CS3342 – Assignment 2
due Feb. 17, 2022
2-day no-penalty extension until: Feb. 19, 11:55pm
(SRA's cannot be used to extend further)

1. (40pt) Consider the language:

$$P = \{w \mid w \in \{\texttt{(}, \texttt{)}, \texttt{[}, \texttt{]}\}^*\texttt{\$\$},\ \text{all parentheses in } w \text{ are properly balanced}\}\ .$$

   (a) (10pt) Construct a grammar, G_1, for P that is LL(1). Build its LL(1) parse table (as in Fig. 2.20) to prove that it is LL(1).

   (b) (10pt) Construct a grammar, G_2, for P that is SLR(1) but not LL(1). Build its SLR(1) parse table (as in Fig. 2.28) to prove that it is SLR(1). Prove also that it is not LL(1).

   (c) (4pt) Construct a grammar, G_3, for P that is not SLR(1). Prove that it is not SLR(1).

   (d) (4pt) Show the parse tree and the left derivation for the string `[([])()]$$` in G_1.

   (e) (4pt) Show the trace of the table-driven LL(1) parse (as in Fig. 2.21) using G_1 for the same string.

   (f) (4pt) Show the parse tree and the right derivation for the string `[([])()]$$` in G_2.

   (g) (4pt) Show the trace of the table-driven SLR(1) parse (as in Fig. 2.30) using G_2 for the same string.

   The grammars $G_{1..3}$ above must have a production $P \rightarrow S\texttt{\$\$}$, with P not appearing anywhere else in the grammar. The parse tables are built as done in the textbook, not as done by jflap. You can use jflap to help but you need to modify its output. Do not upload screenshots from jflap. Write your own tables.

2. (30pt) Expressions can be written without the need of parentheses in postfix form (also called reverse Polish notation). The name comes from the fact that the operator comes after the operands: $a + b$ is written as $a\,b\,+$. A postfix expression can be easily evaluated using a stack.

   (a) (10pt) Using the underlying grammar from Fig. 4.1, write an S-attributed grammar that associates with the root of the parse tree the postfix expression corresponding to the (infix) expression produced by the tree.

   (b) (5pt) Use this S-attributed attributed grammar to draw the annotated parse tree for the expression `(- 3 + 2) * 7 - 1`. Show the attribute flow (arrows and values).

   (c) (10pt) Using the underlying grammar from Fig. 4.3, write an L-attributed grammar that associates with the root of the parse tree the postfix expression corresponding to the (infix) expression produced by the tree.

   (d) (5pt) Use this L-attributed attributed grammar to draw the annotated parse tree for the expression `(- 3 + 2) * 7 - 1`. Show the attribute flow (arrows and values).

3. (30pt) Consider the grammar below for floating point numbers:

$$\begin{aligned}
\textit{Float} &\longrightarrow \textit{Left}\ \texttt{.}\ \textit{Right}\\
\textit{Left} &\longrightarrow \textit{Digit}\ \textit{Left_more}\\
\textit{Left_more} &\longrightarrow \textit{Left}\\
\textit{Left_more} &\longrightarrow \varepsilon\\
\textit{Right} &\longrightarrow \textit{Digit}\ \textit{Right_more}\\
\textit{Right_more} &\longrightarrow \textit{Right}\\
\textit{Right_more} &\longrightarrow \varepsilon\\
\textit{Digit} &\longrightarrow \texttt{0|1|2|3|4|5|6|7|8|9}
\end{aligned}$$

(a) (20pt) Using the above grammar, write an attribute grammar which contains an attribute *val* that stores the value of a number such that the *val* of any internal node is the sum of the *val*'s of its children. Besides the *val* attribute, you can use only one other attribute. The grammar does not have to be L-attributed.

(b) (10pt) Draw the annotated parse tree for the string `12.34`. Show the attribute flow (arrows and values).

READ ME! Submit your answers as a *single pdf file* in OWL. Solutions should be typed but readable (by others!) hand-written solutions are acceptable. Source code, if required, is submitted as separate files.

JFLAP: You are allowed to use JFLAP to help you solve the assignment. Make sure you understand what it does; JFLAP will not be available during in-person exams!

