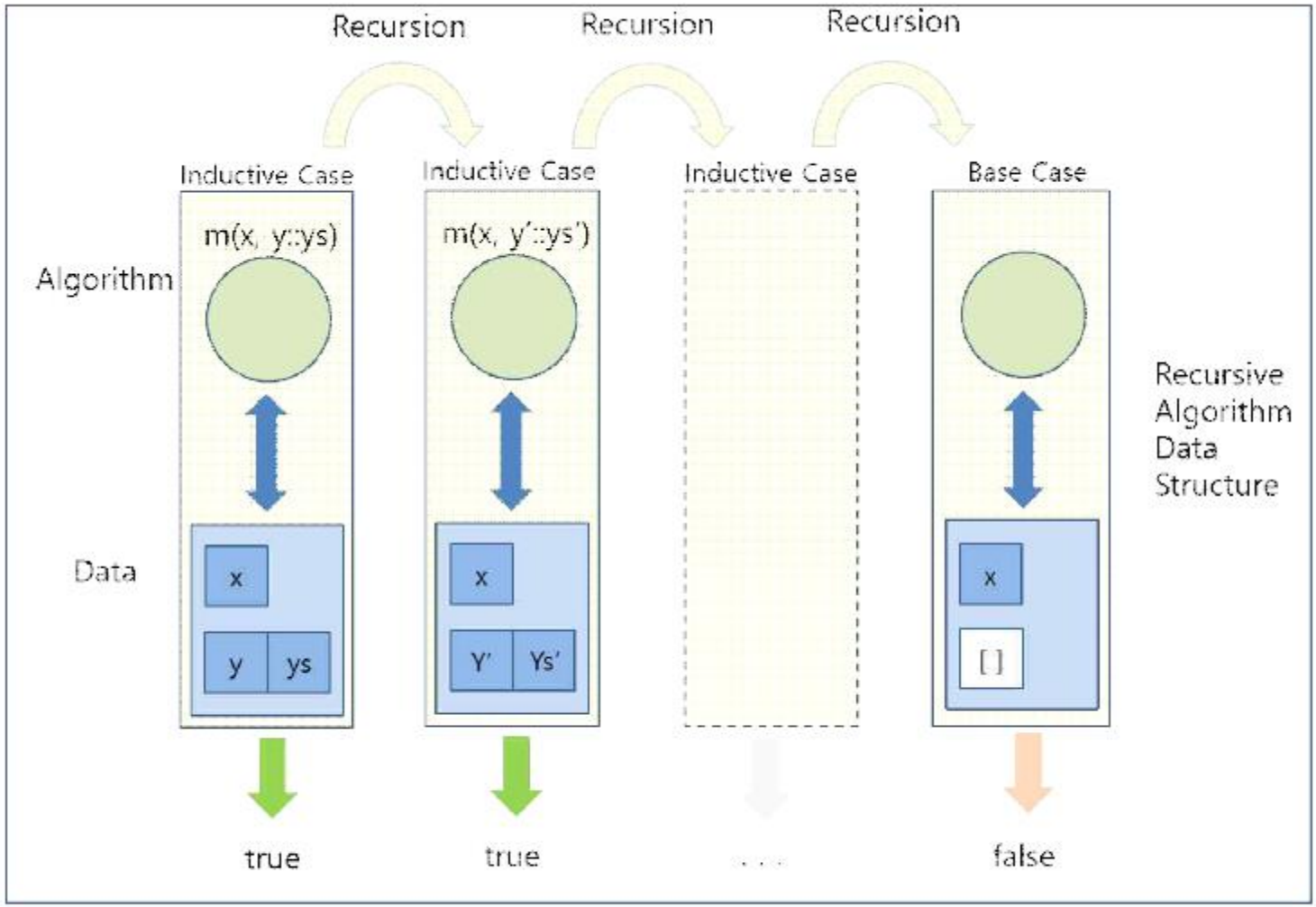
```
1  fun member(x, nil)   = false
2    | member(x, y::ys) =
3        if x=y then true
4        else member(x, ys);
5  val member= fn : ''a * ''a list -> bool
```

