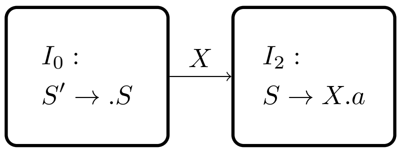
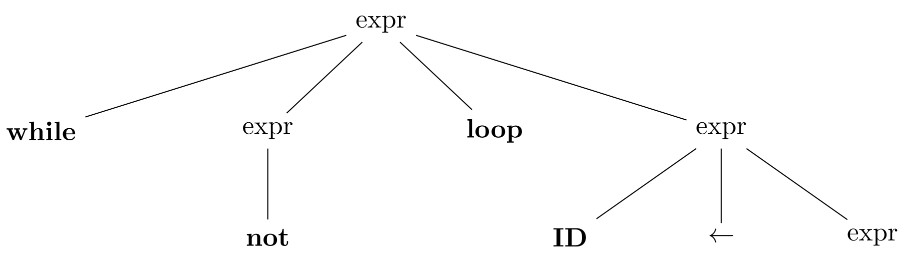
CS131 Compilers: Writing Assignment 2
Due Sunday, April 3, 2021 at 23:59pm

Tian Haoyuan - 2020533013

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.
Sample for `tikz` DFA:



Sample for `tikz` Tree:



I worked with: nobody

1. ($2 * 3 + 4 = 10$ **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 alphabet is $\{1, 2, -, *\}$, all strings of the valid products of integers that is smaller than 0.

$$\begin{array}{lcl} E & \to & N * P | P * N \\ N & \to & E | -P \\ P & \to & N * N | P * P | D \\ D & \to & (1|2)D|(1|2) \end{array}$$

   (b) The alphabet is $\{$ `[`,`]`,`{`,`}`,`,` $\}$, all strings of the valid comma separated sets and list. There must exist one element in set or list.

$$\begin{array}{lcl} S & \to & \{T\}|[T] \\ T & \to & T,T|S|\epsilon \end{array}$$

   (c) The alphabet is $\{0, 1\}$, all strings of the number of 1 's is at most two more than the number of 0 's.

$$\begin{array}{lcl} E & \to & ZSZ1ZSZ1ZSZ|ZSZ1ZSZ|ZSZ \\ S & \to & 0S1S|1S0S|\epsilon \\ Z & \to & 0Z|\epsilon \end{array}$$

(d) All regular languages can be described by a context free grammar, but the converse is not true. Can we write a CFG that translate all five core regular expressions(epsilon, character concatenation, union, Kleene star)?

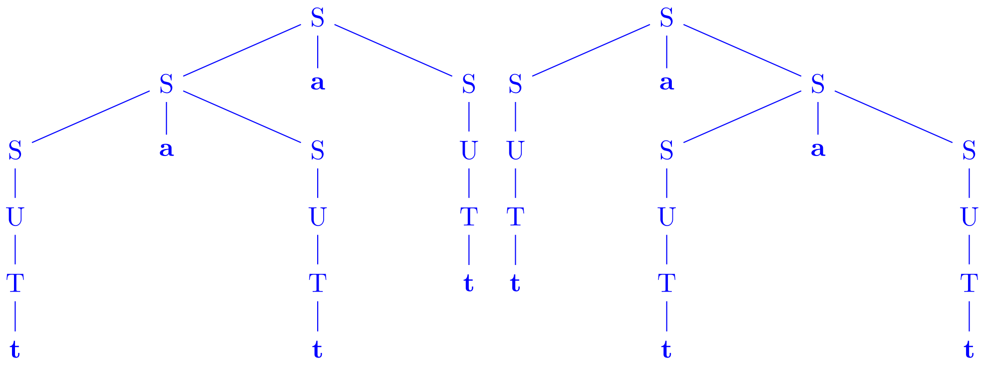
For any regular expression R, your translation should construct a CFG G such that the languages of R and G are equal. Fill in the right-hand sides of the productions below.

