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

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

$$
\begin{aligned}
S &\rightarrow AB\$ && (1)\\
A &\rightarrow xAx && (2)\\
A &\rightarrow B && (3)\\
A &\rightarrow \lambda && (4)\\
B &\rightarrow yBy && (5)\\
B &\rightarrow A && (6)\\
B &\rightarrow \lambda && (7)\\
& && (8)
\end{aligned}
$$

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

   Answer: Terminals: {x, y, $}. Non-terminals: {S, A, B}

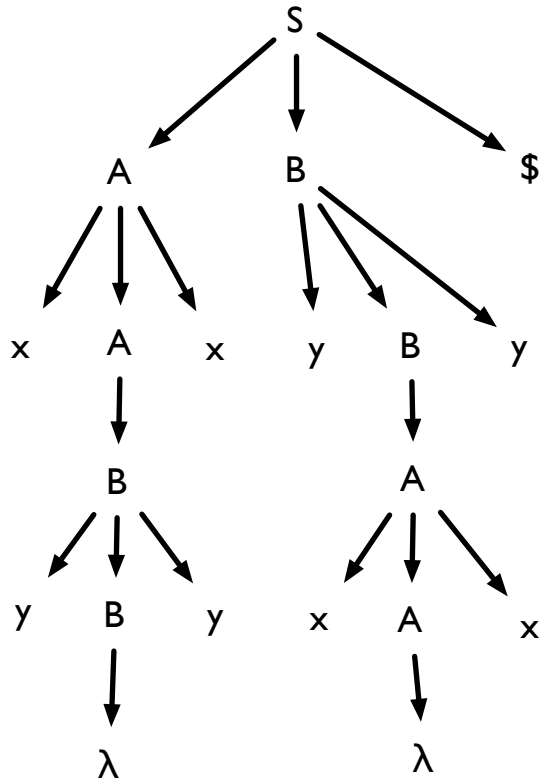
   (b) Is this a regular language (*i.e.*, could you write a regular expression that generates this language)?

   Answer: No, this is not a regular language. To see this, consider starting with S, and rewriting the first B into λ, leaving us with just A. Then note that each x added to the beginning of the string must be matched by the same number of xs at the end of the string. We can then turn the A into a B to generate matched ys, etc. What we are generating is a palindrome made up of xs and ys. To create a palindrome, we would have to "remember" what the first half of the palindrome looked like to make sure the second half corresponds. That's not possible to do with a finite amount of state, so we can't build a finite automaton to capture these strings. Hence, no regular expression can match these strings, so the language is not regular.

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

   Answer:

$$
\begin{array}{rcll}
S & \rightarrow & AB\$ & \text{Rule 1} \\
 & \rightarrow & xAxB\$ & \text{Rule 2} \\
 & \rightarrow & xBxB\$ & \text{Rule 3} \\
 & \rightarrow & xyByxB\$ & \text{Rule 5} \\
 & \rightarrow & xyyxB\$ & \text{Rule 7} \\
 & \rightarrow & xyyxyBy\$ & \text{Rule 5} \\
 & \rightarrow & xyyxyAy\$ & \text{Rule 6} \\
 & \rightarrow & xyyxyxAxy\$ & \text{Rule 2} \\
 & \rightarrow & xyyxyxxy\$ & \text{Rule 4}
\end{array}
$$



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

Answer:
We go through each of the productions in turn, generating new set constraints until we can't generate any more. The set constraints we generate are:

$First(S) \supseteq First(A) - \lambda$ From production 1

$$
\begin{aligned}
First(A) &\supseteq \{x\} \text{ From production 2} \\
First(A) &\supseteq First(B) - \lambda \text{ From production 3} \\
First(A) &\supseteq \{\lambda\} \text{ From production 4} \\
First(B) &\supseteq \{y\} \text{ From production 5} \\
First(B) &\supseteq First(A) - \lambda \text{ From production 6} \\
First(B) &\supseteq \{\lambda\} \text{ From production 7} \\
First(S) &\supseteq First(B) - \lambda \text{ From production 1 because } \lambda \in First(A) \\
First(S) &\supseteq \{\$\} \text{ From production 1 because } \lambda \in First(B)
\end{aligned}
$$

