1. The following grammar generates some of the types of ML:

\[
\text{Type} ::= \text{TVar} \mid \text{Type} \rightarrow \text{Type} \mid \text{Type} \ast \text{Type}
\]

Note: \text{TVar} and \text{Type} are non-terminals. “→” and “∗” are terminals. \text{TVar} generates a set of names (the \text{Type} variables) “α”, “β”, “γ”, etc. The productions for \text{TVar} are not represented here.

1. Show that the grammar is ambiguous.

2. Modify it so to obtain an unambiguous grammar, generating the same language, by enforcing the following properties: → is right associative, * has higher priority than →,* is left associative
2. Define a grammar which generates all palindromes over the alphabet \{a, b, c\}. Note: a palindrome is any string which reads the same backward as forward. For example, the empty string, “a”, “aa”, “aba”, and “abba” are palindromes, whereas “ab”, “abb”, and “abc” are not.
3. Write a Scheme program that will run successfully, but whose ML corresponding program will produce an error.
4. Write a Scheme tail-recursive procedure to compute the length of an arbitrary list.
5 Draw the internal representation of the following Scheme expressions:

\[
(((((1))))
\]

\[
(1 \ (2 \ (3 \ 4)) \ (5))
\]
6. Consider the following data structure “binary tree”:

datatype 'a btree = emptybt | consbt of 'a * 'a btree * 'a btree;

- Define a function mirror : 'a btree → 'a btree which returns the “mirror” of the input tree. Example:

```
    1
  / \ 1
 /  
2 3 3 2
 / \ /  
4 5 5 4
 \   / 
 6 6
```

- Define a function breadth_first : 'a btree → 'a list which takes a binary tree and returns a list representing the breadth first visit of the tree. Example:

```
    1
  /  
2 3
 /  
4 5 6 5, 4, 3, 2, 1
 /  
7 8
 / 
9
```
7. Write in ML a higher-order function \textit{incn} that takes an integer \( n \) as a parameter and returns an \( n \)-th increment function, which increments its parameter by \( n \). For example, \((\text{incn} 3) 2 = 5\) and \((\text{incn} -2) 3) = 1\).
8. Give the ML types of the following functions:

\[
\text{fun } f (g, h) = g(h) + 2
\]

\[
\text{fun } f x y z = x + y + z ;
\]

\[
\text{fun } f (g, h, x) = g(h(x)) ;
\]

\[
\text{fun } g(n, f) = f n ;
\]
9. Using one example to explain the difference between call-by-value and call-by-name.