$$\begin{array}{ccl} S_\epsilon & \rightarrow & \epsilon \\ S_{c\in\Sigma} & \rightarrow & c \\ S_{AB} & \rightarrow & A\ B \\ S_{A+B} & \rightarrow & A \mid B \\ S_{A^*} & \rightarrow & A\ S_{A^*} \mid \epsilon \end{array}$$

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

$$\begin{array}{ccl} S & \rightarrow & SaS \mid U \\ U & \rightarrow & UuU \mid T \\ T & \rightarrow & t|f|Tn \mid (S) \end{array}$$

(a) 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.
It is ambiguous.
Example: tatat



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

$$\begin{array}{ccl} S & \rightarrow & U\ S' \\ S' & \rightarrow & a\ U\ S' \mid \epsilon \\ U & \rightarrow & T\ U' \\ U' & \rightarrow & u\ T\ U' \mid \epsilon \\ T & \rightarrow & t\ T' \mid f\ T' \mid (\ S\ )\ T' \\ T' & \rightarrow & n\ T' \mid \epsilon \end{array}$$

(c) What advantage does left recursion have over right recursion in shift-reduce parsing?
In shift-reduce parsing, right recursion requires immediate decision on whether to shift or to reduce when it encounters a specific nonterminal, while left recursion only have one option.
For example, consider left recursion $A \rightarrow Aa|a$ and right recursion $A \rightarrow aA|a$, which are obviously equivalent. With left recursion $A \rightarrow Aa|a$, the parser would only have one option when the top of the stack (or the immediate input token) is either A or a, while with right recursion $A \rightarrow aA|a$, the parser (LR(0)) would have a shift-shift conflict with input a, and it may need SLR or LR.

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

$$\begin{array}{lcl} E & \rightarrow & (\ T \\ T & \rightarrow & Q\ ) \\ Q & \rightarrow & q\ A\ q \\ A & \rightarrow & a\ A \\ A & \rightarrow & \epsilon \end{array}$$

(a) Compute the Nullable, First and Follow sets for the grammar.
$Nullable(E) = false, Nullable(T) = false, Nullable(Q) = false, Nullable(A) = true,$
$First(E) = \{(\}, First(T) = \{q\}, First(Q) = \{q\}, First(A) = \{a, \epsilon\},$
$Follow(E) = \{\$\}, Follow(T) = \{\$\}, Follow(Q) = \{)\}, Follow(A) = \{q\}.$

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

	(	)	q	a	$
E	$E \rightarrow (T$				
T			$T \rightarrow Q)$		
Q			$Q \rightarrow qAq$		
A			$A \rightarrow q$	$A \rightarrow aA$	

(c) Is this grammar LALR(1)? Give your reason if it is not LALR(1), otherwise give the LALR(1) parsing table for the grammar.

$$\begin{array}{lcl} 1.\ E & \rightarrow & (\ T \\ 2.\ T & \rightarrow & Q\ ) \\ 3.\ Q & \rightarrow & q\ A\ q \\ 4.\ A & \rightarrow & a\ A \\ 5.\ A & \rightarrow & \epsilon \end{array}$$

