CS131 Compilers: Writing Assignment 2
Due Saturday, April 10, 2021 at 23:59pm

Name - Number

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. Written assignments are turned in at the start of lecture. You should use the Latex template provided at the course web site to write your solution.

I worked with: ()()

1. $2 * 2 + 8 = 10$ **pts** Give context-free grammar (CFG) for each of the following languages, for the last part, you don't need to draw the tree but the semantic alongside with the production rule:

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

   S $\rightarrow A0A \mid A1A$
   A $\rightarrow AA$
   $A \rightarrow 0A1 \mid 1A0$
   A $\rightarrow \epsilon$

   (b) 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 $\{a, b\}$.
   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 $\rightarrow$ bBb
   B $\rightarrow$ aBa | bBb | ϵ

   (c) The set of all finite strings over the alphabet $\{0, 1\}$ that generate all the binary number including digits (e.g. 110.0110).

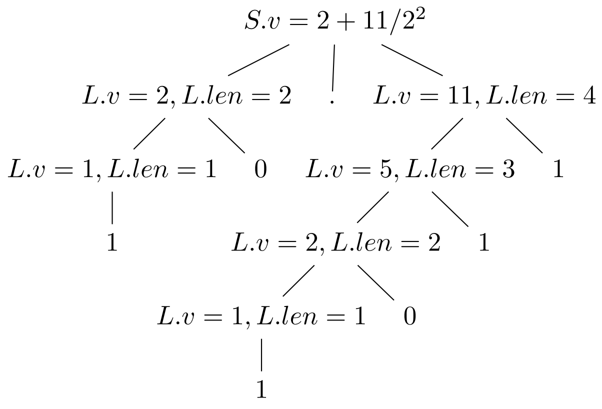
      i. Write the syntax-directed translation scheme (SDT) with S-attributed definition.

S $\rightarrow$ L	S.v = L.v
S $\rightarrow L_1.L_2$	S.v = L_1.v + L_2.v / $2^{L_1.len}$
L $\rightarrow L_1B$	L.v = $L_1.v$*2+B.v, L.len++
L $\rightarrow$ B	L.v = B.v, L.len=1
B $\rightarrow$ 0 | 1	B.v = I

      ii. Write the syntax-directed translation scheme (SDT) with L-attributed definition.

S → L	S.v = L.v
S → $L_1.L_2$	S.v = L_1.v + L_2.v / $2^{L_1.len}$
L → $L_1 B$	L.v = $L_1.v$*2+B.v, L.len++
L → B	L.v = B.v, L.len=1
B → 0 | 1	B.v = I

iii. Using the CFG you wrote, provide a derivation for the following input string: 10.1011



2. $3 \times 3 = 9$ **pts** Consider the following CFG.

$$\begin{array}{lcl} S & \rightarrow & G \\ G & \rightarrow & P \mid PG \\ P & \rightarrow & \textbf{id} : R \\ R & \rightarrow & \epsilon \mid \textbf{id}R \end{array}$$

(a) What is the language generated by this grammar?

The language generated by this grammar is [[**id** :][**id**]*]+

(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.

The grammar is unambiguous. Because Unambiguous Grammar has exactly one set of handles for a Right-most Derivation, and **id**:R and **id**R are exactly two separate rules, which contributes to one parser tree.

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

There's no need since it's unambiguous.

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

$$\begin{array}{lcl} S & \rightarrow & (X \\ S & \rightarrow & E] \\ S & \rightarrow & F) \\ X & \rightarrow & E) \\ X & \rightarrow & F] \\ E & \rightarrow & A \\ F & \rightarrow & A \\ A & \rightarrow & \epsilon \end{array}$$

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

Nullable(S) = True
Nullable(X) = True
Nullable(E) = True
Nullable(F) = True
Nullable(A) = True

First(S) = $\{(, \epsilon\}$
First(X) = $\{\epsilon,),]\}$
First(F) = $\{\epsilon\}$
First(E) = $\{\epsilon,),]\}$
First(A) = $\{\epsilon,),]\}$

Follow(S) = $\{\$\}$
Follow(E) = $\{],)\}$
Follow(F) = $\{),]\}$
Follow(X) = $\{\$\}$
Follow(A) = $\{),]\}$

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

Non-terminal			INPUT SYMBOL	
	(	)	]	$
S	S → X	S → E] S → F)	S → E] S → F)	S → E] S → F)
X		X → E)	X → F]	X → E)X → F]
E		E → A	E → A	
F		F → A	F → A	
A		A → ϵ	A → ϵ	

