ECE 468 & 573
Problem Set 2: Context-free Grammars, Parsers (Solutions)

1. For the following sub-problems, consider the following context-free grammar:

$$
\begin{aligned}
S &\rightarrow AA\$ && (1)\\
A &\rightarrow xAy && (2)\\
A &\rightarrow BB && (3)\\
B &\rightarrow zB && (4)\\
B &\rightarrow q && (5)\\
& && (6)
\end{aligned}
$$

   (a) What are the terminals and non-terminals of this language?

   Solution: The non-terminals are any of the symbols that appear on the left hand side of productions: $\{S, A, B\}$. The terminals are any of the symbols that appear in completely-rewritten strings (i.e., strings with no non-terminals in them). These are all of the symbols that *don't* appear on the right hand side, except for λ, which is not a symbol that appears in strings: $\{x, y, z, q, \$\}$

   (b) Show the derivation of the string $xqqyzqq\$$ starting from S (specify which production you used at each step), and give the parse tree according to that derivation.

   Solution: Here is the derivation. I will use a left derivation for this (we will rewrite starting from the left-most non-terminal)

$$
\begin{aligned}
S &\Rightarrow AA\$ && \text{Using } S \rightarrow AA\$\\
&\Rightarrow xAyA\$ && \text{Using } A \rightarrow xAy\\
&\Rightarrow xBByA\$ && \text{Using } A \rightarrow BB\\
&\Rightarrow xqByA\$ && \text{Using } B \rightarrow q\\
&\Rightarrow xqqyA\$ && \text{Using } B \rightarrow q\\
&\Rightarrow xqqyBB\$ && \text{Using } A \rightarrow BB\\
&\Rightarrow xqqyzBB\$ && \text{Using } B \rightarrow zB\\
&\Rightarrow xqqyzqB\$ && \text{Using } B \rightarrow q\\
&\Rightarrow xqqyzqq\$ && \text{Using } B \rightarrow q
\end{aligned}
$$

   (c) Give the first and follow sets for each of the non-terminals of the grammar.

   Solution: We begin by computing the first sets for each non-terminal. To build up the first sets, we start by examining each production to determine

which constraints that production creates on the various first sets. In the first pass over each production, we get:

$$\begin{array}{lcll} First(S) & \supseteq & First(A) - \lambda & \text{from production 1} \\ First(A) & \supseteq & First(x) = \{x\} & \text{from production 2} \\ First(A) & \supseteq & First(B) - \lambda & \text{from production 3} \\ First(B) & \supseteq & First(z) = \{z\} & \text{from production 4} \\ First(B) & \supseteq & First(q) = \{q\} & \text{from production 5} \end{array}$$

Solving these constraints to find the minimum assignment to each first set (using the method we discussed in class), we get an initial "guess" for the first sets of:

$$\begin{array}{rcl} First(S) & = & \{x, z, q\} \\ First(A) & = & \{x, z, q\} \\ First(B) & = & \{z, q\} \end{array}$$

Because none of these first sets include λ, when we go over the productions again, we don't have to add any new constraints. (If the first set for a non-terminal contained λ, then for any production whose right-hand-side started with that non-terminal, we would have to look at the first set of the second symbol on the RHS).

We can now compute the follow sets. Follow sets begin by looking at the non-terminals that appear on the *right hand side* of rules.

$$\begin{array}{lcll} Follow(A) & \supseteq & First(A\$) = \{x, z, q\} & \text{from the first } A \text{ in production 1} \\ Follow(A) & \supseteq & First(\$) = \{\$\} & \text{from the second } A \text{ in production 1} \\ Follow(A) & \supseteq & First(y) & \text{from production 2} \\ Follow(B) & \supseteq & First(B) = \{z, q\} & \text{from the first } B \text{ in production 3} \\ Follow(B) & \supseteq & Follow(A) & \text{from the second } B \text{ in production 3} \\ Follow(B) & \supseteq & Follow(B) & \text{from production 4} \end{array}$$

When we solve these set constraints, we get:

$$\begin{array}{rcl} Follow(S) & = & \{\} \\ Follow(A) & = & \{x, y, z, q, \$\} \\ Follow(B) & = & \{x, y, z, q, \$\} \end{array}$$

(d) What are the predict sets for each production?