Solving these set constraints gives us:

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

We then turn to the follow sets. Here, we go through the rules and add constraints based on non-terminals that appear on the *right* hand side:

$$
\begin{aligned}
Follow(A) &\supseteq First(B\$) = \{x, y, \$\} \text{ From production 1} \\
Follow(B) &\supseteq First(\$) = \{\$\} \text{ From production 1} \\
Follow(A) &\supseteq First(x) = \{x\} \text{ From production 2} \\
Follow(B) &\supseteq Follow(A) \text{ From production 3} \\
Follow(B) &\supseteq First(y) = \{y\} \text{ From production 5} \\
Follow(A) &\supseteq Follow(B) \text{ From production 6}
\end{aligned}
$$

Solving these set constraints gives us:

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

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

Answer:

$$
\begin{aligned}
Predict(1) &= First(AB\$) = \{x, y, \$\} \\
Predict(2) &= First(xAx) = \{x\} \\
Predict(3) &= (First(B) - \lambda) \cup Follow(A) = \{x, y, \$\} \\
Predict(4) &= Follow(A) = \{x, y, \$\} \\
Predict(5) &= First(yBy) = \{y\} \\
Predict(6) &= (First(A) = \lambda) \cup Follow(B) = \{x, y, \$\} \\
Predict(7) &= Follow(B) = \{x, y, \$\}
\end{aligned}
$$

Note that the predict sets for productions 3 and 6 include the follow sets of their left hand sides. That's because the *first* sets of the right hand sides include λ. If the right hand side can be rewritten to nothing, then we also need to consider what could come *after* the non-terminal we are predicting.

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

Answer

	x	y	$
S	1	1	1
A	2, 3, 4	3, 4	3, 4
B	6, 7	5, 6, 7	6, 7

This is not LL(1) because there are conflicts in the parse table.

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

$$
\begin{aligned}
S &\rightarrow AB\$ && (9) \\
A &\rightarrow Ax && (10) \\
A &\rightarrow z && (11) \\
A &\rightarrow x && (12) \\
B &\rightarrow yB && (13) \\
B &\rightarrow w && (14)
\end{aligned}
$$

(a) Explain why this grammar is not LL(k) for any k.

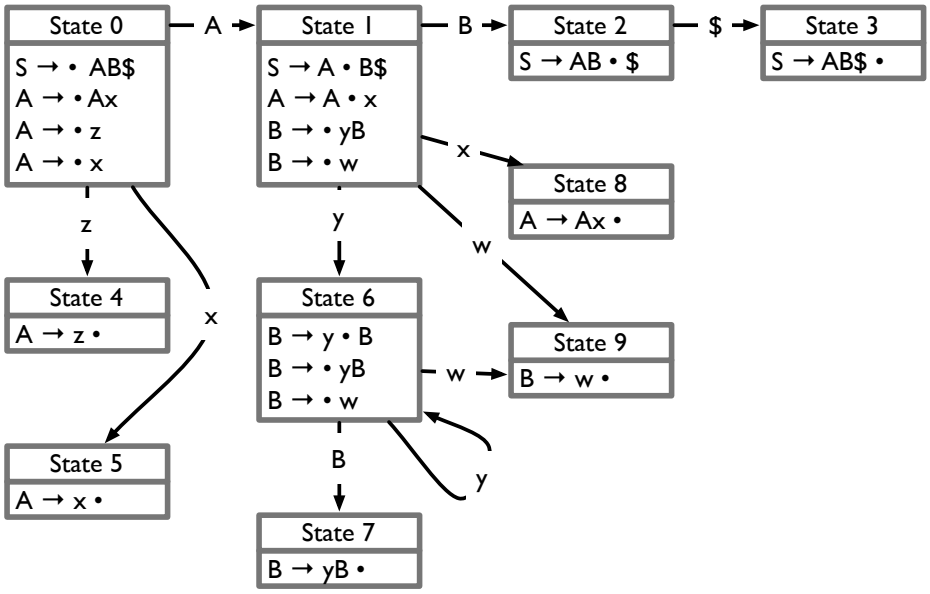
Answer: The easy answer is that the grammar is left-recursive (note the production $A \rightarrow Ax$).