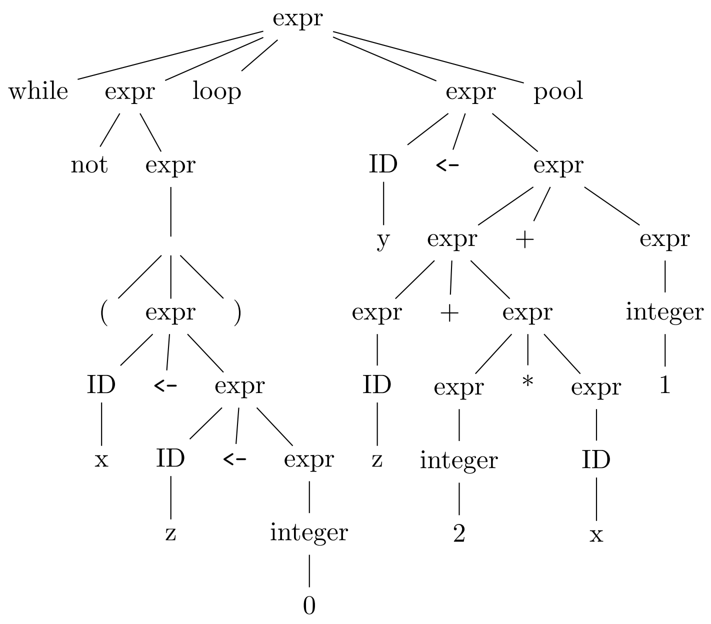
(c) Is this grammar LALR(1)? and Why.

No. There are conflicts in $E \rightarrow A$ and $F \rightarrow A$.

4. 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.



5. $4 \times 4 = 16$ **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.

S → T;S | ϵ
T → UT'
T' → $\star T$ | ϵ
$U \rightarrow x \mid y \mid [S]$

(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(U) = $\{x, y, [\}$
First(T') = $\{\star, \epsilon\}$
Follow(S) = $\{], \$\}$
Follow(T) = $\{;\}$
Follow(U) = $\{\star, ;\}$
Follow(T') = $\{;\}$

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

Non-terminal		Input symbol					
	;	$\star$	x	y	[	]	$
S			S → T;S	S → T;S	S → T;S	S → ϵ	S → ϵ
T			T → UT'	T → UT'	T → UT'		
T'	T' → ϵ	T' → $\star T$					
U			$U \rightarrow x$	$U \rightarrow y$	$U \rightarrow [S]$		

(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?

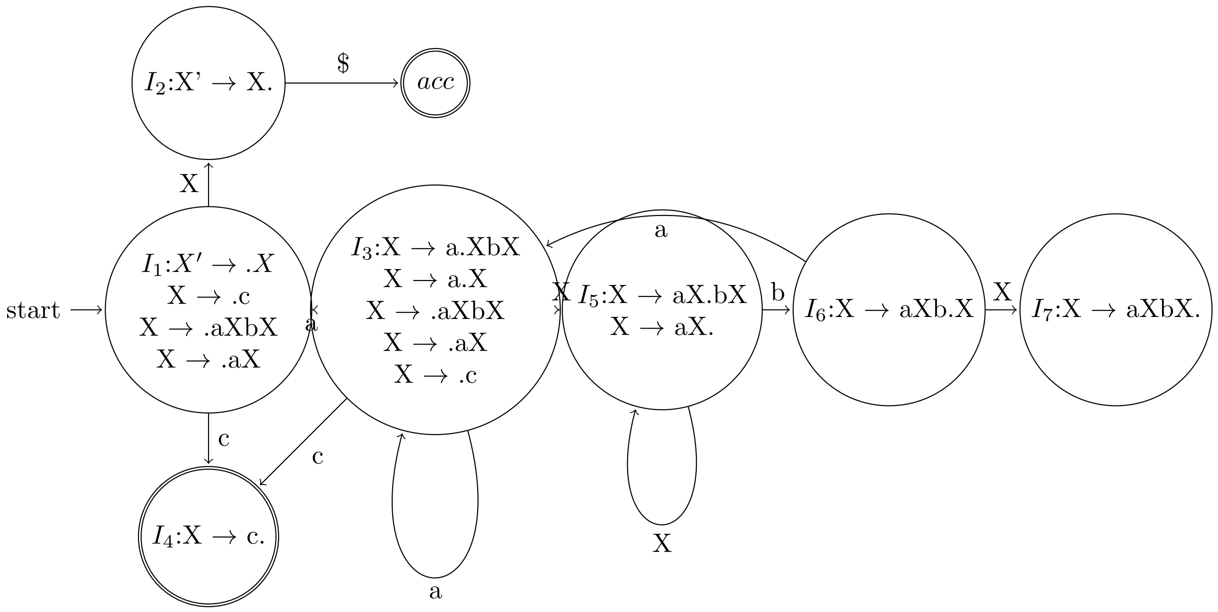
$$[x; y] \star [;$$

There's no rule accepting ; after [

6. $3 \times 2 + 5 \times 2 = 16$ **pts** Consider the following CFG, which has the set of terminals $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$.

$$X \quad \rightarrow \quad \mathbf{c} \mid \mathbf{a}X\mathbf{b}X \mid \mathbf{a}X$$

(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.

I_5

(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 **aacbc**.

```
$ | aacbc$ |
$a | acbc$ |
$aa | cbc$ |
$aac | bc$ |
$aaX | bc$ | X → c
$aaXb | c$ |
$aaXbc | $ |
$aaXbX | $ | X → c
$aX | $ | X → aXbX
$ | $ | X → aX
```

(d) Suppose you can use the **Bison** type of grammar to specify the priority to reduce the conflict, how can you do so? Write the **Bison** code.

```
%token X
%left c
%left aX
%left aXbX
```

(e) If you can't apply the piority to solve the problem, how can you do it with modified CFG? Write the modified CFG.

X $\rightarrow$ c | aXA
A $\rightarrow$ bX | ϵ