<프로그램 3-22> member Example



<그림 3-8> Patterns

```
struct Integer_List
{
   int x;
   struct Integer_List * nest;
}
typedef struct Integer List IL;
```



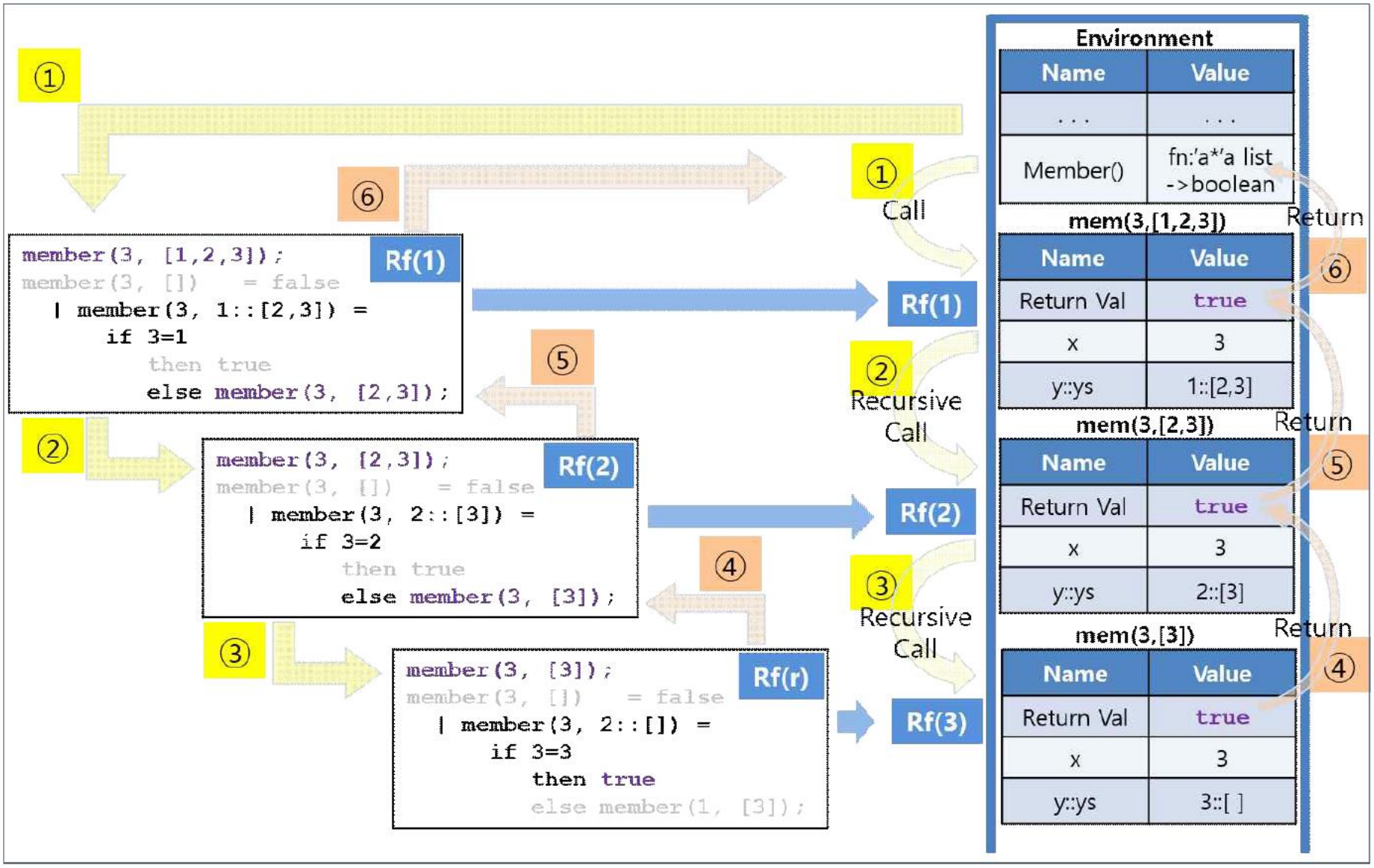
<그림 3-9> rAD 구조: member 함수

```
1  fun member(x, nil)    = false
2    | member(x, x::ys) = true                    x=x
3    | member(x, y::ys) = member(x, ys);          x=y
```