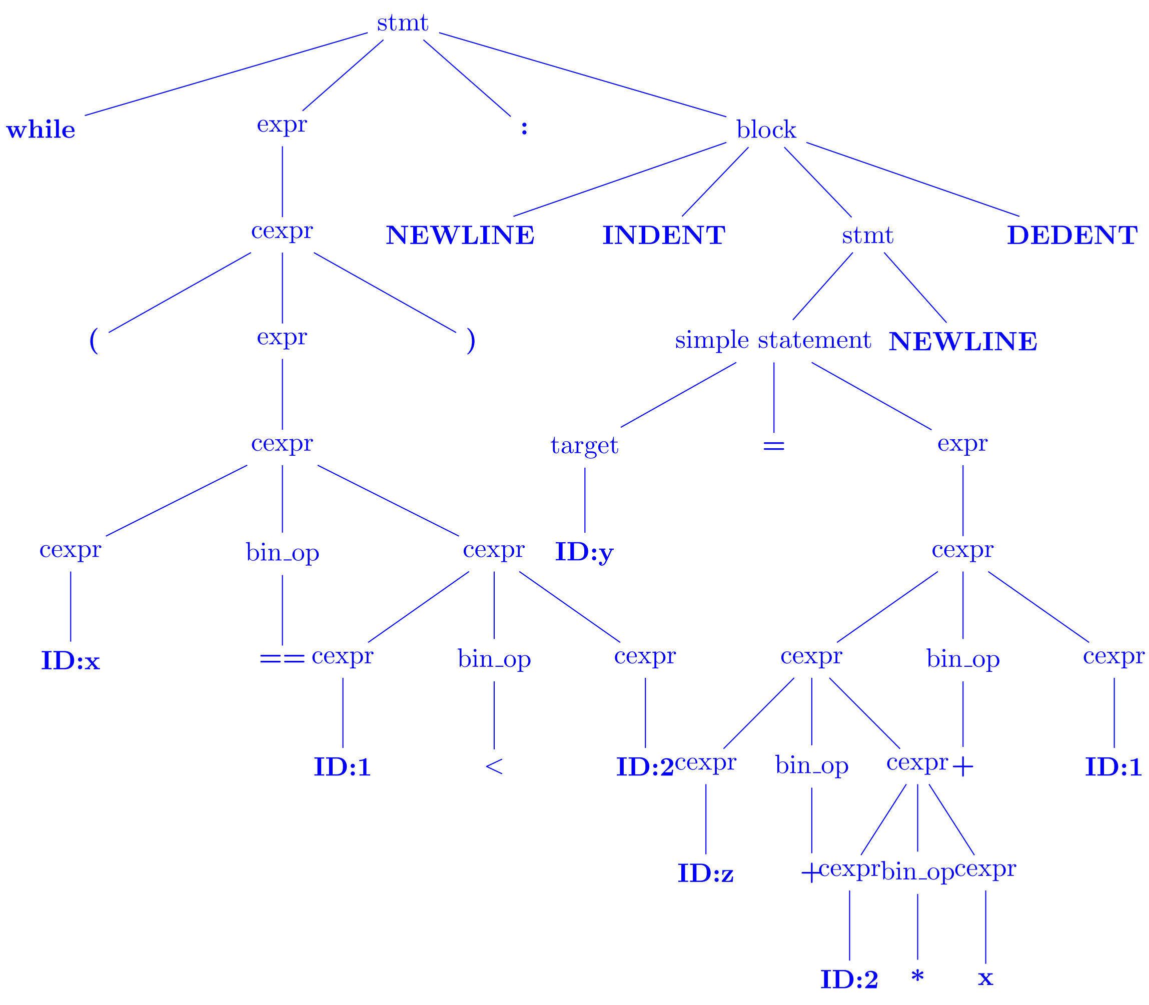
	(	)	q	a	$	E	T	Q	A
0	s2					1			
1					acc				
2			s5				3	4	
3					r1				
4		s7							
5				s8					7
6					r2				
7			s9						
8			r5	s8					10
9		r3							
10			r4						

It is LALR(1).

4. (8 **pts**) Using the context-free grammar for ChocoPy given in the ChocoPy manual, draw a parse tree for the following expression.

```
while (x == 1 < 2):
    y = z + 2 * x + 1
```

Note that the context-free grammar by itself is ambiguous, so you will need to use the precedence and as-



sociativity rules.

5. ($4 \times 4 = 16$ **pts**) Consider the following grammar describing a simplified programming language:

$$\begin{array}{lcl} P & \rightarrow & \epsilon \\ P & \rightarrow & E\ ; \\ E & \rightarrow & 1 \\ E & \rightarrow & E\ +\ E \\ E & \rightarrow & E\ \times\ E \\ E & \rightarrow & E\ ?\ E\ :\ E \\ E & \rightarrow & (\ E\ ) \end{array}$$

P and E are nonterminals, while others are terminals.

