CS131 Compilers: Writing Assignment 2
Due Sunday, April 15, 2018 at 23:29

Rong Yuyang - 69850764

This assignment asks you to prepare written answers to questions on context-free grammars and parsing. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work. and you should indicate in your submission who you worked with, if applicable. You should use the Latex template provided at the course web site to write your solution.

I worked with: (Name,ID), (Name,ID)...

1. $2 \times 3 = 6$ **pts** Give context-free grammar (CFG) for each of the following languages:

   (a) The set of all finite strings over the alphabet $\{0, 1\}$ with an equal number of 0's and 1's.

   $$A \to 0A1A \mid 1A0A \mid \epsilon$$

   (b) The set of all finite strings over the alphabet $\{0, 1\}$ with the number of 0's greater than the number of 1's.

   $$S \to EqualZerosEqual \mid EqualZerosEqualS$$
   $$Equal \to 0Equal1Equal \mid 1Equal0Equal \mid \epsilon$$
   $$Zeros \to 0 \mid 0Zeros$$

   (c) The set $L_3 = L_1 \cap L_2$, where L_1 and L_2 are defined below. Let L_1 be the finite strings consisting of all non-empty *palindromes* over the alphabet $\{a, b\}$. That is L_1 consists of all sequences of a's and b's that read the same forward or backward. For example, $abba$, $aabbbaa \in L_1$, but $abb \notin L_1$. Let L_2 be the language over $\{a, b\}$ representing the language of the regular expression $b(a + b)^*$.

   $$S \to bPb$$
   $$P \to aPa \mid bPb \mid \epsilon$$

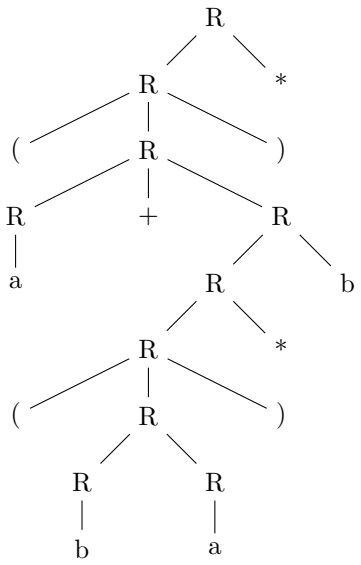
2. $3 \times 2 = 6$ **pts** Consider the following CFG with terminals $\{(,), +, *, a, b\}$ (+ represents union) that is used to represent regular expressions over alphabet $\{a, b\}$:

$$R \to R + R \mid RR \mid (R) \mid R^* \mid a \mid b$$

(a) Using the above CFG, provide a right-most derivation for the following input string $(a + (ba)^* b)^*$.

$$\begin{aligned} R &\Rightarrow R^* \\ &\Rightarrow (R)^* \\ &\Rightarrow (R + R)^* \\ &\Rightarrow (R + RR)^* \Rightarrow (R + Rb)^* \\ &\Rightarrow (R + R^* b)^* \\ &\Rightarrow (R + (R)^* b)^* \\ &\Rightarrow (R + (RR)^* b)^* \\ &\Rightarrow (R + (Ra)^* b)^* \Rightarrow (R + (ba)^* b)^* \Rightarrow (a + (ba)^* b)^* \end{aligned}$$

(b) For the derivation in above solution, provide the corresponding parse tree.



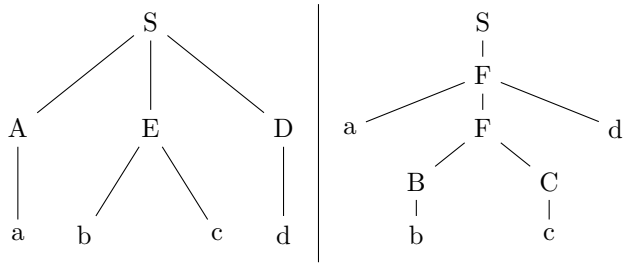
3. $3 \times 3 = 9$ **pts** Consider the following CFG.

$$\begin{array}{lcl} S & \rightarrow & AED \mid F \\ A & \rightarrow & Aa \mid a \\ B & \rightarrow & Bb \mid b \\ C & \rightarrow & Cc \mid c \\ D & \rightarrow & Dd \mid d \\ E & \rightarrow & bEc \mid bc \\ F & \rightarrow & aFd \mid BC \end{array}$$

(a) What is the language generated by this grammar?
It's LL(0). The requirement that $\#a == \#d$ or $\#b == \#c$ makes it non-RE.