<프로그램 3-23> member Example

```
1  member(3,[1,2,3]);
2  val it = true : bool
```

<프로그램 3-24> member Example

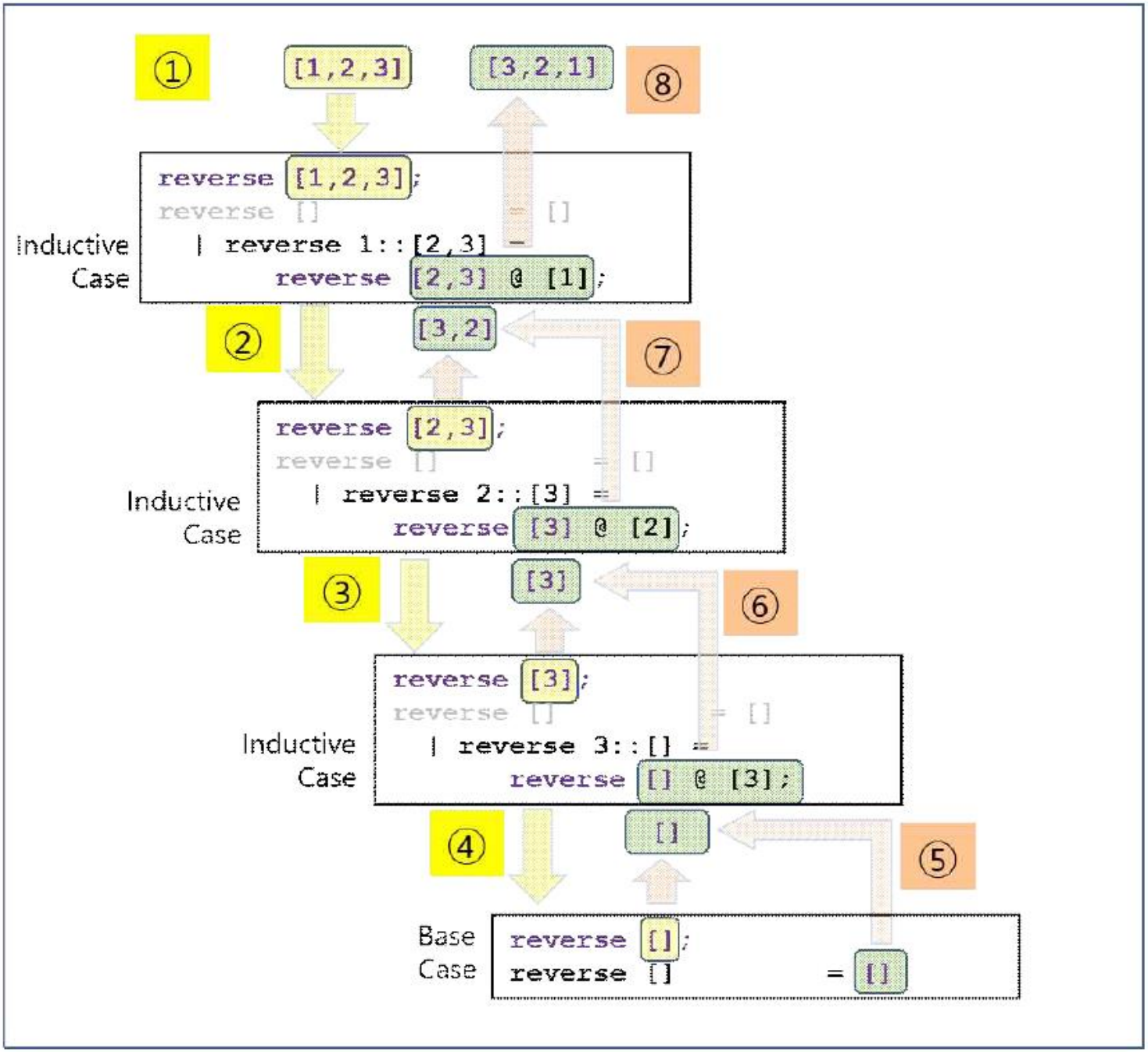

①
⑥
member(3, [1,2,3]);
member(3, []) = false
| member(3, 1::[2,3]) =
if 3=1
then true
else member(3, [2,3]);
Rf(1)
①
Call
Rf(1)
⑤
②
member(3, [2,3]);
member(3, []) = false
| member(3, 2::[3]) =
if 3=2
then true
else member(3, [3]);
Rf(2)
②
Recursive
Call
Rf(2)
④
③
member(3, [3]);
member(3, []) = false
| member(3, 2::[]) =
if 3=3
then true
else member(1, [3]);
Rf(r)
③
Recursive
Call
Rf(3)
Environment
Name
Value
. . .
. . .
Member()
fn:'a*'a list
->boolean
mem(3,[1,2,3])
Name
Value
Return Val
true
x
3
y::ys
1::[2,3]
Return
⑥
mem(3,[2,3])
Name
Value
Return Val
true
x
3
y::ys
2::[3]
Return
⑤
mem(3,[3])
Name
Value
Return Val
true
x
3
y::ys
3::[ ]
Return
④

```
1  fun reverse([])     = []
2    |  reverse(x::xs) =
3        reverse (xs) @ [x];
4  val reverse = fn : 'a list -> 'a list
```

<프로그램 3-25> reverse Example

```
1  reverse([1,2,3]);
2  val it = [3,2,1] : int list
```

