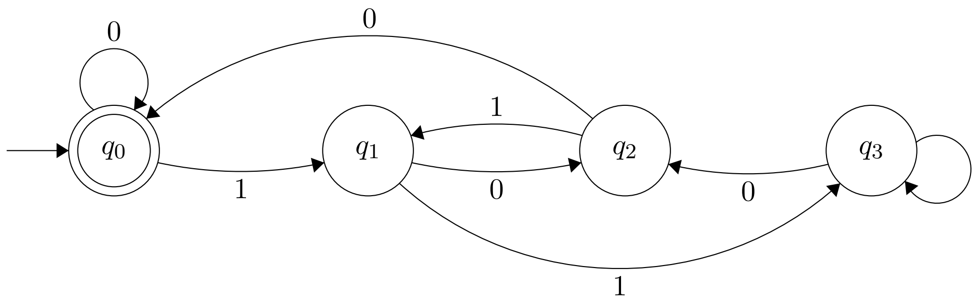
CS314 Fall 2018 Assignment 2 Solutions

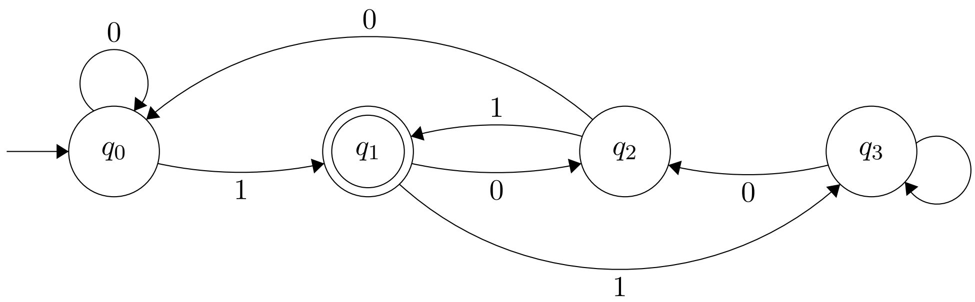
Problem 1

For both (a) and (b), q_i is the state representing ”i mod 4”. In both cases, q_0 is the start state.

(a)



(b)



Problem 2

(a)

<S> ::= <A><B><C>
<A> ::= a<A> | a
<B> ::= a<B>b | ε
<C> ::= c<C> | c

(b)

<S> ::= a<S>bbb | ε

(c)

<S> ::= a<S>a | b<S>b | ε

(d)
The language $\{a^n b^n c^n d^m \mid n \geq 0, m \geq 0\}$ is not context-free. Trying different rules, one can see why it would fail to generate this language. You may also use the pumping lemma.

(e)

$$
\begin{aligned}
\langle \mathrm{S} \rangle &::= \langle \mathrm{A} \rangle \langle \mathrm{A} \rangle \langle \mathrm{A} \rangle \\
\langle \mathrm{A} \rangle &::= a \mid b \mid \varepsilon
\end{aligned}
$$

Problem 3

(a)

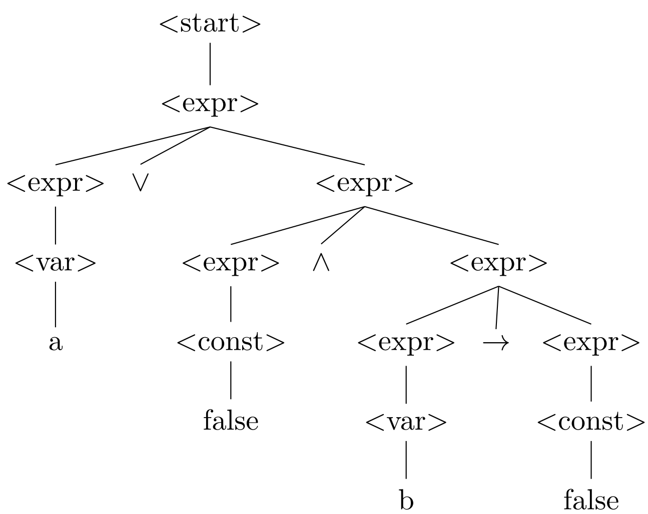
Left-most derivation:

1. <start>
2. $\to_L$ <expr>
3. $\to_L$ <expr> $\lor$ <expr>
4. $\to_L$ <var> $\lor$ <expr>
5. $\to_L$ a $\lor$ <expr>
6. $\to_L$ a $\lor$ <expr> $\land$ <expr>
7. $\to_L$ a $\lor$ <const> $\land$ <expr>
8. $\to_L$ a $\lor$ false $\land$ <expr>
9. $\to_L$ a $\lor$ false $\land$ <expr> $\to$ <expr>
10. $\to_L$ a $\lor$ false $\land$ <var> $\to$ <expr>
11. $\to_L$ a $\lor$ false $\land$ b $\to$ <expr>
12. $\to_L$ a $\lor$ false $\land$ b $\to$ <const>
13. $\to_L$ a $\lor$ false $\land$ b $\to$ false

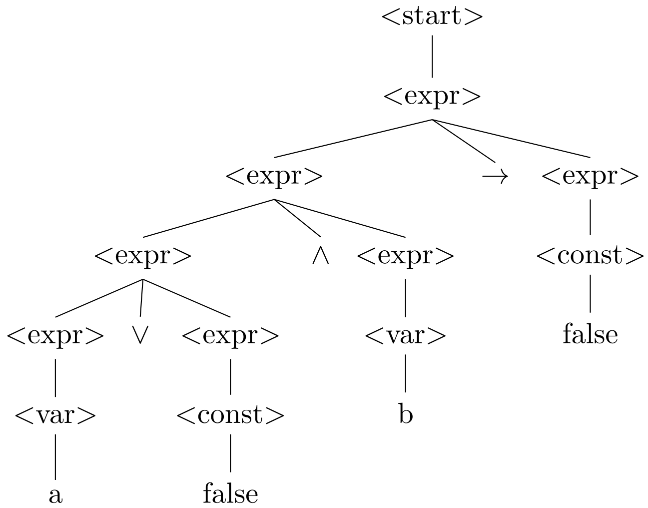
Right-most derivation:

1. <start>
2. $\to_R$ <expr>
3. $\to_R$ <expr> $\to$ <expr>
4. $\to_R$ <expr> $\to$ <const>
5. $\to_R$ <expr> $\to$ false
6. $\to_R$ <expr> $\land$ <expr> $\to$ false
7. $\to_R$ <expr> $\land$ <var> $\to$ false
8. $\to_R$ <expr> $\land$ b $\to$ false
9. $\to_R$ <expr> $\lor$ <expr> $\land$ b $\to$ false
10. $\to_R$ <expr> $\lor$ <const> $\land$ b $\to$ false
11. $\to_R$ <expr> $\lor$ false $\land$ b $\to$ false
12. $\to_R$ <var> $\lor$ false $\land$ b $\to$ false
13. $\to_R$ a $\lor$ false $\land$ b $\to$ false

(b)
Left-most parse tree:

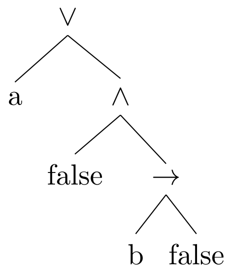


Right-most parse tree:

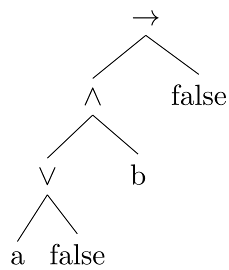


(c)

AST for left-most derivation:



AST for right-most derivation:



(d)

For example, suppose we have the string: a ∨ b ∧ c. Then we have the following left-most derivations.