(a) Give the First and Follow sets for each nonterminal in the grammar.
$First(E) = \{1, (\}, First(P) = \{\epsilon, 1, (\}$
$Follow(E) = \{;, +, \times, ?, :,)\}, Follow(P) = \{\$\}$

(b) Using this information, produce an (improper) LL(1) parsing table for this grammar. It's improper because some slots will have more than one production.

$$\begin{array}{lll}
1.\ P & \rightarrow & \epsilon \\
2.\ P & \rightarrow & E\ ; \\
3.\ E & \rightarrow & 1 \\
4.\ E & \rightarrow & E\ +\ E \\
5.\ E & \rightarrow & E\ \times\ E \\
6.\ E & \rightarrow & E\ ?\ E\ :\ E \\
7.\ E & \rightarrow & (\ E\ )
\end{array}$$

	1	(	)	;	+	×	?	:	$
E	3,4,5,6	4,5,6,7							
P	2	2							1

(c) Modify the grammar to be LL(1) (unambiguous and with at most one production per table entry).

$$\begin{array}{lll}
P & \rightarrow & \epsilon \\
P & \rightarrow & E\ ; \\
E & \rightarrow & B\ ?\ E\ :\ E \mid B \\
B & \rightarrow & T\ B' \\
B' & \rightarrow & +\ T\ B' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & \times\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid 1
\end{array}$$

(d) repeat part (a),(b) with your result of part (c).
$First(E) = First(T) = First(F) = First(B) = \{1, (\}, First(T') = \{\times, \epsilon\}$,
$First(B') = \{+, \epsilon\}, First(P) = \{\epsilon, 1, (\}$,
$Follow(P) = \{\$\}, Follow(E) = \{;,), :\}$,
$Follow(B) = Follow(B') = \{;,), :, ?\}$,
$Follow(T) = Follow(T') = \{+, ;,), :, ?\}, Follow(F) = \{\times, +, ;,), :, ?\}$.

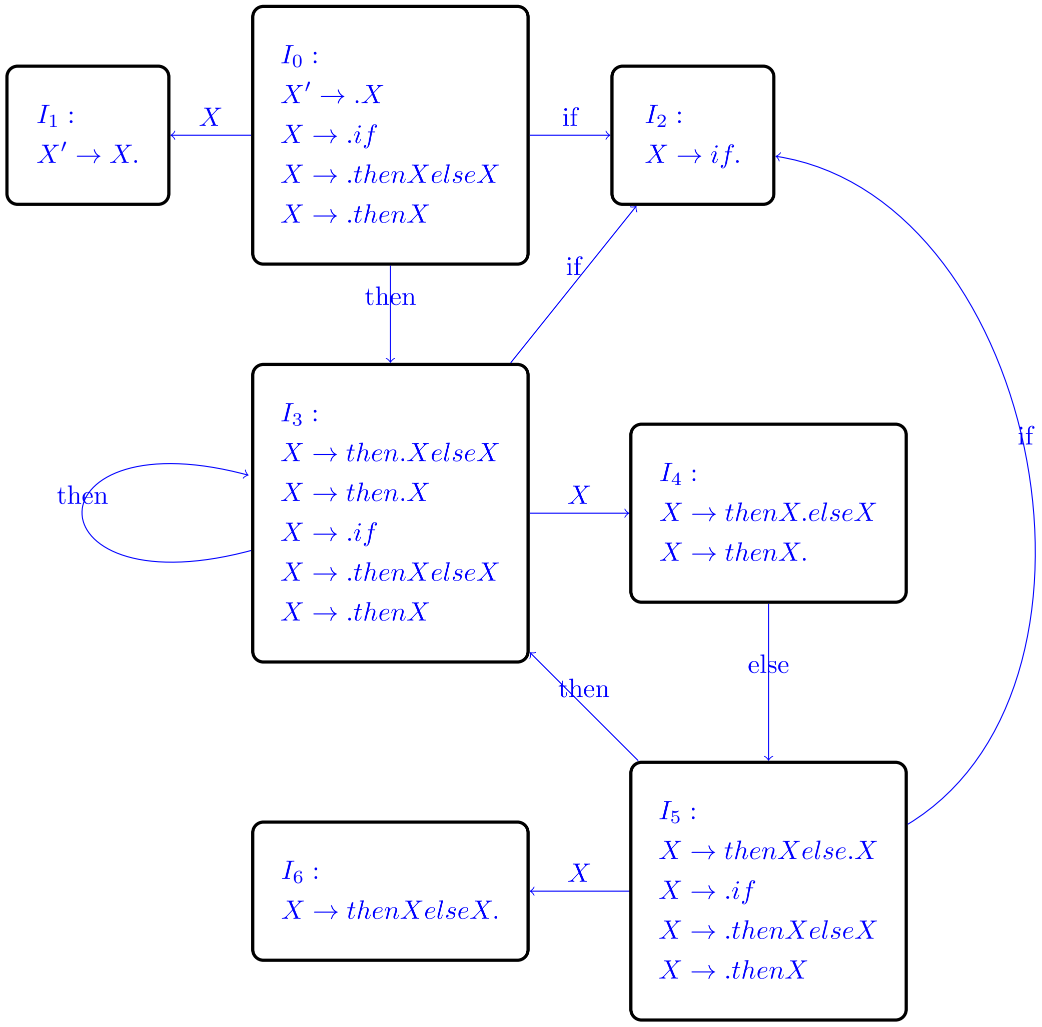
$$\begin{array}{lll}
1.\ P & \rightarrow & \epsilon \\
2.\ P & \rightarrow & E\ ; \\
3.\ E & \rightarrow & B\ ?\ E\ :\ E \\
4.\ E & \rightarrow & B \\
5.\ B & \rightarrow & T\ B' \\
6.\ B' & \rightarrow & +\ T\ B' \\
7.\ B' & \rightarrow & \epsilon \\
8.\ T & \rightarrow & F\ T' \\
9.\ T' & \rightarrow & \times\ F\ T' \\
10.\ T' & \rightarrow & \epsilon \\
11.\ F & \rightarrow & (\ E\ ) \\
12.\ F & \rightarrow & 1
\end{array}$$