<프로그램 3-26> reverse Example



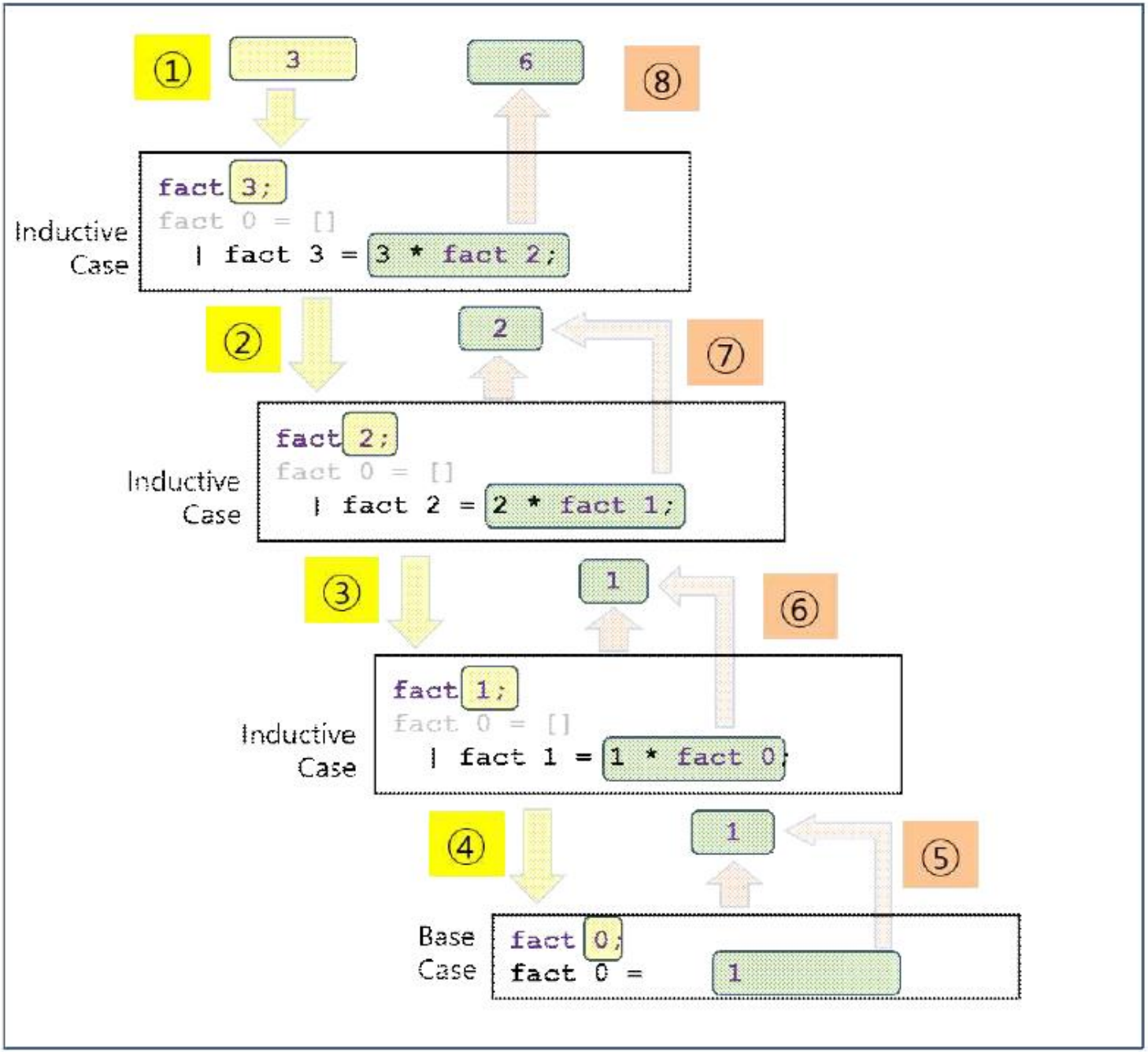
<그림 3-11> Execution for `reverse [1,2,3]`

```
1  fun fact 0 = 1
2    | fact n = n*fact(n-1);
3  val fact = fn : int -> int
```