Without knowing how many xs there are in the string, we don't know how many times to apply the production before turning the A into a z or an x. If you try

to build the predict sets for this grammar, you'll see that no matter how many lookahead tokens you add, you won't be able to resolve the conflict between productions 10 and 11 (when you see a z) and 10 and 12 (when you see an x).

(b) Build the CFSM for this grammar.

Answer: (Note that your state numbering may be different.)



A couple of things to note. In State 1, we had to consider what the B could become from configuration $S \to A \cdot B\$$, which caused us to add all the configurations for B. In state 6, the initial configuration was $B \to y \cdot B$, which means we have to worry about what the *second* B could become. Even though we're already in the middle of matching a B, we haven't seen any of the second B, so the "read head" is at the beginning of those productions.

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

Answer: The goto table can be read directly off the CFSM—it's just the transition table for the finite state machine. Note how each state in the CFSM has one outgoing transition for each symbol to the right of the "read head."

The following states are Shift states: 0, 1, 2, 6. The following states are Reduce states: 4, 5, 7, 8, 9. State 3 is the accept state.

This is an LR(0) grammar because there are no Shift/Reduce or Reduce/Reduce conflicts. Every state is either a shift state or a reduce state (the accept state is just a special reduce state).

(d) Show the steps taken by the parser when parsing the string: $zxyw\$$. 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.

Answer:

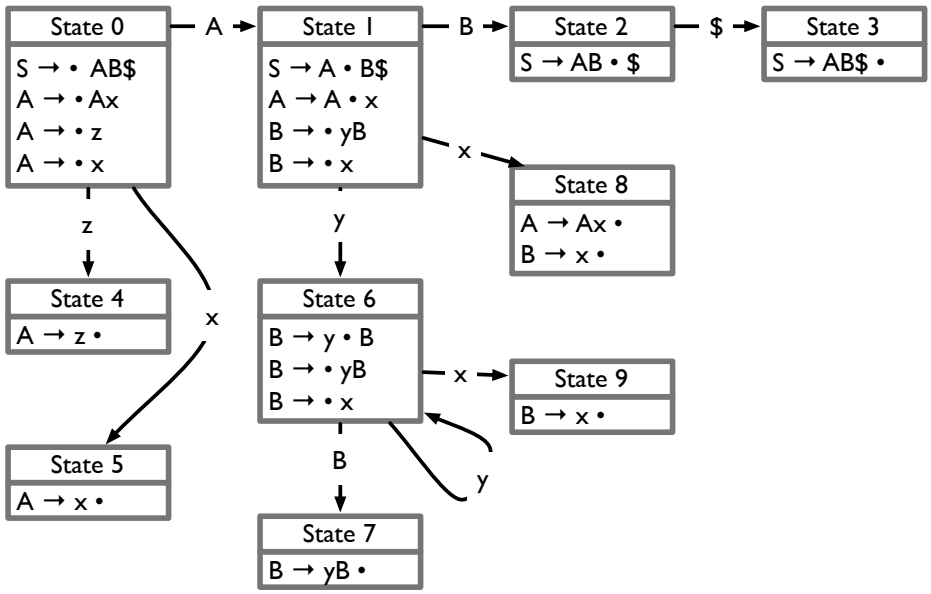
Parse stack	Symbol stack	Remaining input	Action
0		$zxyw\$$	Shift 4
0 4	z	$xyw\$$	Reduce 11, goto 1
0 1	A	$xyw\$$	Shift 8
0 1 8	Ax	$yw\$$	Reduce 10, goto 1
0 1	A	$yw\$$	Shift 6
0 1 6	Ay	$w\$$	Shift 9
0 1 6 9	Ayw	\$	Reduce 14, goto 7
0 1 6 7	AyB	\$	Reduce 13, goto 2
0 1 2	AB	\$	Shift 3
0 1 2 3	$AB\$$		Accept

(e) Suppose we change the last production to:

$$B \rightarrow x$$

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

Answer: No. To see why, consider the CFSM that gets built:



State 8 has a Reduce/Reduce conflict. If the parser gets to that state, it does not know whether to reduce using $A \to Ax$ or $B \to x$.