(b) Is the grammar as given ambiguous? If yes, give an example of an expression with two parse trees under this grammar. If not, explain why that is the case.

adbc is a good example with two parsing trees:



(c) Transform the CFG given above by eliminating ambiguity and left recursion, if needed.

The conflict originates when $\#a == \#d$ and $\#b == \#c$ are both satisfied. Thus the rule is changed such that F cannot have $\#b == \#c$ satisfied. Thus only production rule $S \rightarrow AED$ can satisfy both condition at the same time.

$$\begin{array}{lcl} S & \rightarrow & AED \mid F \\ A & \rightarrow & a \mid aA \\ B & \rightarrow & b \mid bB \\ C & \rightarrow & c \mid cC \\ D & \rightarrow & d \mid dD \\ E & \rightarrow & bEc \mid bc \\ F & \rightarrow & aFd \mid bE \mid Ec \end{array}$$

4. $3 \times 3 = 9$ **pts** Consider the following CFG.

$$\begin{array}{lcl} A & \rightarrow & [AB] \mid a \\ B & \rightarrow & \epsilon \mid +AC \mid Cb \\ C & \rightarrow & \epsilon \mid -ABc \end{array}$$

(a) Compute the First and Follow sets for the grammar.

$FIRST([AB]) = \{[\}$
$FIRST(a) = a$
$FIRST(A) = FIRST([AB]) \cup FIRST(a) = \{[, a\}$
$FIRST(\epsilon) = \{\epsilon\}$
$FIRST(-ABc) = \{-\}$
$FIRST(C) = FIRST(\epsilon) \cup FIRST(-ABc) = \{\epsilon, -\}$
$FIRST(+AC) = \{+\}$
$FIRST(Cb) = \{-, b\}$
$FIRST(B) = FIRST(\epsilon) \cup FIRST(+AC) \cup FIRST(Cb) = \{\epsilon, +, -, b\}$

$FOLLOW(A) = \{+, -, b, c, \$\}$
$FOLLOW(B) = \{], c\}$
$FOLLOW(C) = \{], b, c\}$

(b) Give the LL(1) parsing table for the grammar.

NT	Input							
	[	]	a	+	-	b	c	$
A	$A \rightarrow [AB]$		$A \rightarrow a$					
B		$B \rightarrow \epsilon$		$B \rightarrow +AC$	$B \rightarrow Cb$	$B \rightarrow Cb$	$B \rightarrow \epsilon$	
C		$C \rightarrow \epsilon$			$C \rightarrow -ABc$	$C \rightarrow \epsilon$	$C \rightarrow \epsilon$	

