Problem 1: Context Free Grammars

Consider the following Grammar:

\[ E \rightarrow aSSc \]
\[ S \rightarrow aSb | bSc | T \]
\[ T \rightarrow a | b | c \]

(a) Is this an ambiguous grammar? [2 pts]

\[ \text{A} \ ] \text{Yes} \quad \text{B} \ ] \text{No} \]

(b) If you believe it to be ambiguous, prove it, otherwise derive "aaabbc" [6 pts]

Problem 2: Lexing Parsing and evaluating

Given the following CFG, and assuming strong, static typing as is used in OCaml, at what stage of language processing would the nearby expressions fail? Mark 'Valid' if the expression would be accepted by the grammar and type checker.

\[ E \rightarrow M \text{ and } E|M \text{ or } E|M \]
\[ M \rightarrow N \cdot M | N - M | N \]
\[ N \rightarrow 1 | 2 | 3 | 4 | \text{true} | \text{false} \]|(E) \]

Hint: Pay careful attention to the terminal symbols allowed in the grammar.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Lexer</th>
<th>Parser</th>
<th>Evaluator</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + 2 - (true and false)</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
<tr>
<td>true + (3 - 2)</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
<tr>
<td>3 * 1 - 2</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
<tr>
<td>2 - 1 + 4</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
<tr>
<td>)2 or + -</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
<tr>
<td>true</td>
<td>L</td>
<td>P</td>
<td>E</td>
<td>V</td>
</tr>
</tbody>
</table>
Problem 3: OCaml Higher Order Functions

Complete the skeleton code below which defines a simplified version of partition which takes a single "pivot value" and a list. It returns a pair of lists, the first with elements below the pivot value, the second with elements equal to or above the pivot value. The lists returned can have elements from the original list in any order (forward, reverse, other).

(* Definition for fold_left *)
let rec fold_left f a lst =
  match lst with
  [] -> a
  |x::t -> fold_left f (f a x) t

EXAMPLES:

# let partition pivot lst = ...;;
val partition : 'a -> 'a list -> 'a list * 'a list = <fun>

# partition 5 [12; 2; 9; 7; 6; 5; 1; 4];;
- : int list * int list = ([4; 1; 2], [5; 6; 7; 9; 12])

(* below 5 ... equal/above 5 *)

# partition "c" ["banana"; "grape"; "carrot"; "pear"; "apple"];;
- : string list * string list = (["apple"; "banana"], ["pear"; "carrot"; "grape"])

(* below "c" ... equal/above "c" *)

(*'a -> 'a list -> ('a list * 'a list)*)
let partition pivot lst =
  let helper acc x =
    in fold_left helper __________ lst