Left-most derivation #1:

1. <start>
2. $\rightarrow_L$ <expr>
3. $\rightarrow_L$ <expr> ∨ <expr>
4. $\rightarrow_L$ <var> ∨ <expr>
5. $\rightarrow_L$ a ∨ <expr>
6. $\rightarrow_L$ a ∨ <expr> ∧ <expr>
7. $\rightarrow_L$ a ∨ <var> ∧ <expr>
8. $\rightarrow_L$ a ∨ b ∧ <expr>
9. $\rightarrow_L$ a ∨ b ∧ <var>
10. $\rightarrow_L$ a ∨ b ∧ c

Left-most derivation: #2

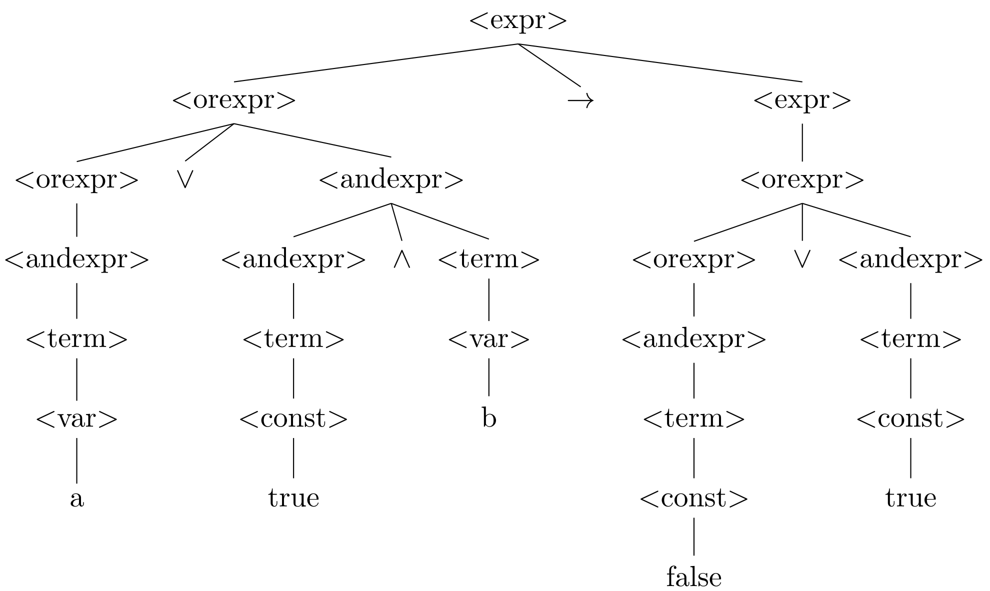
1. <start>
2. $\rightarrow_L$ <expr>
3. $\rightarrow_L$ <expr> ∧ <expr>
4. $\rightarrow_L$ <expr> ∨ <expr> ∧ <expr>
5. $\rightarrow_L$ <var> ∨ <expr> ∧ <expr>
6. $\rightarrow_L$ a ∨ <expr> ∧ <expr>
7. $\rightarrow_L$ a ∨ <var> ∧ <expr>
8. $\rightarrow_L$ a ∨ b ∧ <expr>
9. $\rightarrow_L$ a ∨ b ∧ <var>
10. $\rightarrow_L$ a ∨ b ∧ c

Since this string has two possible left-most derivations the grammar is ambiguous.

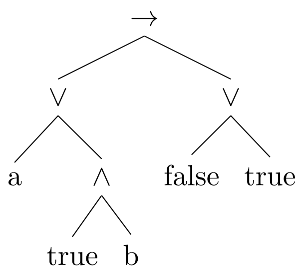
(e)
The new grammar looks like the following, by adding new non-terminals <orexpr>, <andexpr>, and <term>:

$$
\begin{aligned}
\langle\text{start}\rangle &::= \langle\text{expr}\rangle \\
\langle\text{expr}\rangle &::= \langle\text{orexpr}\rangle \rightarrow \langle\text{expr}\rangle \mid \langle\text{orexpr}\rangle \\
\langle\text{orexpr}\rangle &::= \langle\text{orexpr}\rangle \vee \langle\text{andexpr}\rangle \mid \langle\text{andexpr}\rangle \\
\langle\text{andexpr}\rangle &::= \langle\text{andexpr}\rangle \wedge \langle\text{term}\rangle \mid \langle\text{term}\rangle \\
\langle\text{term}\rangle &::= \langle\text{const}\rangle \mid \langle\text{var}\rangle \\
\langle\text{const}\rangle &::= \textbf{true} \mid \textbf{false} \\
\langle\text{var}\rangle &::= a|b|c|...|z
\end{aligned}
$$

