YOURNAME - SUNETID
CS143 - Written Assignment 2

1. **(8 pts)** Give the context-free grammar (CFG) for each of the following languages:

   (a) **(2 pts)** The set of nested addition expressions formed using the alphabet $\{int, +, (,), [,]\}$. The associativity of the additions must be established with brackets and adjacent brackets should be of different types. For example, "[(int + int) + int]" and "int + [int + (int + int)]" are strings in the language while "int + int + int" and "(int + (int + int))" are not in the language.

$$S \to A + A \mid A + B \mid B + A \mid B + B \mid A \mid B$$
$$A \to int \mid (int) \mid (B + B)$$
$$B \to int \mid [int] \mid [A + A]$$

   (b) **(2 pts)** The set of all strings over the alphabet $\{a, b\}$ with more "a"s than "b"s

$$S \to SM \mid MS \mid A$$
$$A \to Aa \mid a$$
$$M \to bMA \mid AMb \mid \epsilon$$

   (c) **(2 pts)** The set of all strings over the alphabet $\{a, b, c\}$ in the language $L : \{a^i b^j c^k \mid j = i + k\}$.

$$S \to AC$$
$$A \to \epsilon \mid aAb$$
$$C \to \epsilon \mid bCc$$

   (d) **(2 pts)** The set of all strings over the alphabet $\{0, 1\}$ in the language $L : \{0^i 1^j 0^k \mid i \neq j \vee j \neq k\}$.

$$S \to 0ZTZ \mid T1YZ \mid Z1YU \mid ZU0Z$$
$$T \to 0T1 \mid \epsilon$$
$$U \to 1U0 \mid \epsilon$$
$$Y \to 1Y \mid \epsilon$$
$$Z \to 0Z \mid \epsilon$$

2. (a) Left factor the following grammar:

$$P \to x \;\Big|\; P \Rightarrow P \;\Big|\; P \Leftarrow P \;\Big|\; (P) \Leftrightarrow (P) \;\Big|\; P; P;$$

Answer

There was some confusion about the requirements of left factoring. For the purposes of this class, we use left factoring to only mean factoring shared syntactic prefixes from the productions of a nonterminal. Some students may have chosen to additionally (i) remove left recursion; or (ii) factor out terminals which are common to the FIRST sets of multiple productions from one nonterminal (which is necessary to prevent conflicts in the LL(1) parse table). We accept solutions with or without this additional work.

This solution performs the minimal left factoring:

$$P \to x \;\Big|\; PQ \;\Big|\; (P) \Leftrightarrow (P)$$

$$Q \to \Rightarrow P \;\Big|\; \Leftarrow P \;\Big|\; ; P;$$

This solution performs left factoring, removes left recursion, and removes conflicts from the LL(1) table:

$$P \to xQ \;\Big|\; (P) \Leftrightarrow (P)Q$$

$$Q \to \Rightarrow P \;\Big|\; \Leftarrow P \;\Big|\; ; P; \;\Big|\; \epsilon$$

(b) Eliminate left recursion from the following grammar:

$$A \to Aa \;\Big|\; ABA \;\Big|\; \epsilon$$

$$B \to Bb \;\Big|\; BB \;\Big|\; \epsilon$$

Answer

$$A \to BAA \;\Big|\; aA \;\Big|\; \epsilon$$

$$B \to bB \;\Big|\; \epsilon$$

3. **(9 pts)** Consider the following grammar:

$$A \to uA \mid wuA \mid B + B \mid \epsilon$$
$$B \to bB \mid CB \mid \epsilon$$
$$C \to cAw$$

A, B, and C are the non-terminals in the grammar.