Solution: The predict set for a production of the form $X \to \alpha$ is:

$$First(\alpha)$$

if $\lambda \notin First(\alpha)$ and:

$$First(\alpha) - \lambda \cup Follow(X)$$

if $\lambda \in First(\alpha)$. From this, we can compute the predict sets for each production:

$$\begin{aligned} Predict(S \rightarrow AA\$) &= \{x, z, q\} \\ Predict(A \rightarrow xAy) &= \{x\} \\ Predict(A \rightarrow BB) &= \{z, q\} \\ Predict(B \rightarrow zB) &= \{z\} \\ Predict(B \rightarrow q) &= \{q\} \end{aligned}$$

(e) Give the parse table for the grammar. Is this an LL(1) grammar? Why or why not?

Solution: The parse table is just a different representation of the predict sets. Each non-terminal gets a row, and each terminal gets a column. Each entry is either a production or an error. Entry $T[V_n][V_t] = p$ if production $p = V_n \rightarrow \alpha$ and $V_t \in Predict(p)$. In other words, if the non-terminal of a production p is V_n, then for every terminal V_t in the predict set of p, $T[V_n][V_t] = p$. In this case, we get the following table:

	x	y	z	q	\$
S	1		1	1	
A	2		3	3	
B			4	5	

Because there are no conflicts in the predict table (i.e., because each production for a given non-terminal has a disjoint predict set), this grammar is LL(1).

(f) Show the steps your parser would take to parse "xqqyzqq\$".

Solution:

Parse stack	**Remaining input**	**Action**
S	xqqyzqq$	Predict 1
AA$	xqqyzqq$	Predict 2
xAyA$	xqqyzqq$	match(x)
AyA$	qqyzqq$	Predict 3
BByA$	qqyzqq$	Predict 5
qByA$	qqyzqq$	match(q)
ByA$	qyzqq$	Predict 5
qyA$	qyzqq$	match(q)
yA$	yzqq$	match(y)
A$	zqq$	Predict 3
BB$	zqq$	Predict 4
zBB$	zqq$	match(z)
BB$	qq$	Predict 5
qB$	qq$	match(q)
B$	q$	Predict 5
q$	q$	match(q)
$	$	match($) (accept)

(g) Suppose we change the last rule to $B \rightarrow \lambda$. Is the resulting grammar LL(1)? Why or why not?

Solution: Rebuilding the first and follow sets for the non-terminals, we get:

$$\begin{aligned} First(S) &= \{x, z, \$\} \\ First(A) &= \{x, z, \lambda\} \\ First(B) &= \{z, \lambda\} \end{aligned}$$

$$\begin{aligned} Follow(S) &= \{\} \\ Follow(A) &= \{x, y, z, \$\} \\ Follow(B) &= \{x, y, z, \$\} \end{aligned}$$

Putting them together, we get the following predict sets (note that now we sometimes have to worry about follow sets!):

$$\begin{aligned} Predict(S \rightarrow AA\$) &= \{x, z, \$\} \\ Predict(A \rightarrow xAy) &= \{x\} \\ Predict(A \rightarrow BB) &= \{x, y, z, \$\} \\ Predict(B \rightarrow zB) &= \{z\} \\ Predict(B \rightarrow \lambda) &= \{x, y, z, \$\} \end{aligned}$$

Now we have some clear prediction conflicts. If we're trying to predict what to rewrite an A into, and we see an x, we don't know whether to rewrite it into xAy or into BB. Similarly, if we're trying to predict what to rewrite a B into, and we see a z, we don't know whether to rewrite it into zB or λ.

(h) Explain why the grammar isn't LL(k) for *any* k.

Solution: This grammar is ambiguous. The following string has numerous possible parse trees: z. As a result of the ambiguity, no matter how much lookahead we use, the parser will never be able to decide which production to predict.