<프로그램 3-27> **fact** Example

```
1  fact 3;
2  val it = 6 : int
```

<프로그램 3-28> **fact** Example

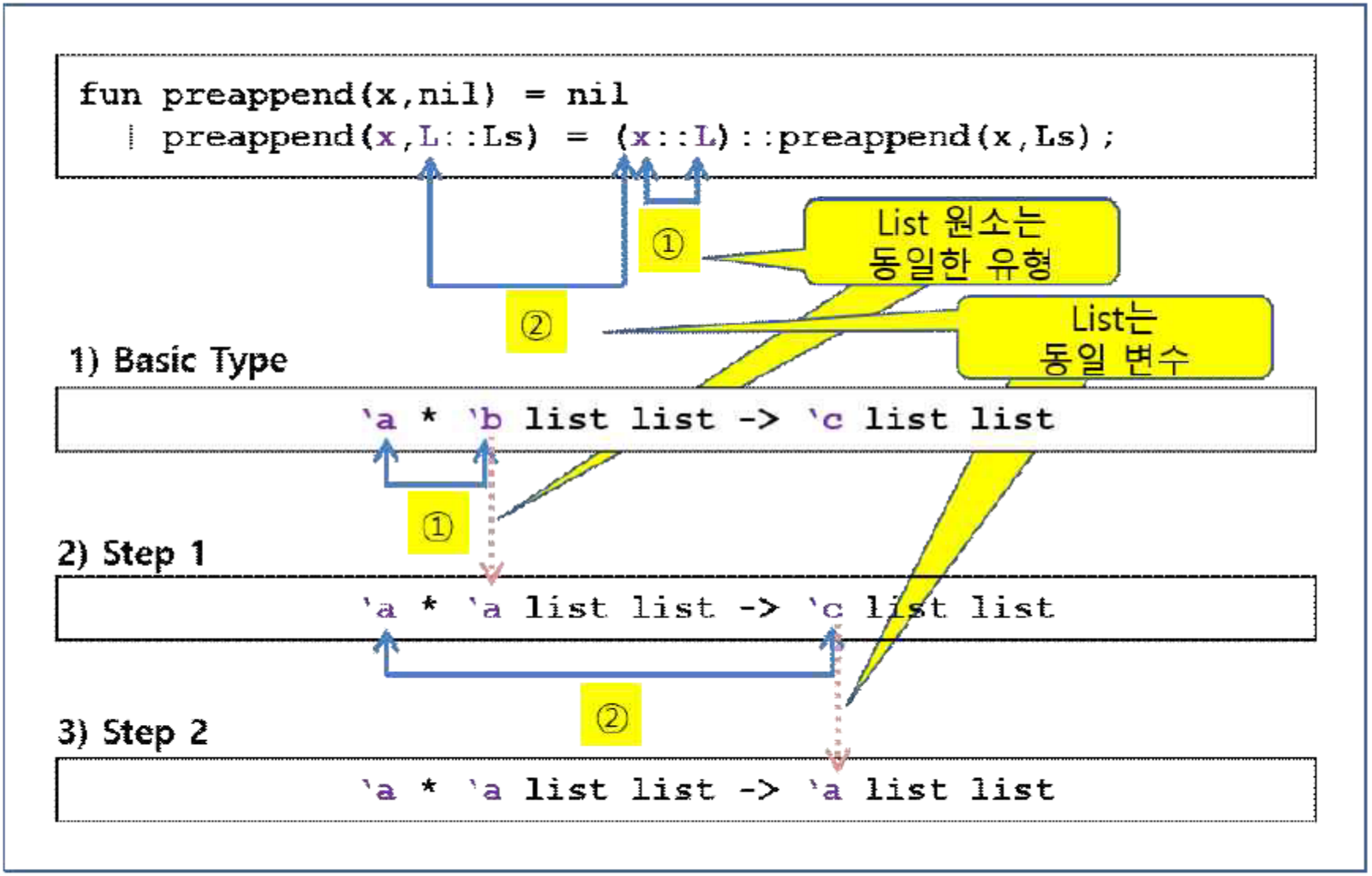


<그림 3-12> Execution for **fact 3**

```
let
   val <1st variable> = <1st expression>;
   val <2nd variable> = <2nd expression>;
   . . .
   val <last variable> = <last expression>
in
   <expression>
end
```

```
1  fun preappend(x,nil) = nil
2    | preappend(x,L::Ls) = (x::L)::preappend(x,Ls);
3  val preappend = fn : 'a * 'a list list  -> 'a list list
```