	1	(	)	;	+	×	?	:	$
P	2	2							1
E	3	3							
B	5	5							
B’			7	7	6		7	7	
T	8	8							
T’			10	10	10	9	10	10	
F	12	11							

6. ($3\times2+5\times2 = 16$ **pts**) Consider the following CFG, which has the set of terminals $T = \{\textbf{if}, \textbf{else}, \textbf{then}\}$.

$$X \rightarrow \textbf{if} \mid \textbf{then}X\textbf{else}X \mid \textbf{then}X$$

(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.
In state 4 (I_4), there is a shift-reduce conflict, that when the next input is terminal 'else', it can shift to state 5 (I_5), as well as be reduced by $X \to thenX$.

(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 **then then if else then**.

stack	input	action	output
0	then then if else then $	shift 3	
0 then 3	then if else then $	shift 3	
0 then 3 then 3	if else then $	shift 2	
0 then 3 then 3 if 2	else then $	reduce by $X \to if$	$X \to if$
0 then 3 then 3 X 4	else then $	shift 5	
0 then 3 then 2 X 4 else 5	then $	shift 3	
0 then 3 then 2 X 4 else 5 then 3	$	error	

(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.
By adding precendence to token IF and ELSE, respectively correspond to string "if" and "else" in tokenizer. Here's the first part of the parser (.y file).

```
/** Your answer here */
%{
```

```
#include <stdio.h>
%}
%token IF THEN ELSE
%nonassoc THEN
%nonassoc ELSE
%start X
```

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

$$\begin{aligned} X &\rightarrow matchedstmt \mid unmatchedstmt \\ matchedstmt &\rightarrow \textbf{then}\ matchedstmt\ \textbf{else}\ matchedstmt \mid \textbf{if} \\ unmatchedstmt &\rightarrow \textbf{then}\ X \mid \textbf{then}\ matchedstatement\ \textbf{else}\ unmatchedstmt \end{aligned}$$