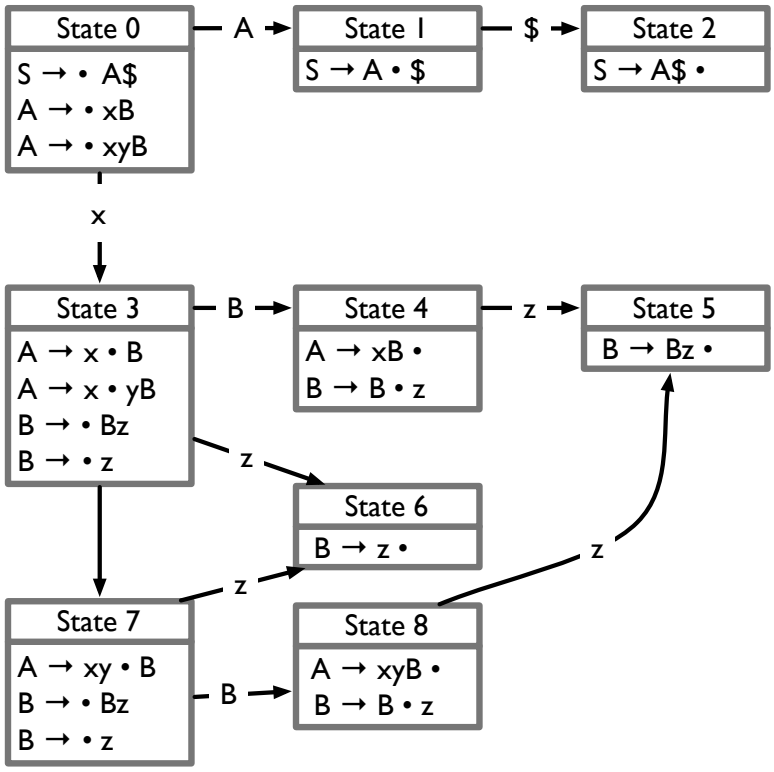
2. for the following sub-problems, consider the following grammar:

$$S \rightarrow A\$ \tag{7}$$
$$A \rightarrow xB \tag{8}$$
$$A \rightarrow xyB \tag{9}$$
$$B \rightarrow Bz \tag{10}$$
$$B \rightarrow z \tag{11}$$

Note that this question had a bug in it. The given grammar was not LR(0), so there is no way to parse it with an LR(0) parser. I will show the steps up through question 2b, and then show you the full process for a fixed grammar.

(a) Build the CFSM for this grammar.

Solution: Here is the CFSM:



(b) Build the goto and action tables for this grammar. Is it an LR(0) grammar? Why or why not?

Solution: There are conflicts in the action table: in state 4, we don't know whether to reduce or shift, and the same in state 8. As a result, this is not an LR(0) grammar.

(c) IGNORE Show the steps taken by the parser when parsing the string: $xyzz\$$. Give the action and show the state stack and remaining input for each step of the parse. Shift actions should be of the form "Shift X" where X is the state you are shifting to, and Reduce actions should be of the form "Reduce R, goto X" where "R" is the rule being used to reduce, and "X" is the state the parser winds up in.

(d) IGNORE Suppose we add another production:

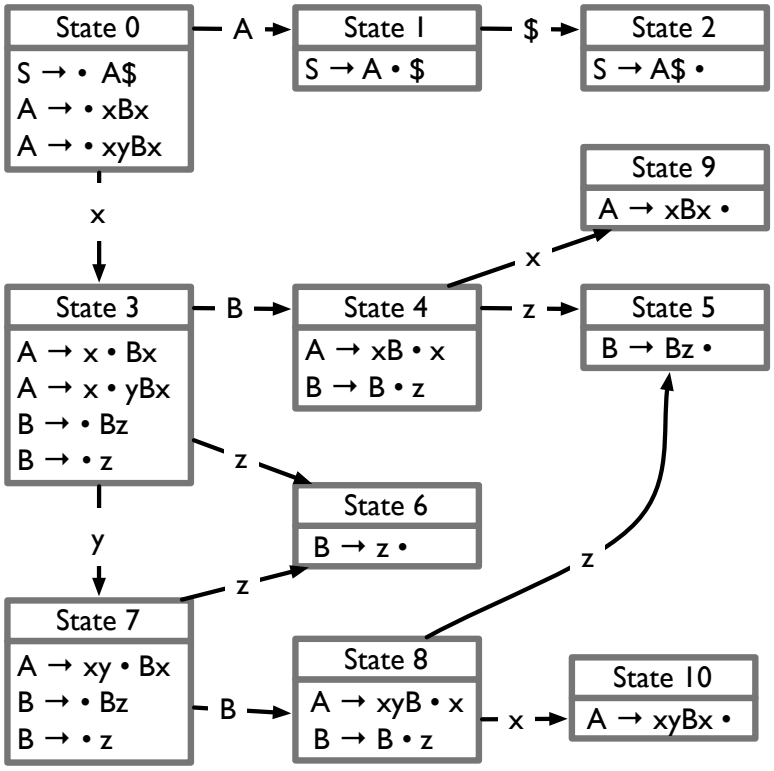
$$B \rightarrow yz$$

Is the resulting grammar LR(0)? Why or why not?

Extra problem
Let's build a CFSM for a related grammar:

$$
\begin{aligned}
S &\rightarrow A\$ \\
A &\rightarrow xBx \\
A &\rightarrow xyBx \\
B &\rightarrow Bz \\
B &\rightarrow z
\end{aligned}
$$

Here is the associated CFSM:



Note that in this CFSM, there are no states where there are both Shift configurations and Reduce configurations. Every state either has all Shift configurations, or exactly one Reduce configuration. Hence, this machine has no Shift/Reduce or Reduce/Reduce conflicts. The action table is just the tabular representation of this machine. The Action table is as follows:

State	Action
0	Shift
1	Shift
2	Reduce 1 (Accept)
3	Shift
4	Shift
5	Reduce 4
6	Reduce 5
7	Shift
8	Shift
9	Reduce 2
10	Reduce 3

Note that State 2 looks like a Reduce state. But because the production being reduced is the goal production (the production for S), we can Accept if we ever get to this state.

Let's see how the following string is parsed by this machine.

$$xyzzx\$$$

Recall that we parse by keeping a stack of states (starting in state 0). The state at the "top" of the stack is the state we are currently in. When we shift, we consume the next token off the input, and use the goto table to decide which table to go to; that state is pushed onto the stack. When we reduce, we "back up" as many steps as there are symbols on the right hand side of the rule we are reducing—we pop that many symbols off the stack. Then, from the state we end up in, we look at the goto table to decide which state to go to based on the symbol on the *left hand side* of the rule: we Reduce (according to a rule) and goto (the next state).

State stack	Remaining input	Action	Explanation
0	xyzzx$	Shift 3	Consume x from the input and shift
0 3	yzzx$	Shift 7	Consume y from the input and shift
0 3 7	zzx$	Shift 6	Consume z from the input and shift
0 3 7 6	zx$	Reduce 5, goto 8	Back up one (z) and replace with B
0 3 7 8	zx$	Shift 5	Consume z from the input and shift
0 3 7 8 5	x$	Reduce 4, goto 8	Back up two (Bz) and replace with B
0 3 7 8	x$	Shift 10	Consume x from the input and shift
0 3 7 8 10	$	Reduce 3, goto 1	Back up four (xyBx) and replace with A
0 1	$	Shift 2	Consume $ from the input and shift
0 1 2		Accept	We have matched the string.

Note what happens if we try to parse a string that *does not* match the grammar:

$$xyzzy$$

State stack	Remaining input	Action	Explanation
0	xyzzy$	Shift 3	Consume x from the input and shift
0 3	yzzy$	Shift 7	Consume y from the input and shift
0 3 7	zzy$	Shift 6	Consume z from the input and shift
0 3 7 6	zy$	Reduce 5, goto 8	Back up one (z) and replace with B
0 3 7 8	zy$	Shift 5	Consume z from the input and shift
0 3 7 8 5	y$	Reduce 4, goto 8	Back up two (Bz) and replace with B
0 3 7 8	y$	Error	No valid transition for y, so we have an error