(a) Construct the FIRST sets for the grammar.

i. $A : \{b, c, u, w, +, \epsilon\}$
ii. $B : \{b, c, \epsilon\}$
iii. $C : \{c\}$

(b) Construct the FOLLOW sets for the grammar.

i. $A : \{w, \$\}$
ii. $B : \{w, +, \$\}$
iii. $C : \{b, c, w, +, \$\}$

(c) Construct the LL(1) parse table for the grammar.

	b	c	u	w	$+$	$\$$
A	$B + B$	$B + B$	uA	ϵ	$B + B$	ϵ
B	bB	CB		ϵ	ϵ	ϵ
C		cAw				

There is a conflict in the table when A is at the top of the stack and w is the first character in the input. Therefore, the grammar is not actually LL(1). However, we decide to resolve the conflict in favor of the $A \to \epsilon$ transition in order to parse the input in the next part.

(d) Show the sequence of stack and input configurations that occur during an LL(1) parse of the string "$cuw + b$". The stack should contain a single A at the beginning of the parse.

Matched	Stack	Input	Action
	$A\$$	$cuw + b$	
	$B + B\$$	$cuw + b$	output $A \to B + B$
	$CB + B\$$	$cuw + b$	output $B \to CB$
	$cAwB + B\$$	$cuw + b$	output $C \to cAw$
c	$AwB + B\$$	$uw + b$	match c
c	$uAwB + B\$$	$uw + b$	output $A \to uA$
cu	$AwB + B\$$	$w + b$	match u
cu	$wB + B\$$	$w + b$	output $A \to \epsilon$
cuw	$B + B\$$	$+b$	match w
cuw	$+B\$$	$+b$	output $B \to \epsilon$
cuw+	$B\$$	b	match $+$
cuw+	$bB\$$	b	output $B \to bB$
cuw+b	$B\$$		match b
cuw+b	$B\$$		output $B \to \epsilon$
cuw+b	$\$$		accept

4. Suppose you encounter the following grammar G:

$$
\begin{aligned}
S' &\to S \\
S &\to Aa \\
S &\to Bb \\
A &\to Ac \\
A &\to \epsilon \\
B &\to Bc \\
B &\to \epsilon
\end{aligned}
$$

You want to implement G using an SLR(1) parser (note that we have already added the $S' \to S$ production for you).

(a) Why is left recursion preferable in shift-reduce parsing? [Hint: consider left and right recursive grammars for the language a^*. What happens if your input has a million a's?]
Answer:
Consider what happens with the right recursive grammar for a^*, $S \to aS \mid \epsilon$. The resulting shift-reduce machine will shift the entire input onto the stack, before performing the first reduction. It will accept all strings in the language, but it requires an unbounded stack size. Now consider $S \to Sa \mid \epsilon$. This machine will alternate between shifting an a onto the stack, and reducing with the first input character. The required amount of stack space is small regardless of the input length. Using left recursion in grammars for shift-reduce parsers bounds the stack space.

(b) Construct the first state of the LR(0) machine, compute the FOLLOW sets of A and B, and point out the conflicts that prevent the grammar from being SLR(1).
Answer:
Here is the first state of the LR(0) machine.

$$
\begin{aligned}
S' &\to .S \\
S &\to .Aa \\
S &\to .Bb \\
A &\to .Ac \\
A &\to .\epsilon \\
B &\to .Bc \\
B &\to .\epsilon
\end{aligned}
$$

We have that FOLLOW(A) $= \{a, c\}$ and FOLLOW(B) $= \{b, c\}$.
We have a reduce-reduce conflict between production ($A \to \epsilon$) and production 7 ($B \to \epsilon$) here, so the grammar is not SLR(1).

(c) Show that changing productions 4 and 6 to be right-recursive solves the problem in this case (show that the grammar is SLR(1)). Explain the intuition behind this result.
Answer:
We change productions 4 and 6 to be right recursive, as follows:

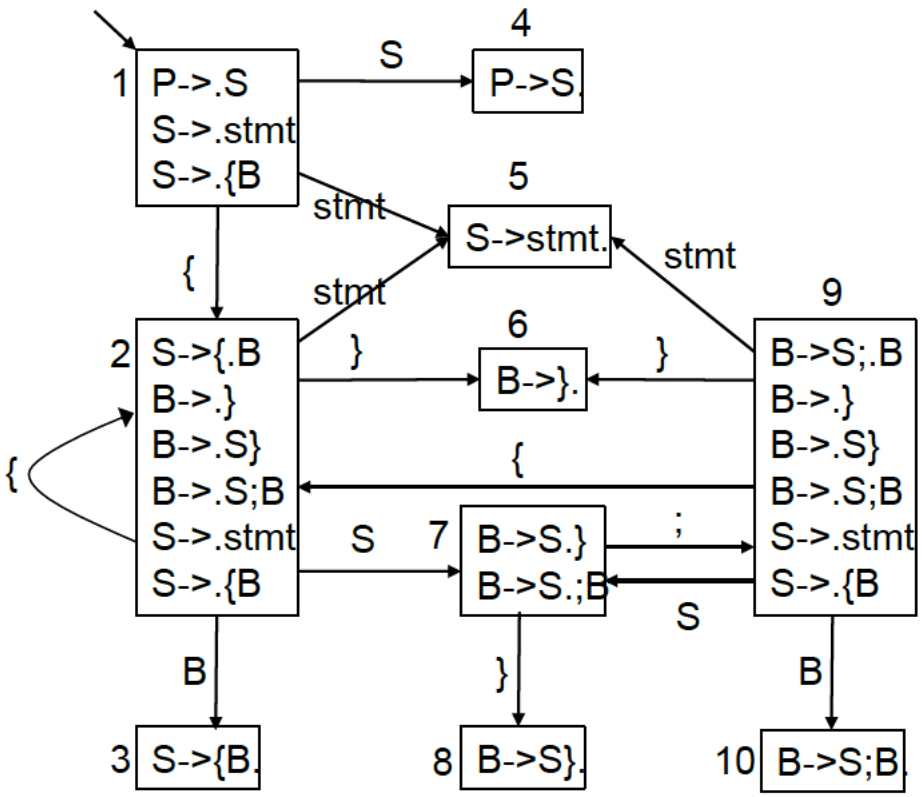
$$
\begin{aligned}
A &\to cA \\
B &\to cB
\end{aligned}
$$

As a result, c is no longer in the Follow sets of either A or B. Since the follow sets are now disjoint, the reduce-reduce conflict in the SLR(1) table is removed. Intuitively, using right recursion causes the parser to defer the decision of whether to reduce a string of c's to A's or B's. When we reach the end of the input the final character gives us enough information to determine which reduction to perform.

5. Consider the following CFG, which has the set of terminals $T = \{\textbf{stmt}, \textbf{\{}, \textbf{\}}, \textbf{;}\}$. This grammar describes the organization of statements in blocks for a fictitious programming language. Blocks can have zero or more statements as well as other nested blocks, separated by semicolons, where the last semicolon is optional. (P is the start symbol here.)

$$P \to S$$
$$S \to \textbf{stmt} \mid \textbf{\{}B$$
$$B \to \textbf{\}} \mid S\textbf{\}} \mid S\textbf{;}B$$

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



(b) Identify any shift-reduce and reduce-reduce conflicts in this grammar under the SLR(1) rules.
There are no conflicts.

(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 **{stmt;stmt}**

Configuration	DFA Halt State	Action
$\vert \{stmt; stmt\}\$$	1	shift
$\{ \vert stmt; stmt\}\$$	2	shift
$\{stmt \vert ; stmt\}\$$	5 $';' \in Follow(S)$	reduce $S \to stmt$
$\{S \vert ; stmt\}\$$	7	shift
$\{S; \vert stmt\}\$$	9	shift
$\{S; stmt \vert \}\$$	5 $\} \in Follow(S)$	reduce $S \to stmt$
$\{S; S \vert \}\$$	7	shift
$\{S; S\} \vert \$$	8 $'\$' \in Follow(B)$	reduce $B \to S\}$
$\{S; B \vert \$$	10 $'\$' \in Follow(B)$	reduce $B \to S; B$
$\{B \vert \$$	3 $'\$' \in Follow(S)$	reduce $S \to \{B$
$S \vert \$$	4 $'\$' \in Follow(P)$	reduce $P \to S$
$P \vert \$$		ACCEPT