LaTeX: For those interested, the best program for scientific writing is LaTeX. It is far superior to all the other programs, it is free, and you can start using it in minutes; here is an introduction: `https://tobi.oetiker.ch/lshort/lshort.pdf`

1. (40pt) Consider the language:

$$P = \{w \mid w \in \{(,), [,]\}^* \$\$, \text{ all parentheses in } w \text{ are properly balanced}\} .$$

(a) (10pt) Construct a grammar, G_1, for P that is LL(1). Build its LL(1) parse table (as in Fig. 2.20) to prove that it is LL(1).

(b) (10pt) Construct a grammar, G_2, for P that is SLR(1) but not LL(1). Build its SLR(1) parse table (as in Fig. 2.28) to prove that it is SLR(1). Prove also that it is not LL(1).

(c) (4pt) Construct a grammar, G_3, for P that is not SLR(1). Prove that it is not SLR(1).

(d) (4pt) Show the parse tree and the left derivation for the string `[([])()]$$` in G_1.

(e) (4pt) Show the trace of the table-driven LL(1) parse (as in Fig. 2.21) using G_1 for the same string.

(f) (4pt) Show the parse tree and the right derivation for the string `[([])()]$$` in G_2.

(g) (4pt) Show the trace of the table-driven SLR(1) parse (as in Fig. 2.30) using G_2 for the same string.

The grammars $G_{1..3}$ above must have a production $P \to S\$\$$, with P not appearing anywhere else in the grammar. The parse tables are built as done in the textbook, not as done by jflap. You can use jflap to help but you need to modify its output. Do not upload screenshots from jflap. Write your own tables.

(a)

1 P -> S $$
2 S -> (S) S
3 S -> [S] S
4. S -> ε

	(	)	[	]	$$
P	1		1		1
S	2	4	3	4	4.

=> to prove one grammar is LL(1), we have to say each cell in the phrase table has at most one entry.

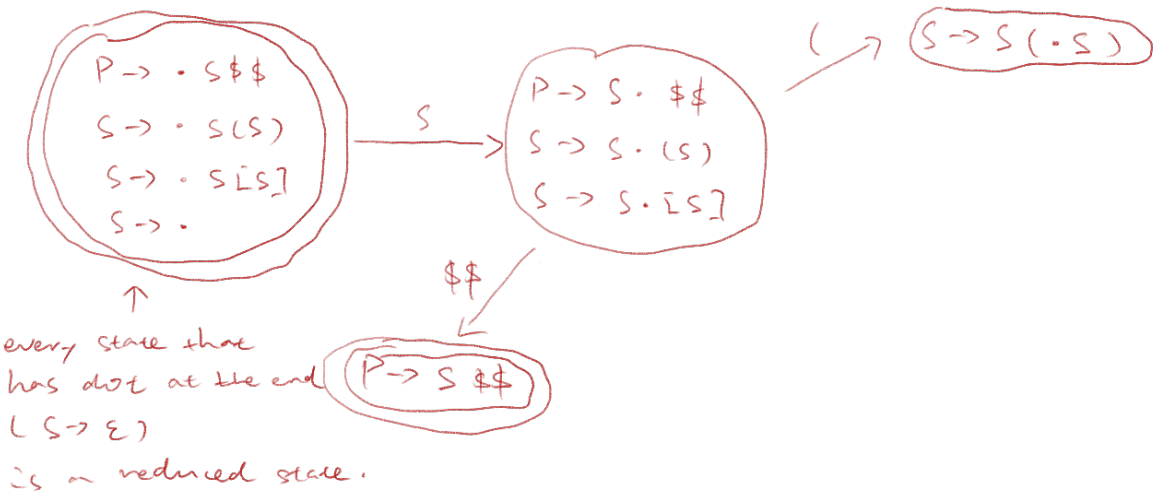
the difference between SLR(1) and LL(1) is that SLR(1) could have a different output with same starting.

(b)

1 P -> S $$
2 S -> S (S)
3 S -> S [S].
4. S -> ε

	(	)	[	]	$$
P					
S					

	FIRST	FOLLOW
P	∅	$$,),]
S	ε	$$,),].



(1) not SLR(1): ambiguous => could have two phase tree to explain the grammar.