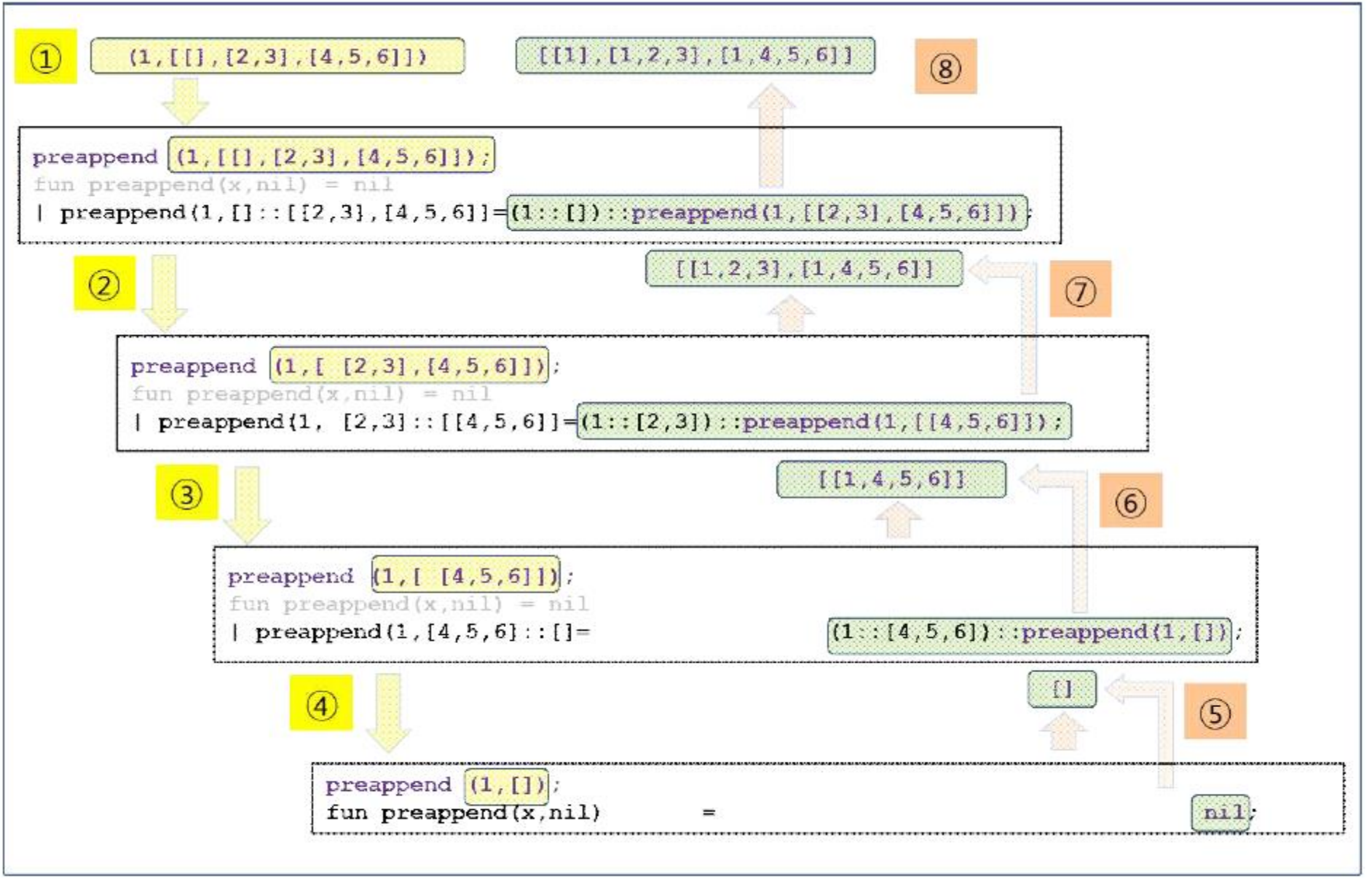
<프로그램 3-29> **preappend** Example



<그림 3-13> Type Checking for **preappend**

```
1  preappend(1, [[],[2,3], [4,5,6]]);
2  val it = [[1], [[1,2,3], [1,4,5,6]] : int list list
```

