Midterm Solutions

Brevity was not needed for the regular expressions, just correctness.

Problem 1a

Regular Expression:

`(11(0|1)(0|1)(0|1)(0|1)) | ((0|1)(0|1)11(0|1)(0|1)) | ((0|1)(0|1)(0|1)(0|1)11)`

You could have used macros to represent `(0|1)`, which would have simplified your answer significantly:

let `X = (0|1)`, then regular expression = `(11XXXX) | (XX11XX) | (XXXX11)`

Problem 1b

Regular Expression:

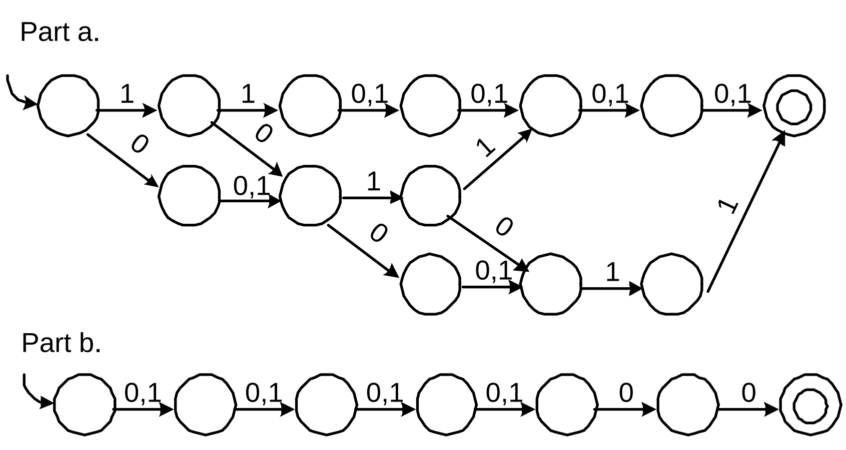
`(0|1)(0|1)(0|1)(0|1)00`

Similarly with macros:

regular expression = `XXXX00`

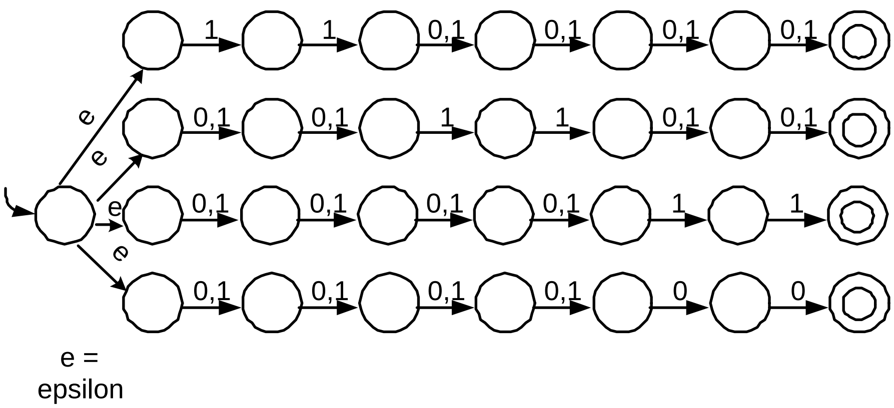
Problem 1c

DFAs:

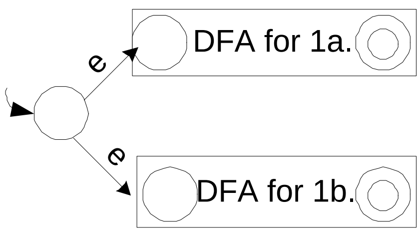


Problem 1d

NFA for the union:



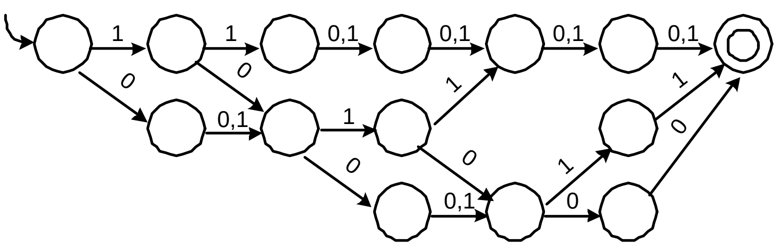
Many people noticed that you could simply OR the previous DFA's together to create the union:



The epsilon transitions from the start state point to the start states of the DFA's in 1a and 1b, and we preserve the same final states in both.

Problem 1e

DFA for the union:



Strings of length not equal to 6 are not accepted, along with some examples of length 6 which do not match the criteria above: `101010`, `010110`, etc. There were a lot of different answers for this question.