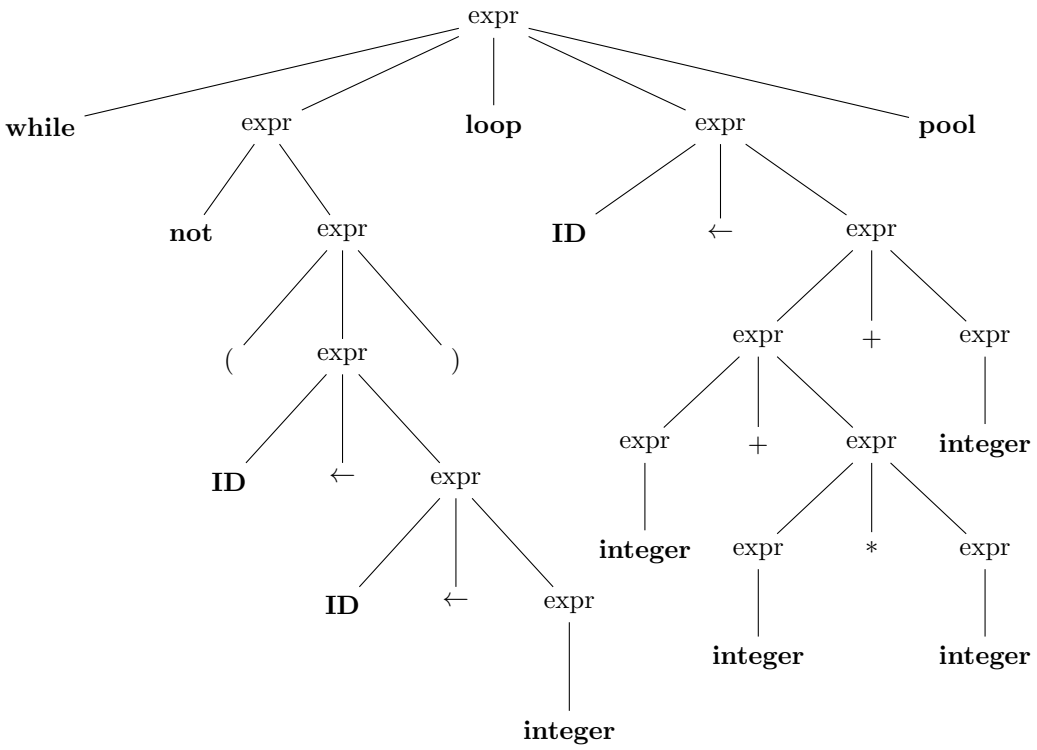
(c) Is this grammar LL(1)? and Why.

Yes, it is LL(1). There is no conflicting production rules in the table. There are no two rules that can derive to the same string, each production rule can only produce to ϵ in one way and other rules don't lead to a beginning terminal in FOLLOW.

5. 8 **pts** Using the context-free grammar for Cool given in Section 11 of the Cool manual, draw a parse tree for the following expression.

```
while not (x <-z <- 0) loop
  y <- z + 2 * x + 1
pool
```

Note that the context-free grammar by itself is ambiguous, so you will need to use the precedence and associativity rules in Section 11.1 to get the correct tree.



6. $4 \times 4 = 16$ **pts** Consider the following grammar describing a certain sort of nested lists:

$$\begin{array}{lcl} S & \rightarrow & T; S \mid \epsilon \\ T & \rightarrow & U \star T \mid U \\ U & \rightarrow & x \mid y \mid [S] \end{array}$$

S, T, and U are nonterminals, while others are terminals.

(a) Left-factor this grammar.

$$\begin{array}{lcl} S & \rightarrow & T; S \mid \epsilon \\ T & \rightarrow & UT' \\ T' & \rightarrow & \star T \mid \epsilon \\ U & \rightarrow & x \mid y \mid [S] \end{array}$$

(b) Give the First and Follow sets for each nonterminal in the grammar obtained in part (a).

$$FIRST(S) = \{x, y, [, \epsilon\}$$
$$FIRST(T) = \{x, y, [\}$$
$$FIRST(T') = \{\star, \epsilon\}$$
$$FIRST(U) = \{x, y, [\}$$

$$FOLLOW(S) = \{\$,]\}$$
$$FOLLOW(T) = \{;\}$$
$$FOLLOW(T') = \{;\}$$
$$FOLLOW(U) = \{;, \star\}$$

(c) Using this information, construct an LL(1) parsing table for the grammar obtained in part (a).

Table 1: My caption

NT	Input						
	[	]	x	y	;	$\star$	\$
S	$S \rightarrow T; S$	$S \rightarrow \epsilon$	$S \rightarrow T; S$	$S \rightarrow T; S$			$S \rightarrow \epsilon$
T	$T \rightarrow UT'$		$T \rightarrow UT'$	$T \rightarrow UT'$			
T'					$T' \rightarrow \epsilon$	$T' \rightarrow \star T$	
U	$U \rightarrow [S]$		$U \rightarrow x$	$U \rightarrow y$			

(d) Suppose we generated an LL(1) parser for the grammar using the table you constructed. What would go wrong if it tried to parse the following input string?