<프로그램 3-30> **preappend** Example



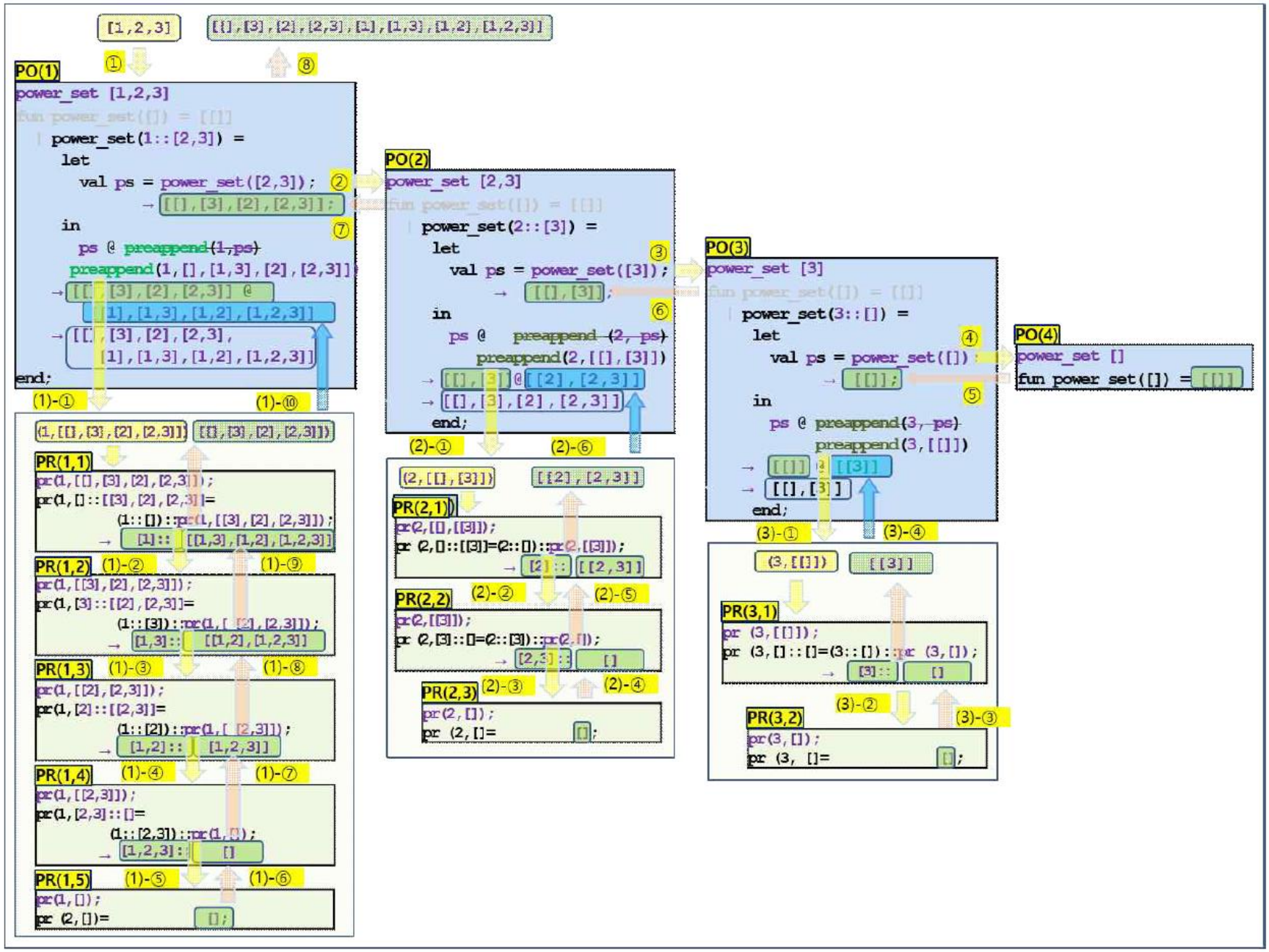
<그림 3-14> Execution for **preappend(1, [[],[2,3],[4,5,6]])**

```
1  fun power_set([]) = [[]]
2    | power_set(x::xs) =
3       let
4          val ps = power_set(xs);
5       in
6          ps @ preappend(x,ps)
7       end;
8  val power set = fn : 'a list -> 'a list list
```

<프로그램 3-31> **power_set** Example

```
1  power_set([1,2,3]);
2  val it = [[],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]] : int list list
```

<프로그램 3-32> **power_set** Example



<그림 3-15> Execution for power_set ([1,2,3])