(f)
Parse tree for “a ∨ true ∧ b → false ∨ true”:



AST for “a ∨ true ∧ b → false ∨ true”:

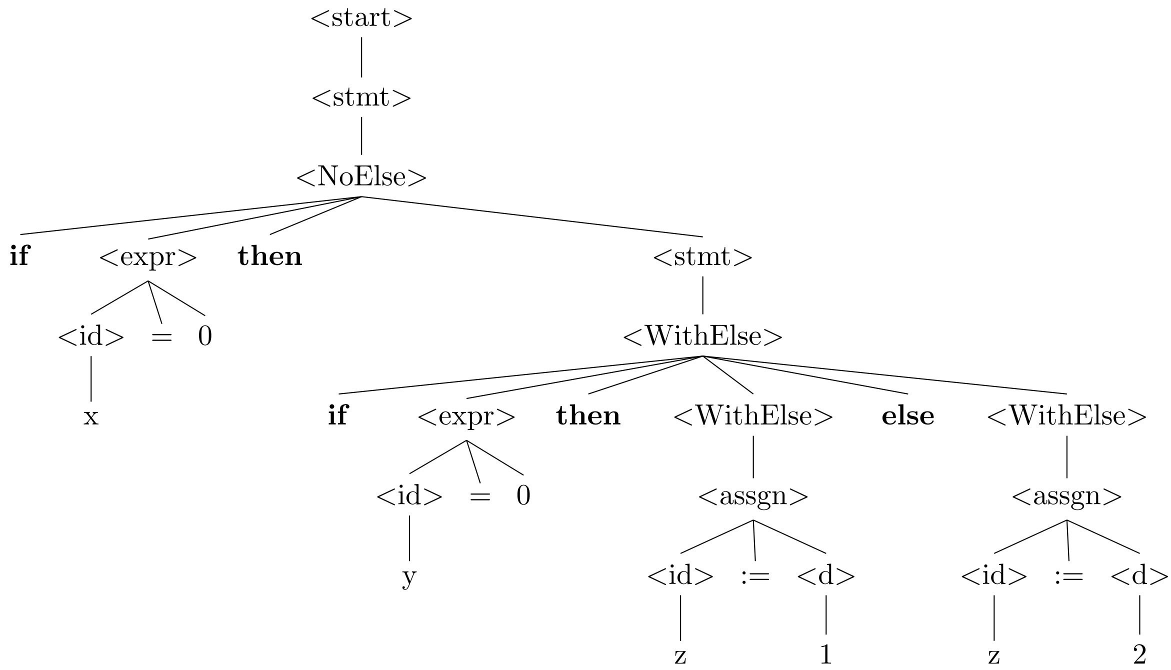


Extra Credit - Problem 4

Yes, we can. For example, we can change the grammar to be:

<start> ::= <stmt>
<stmt> ::= <WithElse>|<NoElse>
<WithElse> ::= **if**<expr>**then**<WithElse>**else**<WithElse> | <assgn>
<NoElse> ::= **if**<expr>**then**<stmt> | **if**<expr>**then**<WithElse>**else**<NoElse>
<assgn> ::= <id> := <d>
<expr> ::= <id> = 0
<d> ::= 0|1|2|3|4|5|6|7|8|9
<id> ::= $a|b|c|\ldots|z$

by adding the non-terminals <WithElse> and <NoElse>, replacing <if-stmt>. The statement **if** x = 0 **then if** y = 0 **then** z := 1 **else** z := 2 will have the following parse tree with this new grammar:



Notice that at depth = 3 (with the root of the tree having depth = 1), we cannot replace <NoElse> by <WithElse>, since the rule

<WithElse> ::= **if** <expr> **then** <WithElse> **else** <WithElse>

requires that the middle non-terminal <WithElse> must be replaced with a string that contains **if**, **then**, and **else**.

Also, notice that no new terminals were added.