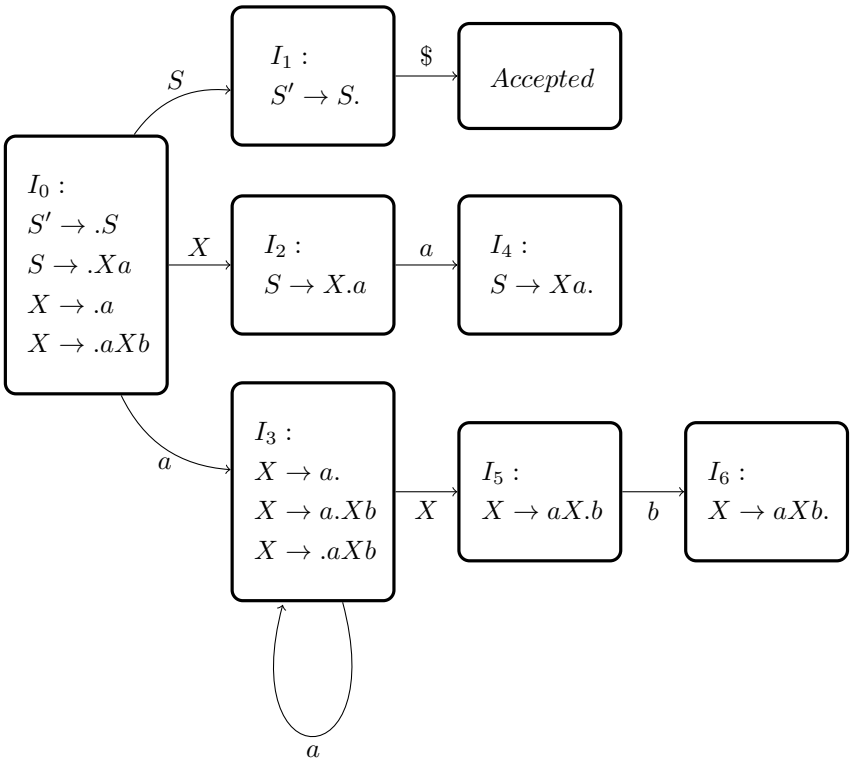
$$[x; y] \star [;$$

It will raise an error when encountered the last semi-colon, as it is in state S and a semi-colon(;) is an illegal input.

7. $3 \times 2 + 5 \times 2 = 16$ **pts** Consider the following CFG, which has the set of terminals $T = \{\mathbf{a}, \mathbf{b}\}$.

$$\begin{array}{lcl} S & \rightarrow & X\mathbf{a} \\ X & \rightarrow & \mathbf{a} \mid \mathbf{a}X\mathbf{b} \end{array}$$

(a) Construct a DFA for viable prefixes of this grammar using LR(0) items.



(b) Identify a shift-reduce conflict in this grammar under the SLR(1) rules.

$$1 : S \to Xa$$
$$2 : X \to a$$
$$3 : X \to aXb$$

State	Actions			Goto	
	a	b	\$	S	X
0	s3			s1	s2
1			acc		
2	s4				
3	**s3/r2**	r2			s5
4			r1		
5		s6			
6	r3	r3			

There is one in I_3 where the parser have to decide whether to reduce $X \to a.$ or keep shifting if there is X or a incoming.

(c) Assuming that an SLR(1) parser resolves shift-reduce conflicts by choosing to shift, show the operation of such a parser on the input string **aaba**.

Stack	**Input**	**Action**
\$	aaba\$	shift
\$a	aba\$	shift
\$aa	ba\$	reduce by $X \to a$
\$aX	ba\$	shift
\$aXb	a\$	reduce by $X \to aXb$
\$X	a\$	shift
\$Xa	\$	reduce by $S \to Xa$
\$S	\$	*Accepted*

(d) Suppose that the production $X \to \varepsilon$ is added to this grammar. Identify a reduce-reduce conflict in the resulting grammar under the SLR(1) rules.

In state I_3, when we have $X \to a.$, we can reduce using rule $X \to a$ or $X \to \epsilon$, and we cannot decide which rule to use.