Problem 2a

X	FIRST(X)	FOLLOW(X)
S	x, y	x, y, $
A	x, y	x, y, $
B	x, ϵ	x, y, $
A B	x, y	
B A B	x, y	
y x	y	
x	x	
ϵ		

Problem 2b

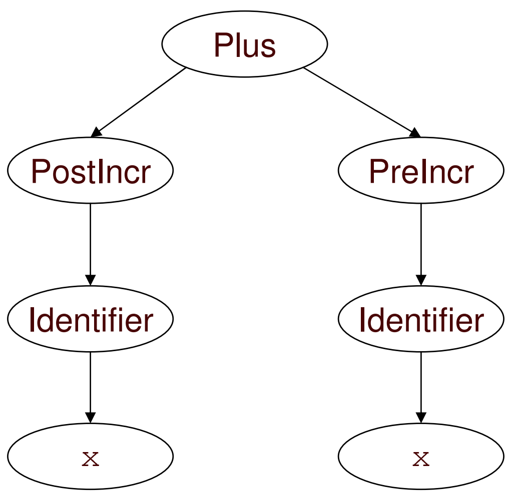
	x	y	$
A	A $\to$ S A	A $\to$ y x A $\to$ SA	(error)

Problem 2c

No, this grammar is not `LL(1)` because it has multiply defined entries in the parse table.

Problem 3a

There are many correct ASTs. Here is one example:



Problem 3b

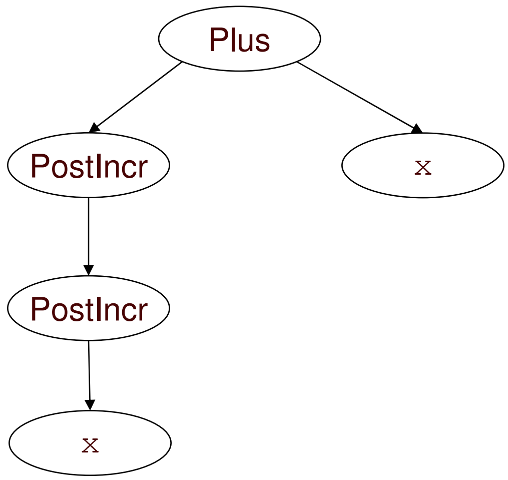
(i): The error was not found in the lexer. The lexer outputs the tokens `IDENTIFIER`, `++`, `++`, `+`, `IDENTIFIER`.

(ii): Two answers were possible. One is that the error is detected in the parser, because of a suitably restrictive grammar production for post-increment expressions, eg:

```
POSTINCREXPR -> IDENTIFIER ++
```

This would disallow the input `x++++`, since `x++` is not an identifier.

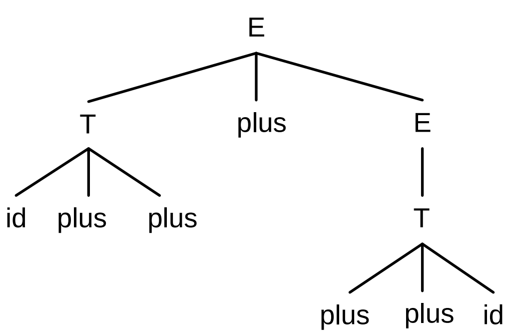
The other answer is that the parser does not detect the error, yielding the following AST:



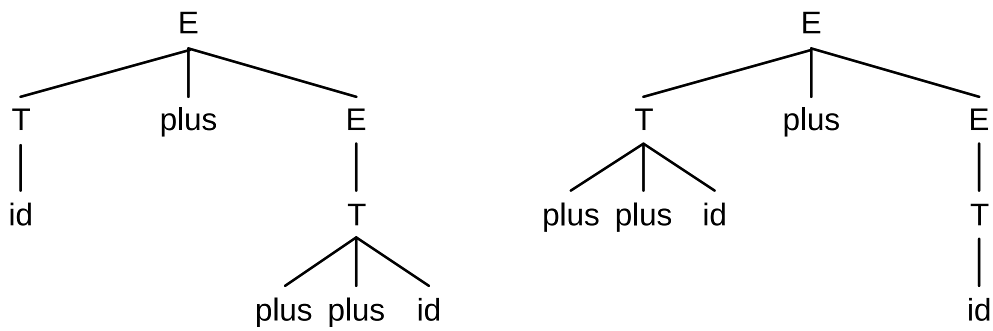
(iii): If you answered this question, the error must have been detected in the semantic checker. To see why, consider an evaluation of the AST with `x` having initial value `1`. First, we evaluate `x++`, which yields the value `1` (the increment happens at the very end of the evaluation). Then, we attempt to evaluate `1++`. But, this causes an error, since `++` must be applied to an argument whose value can be updated, like a variable.

Problem 3c

i Parse tree:



ii The grammar is ambiguous because there are two parse trees for `x+++x`:



Ambiguity is bad because if the program is parsed two different ways, it can mean two different things. The programmer can't be certain about the meaning of the program.

Problem 4

```
start = new NFAState();
end = new NFAState();
start.addTransition(NFAState.EPSILON, childStart);
start.addTransition(NFAState.EPSILON, end);
childEnd.addTransition(NFAState.EPSILON, childStart);
childEnd.addTransition(NFAState.EPSILON, end);
```