

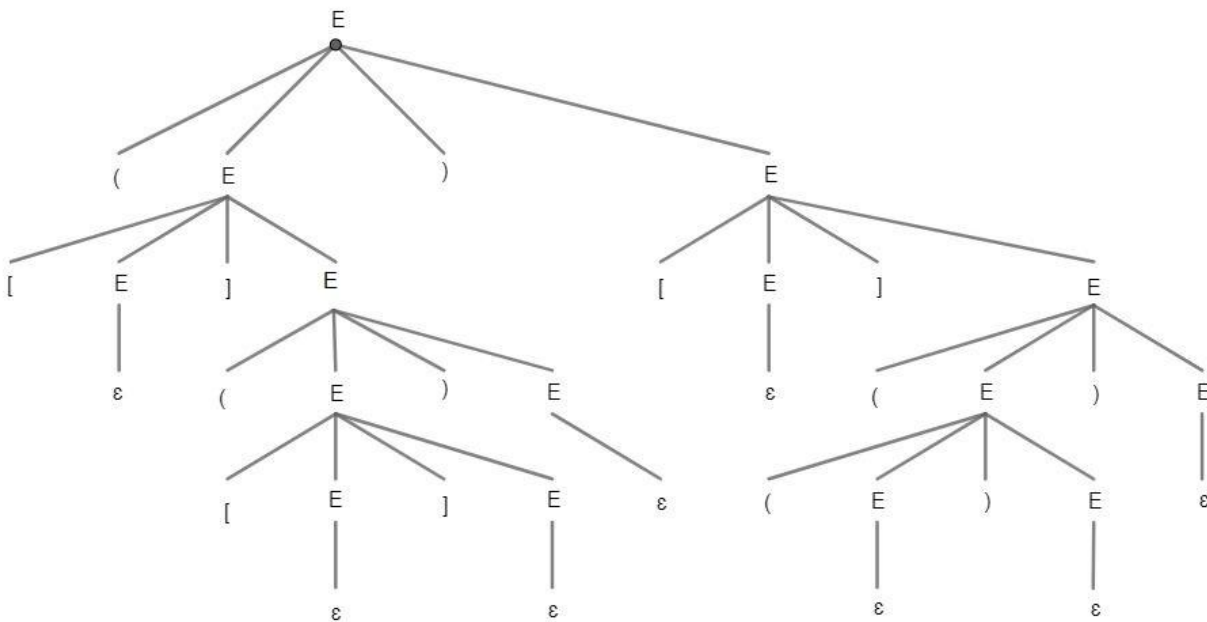
2.14 (a) LL:  $E \rightarrow (E)E \mid [E]E \mid \{E\}E \mid \varepsilon$   
 LR:  $E \rightarrow E(E) \mid E[E] \mid E\{E\} \mid \varepsilon$

(b)

	(	[	{	)	]	}	$\epsilon$
E	(E)E	[E]E	{E}E	-	-	-	$\epsilon$

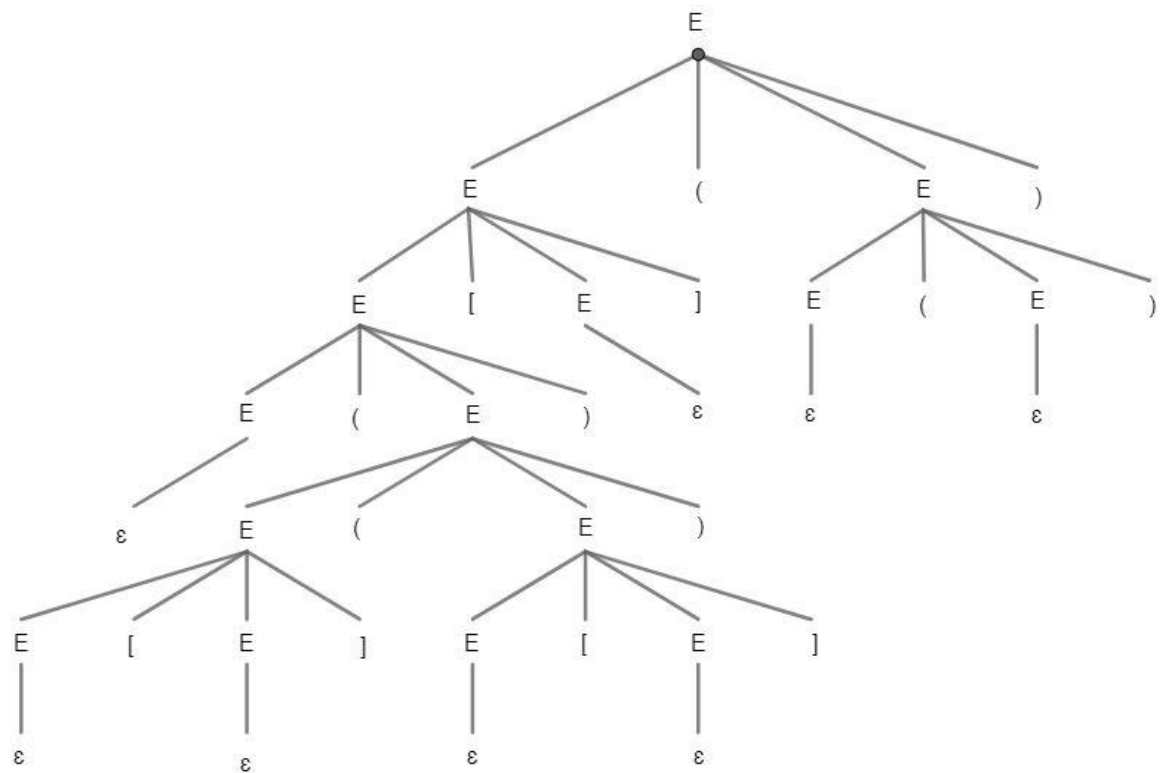
	(	[	{	)	]	}	$\epsilon$
E	E(E)	E[E]	E{E}	-	-	-	$\epsilon$

(c) LL Tree:



(d)  $E \rightarrow (E \rightarrow ([E \rightarrow (\square E \rightarrow (\square(E \rightarrow (\square([E \rightarrow (\square(\square E \rightarrow (\square(\square)E \rightarrow (\square(\square))E \rightarrow$   
 $(\square(\square))[E \rightarrow (\square(\square))\square E \rightarrow (\square(\square))\square(E \rightarrow (\square(\square))\square(E \rightarrow (\square(\square))\square(())E \rightarrow (\square(\square))\square(())$   
 $(\square(\square))\square(())$

## LR Tree



(d) E -> E) -> E)) -> E()) -> E(()) -> E[]() -> E[][]() -> E)))[]() -> E]][])[]() -> E[[[])][]() -> E[[]()][]() -> E([[]])[]() -> E([[]][[]])[]()

- 3.7 (a) When he calls the reverse function, it creates a new list and the original unreversed list remains loaded in memory, causing a leak.
- (b) The reverse function changes the pointer passed to it to point to the tail of the newly reversed list, so when he tries `delete_list(L)`, it deletes the reversed list he just created. Then, after assigning `L = T`, `L` points to the deallocated tail of the list he just deleted.

3.14 Static: 1122 Dynamic: 1121, with a static scope, every time set\_x(n) is called the global variable x is changed, so the final print\_x() will reflect that the second() function changed x to 2. With the dynamic scope, after second() is called only the local x is changed to 2, so the final print\_x() call prints the global variable x which is still 1.