YOURNAME - SUNETID

CS143 2020 - Written Assignment 2

Due Monday, May 4, 2020 at 11:59 PM

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

   (a) The set of all strings over the alphabet $\{2, -, +\}$ representing valid arithmetic expressions where each integer in the expression is a single digit and the expression evaluates to some value ≥ 0.
   Example Strings in the Language:

   2+2 2-2+2 -2-2+2+2

   Strings not in the Language:

   -2 -2+22 2++2-2

   Answer:

$$S \rightarrow 2E \mid -2E + 2E \mid 2E - 2E$$
$$E \rightarrow -2E + 2E \mid +2E \mid +2E - 2E \mid \epsilon$$

(b) The set of all strings over the alphabet $\{string, \ , \%s, arg, ", ,\}$ representing valid arguments to the c printf() function. For the purposes of this problem, treat *string* and *arg* as tokens of your language where *string* represents an arbitrary sequence of legal characters in a c string and *arg* represents any arbitrary char* used as the location for a $\%s$. **Note** that ',' and ' ' are in the alphabet.

Example Strings in the Language (surrounded by "):

$$"""" \qquad ""string\ \ \%s\%s\ \ string", arg, arg"$$
$$""\%s \qquad ", \qquad arg"$$

Strings not in the Language (surrounded by "):

$$"arg""string" \qquad ""arg\ \ string"" \qquad "string\ \ \%s,\ \ arg"$$

Answer: First, let's denote ‘ ’ as a white space.

$$\begin{aligned}
S &\to "B \\
B &\to " \mid S_r\%SS_rBA \\
A &\to W, WargW \\
W &\to ‘\ ’ \mid W‘\ ’ \mid \epsilon \\
S_r &\to \epsilon \mid S_r string S_r \mid C \\
C &\to ‘\ ’ \mid , \mid CC \mid \epsilon
\end{aligned}$$

(c) The set of all strings over the alphabet $\{0, 1\}$ in the language $L : \{0^i 1^j 0^k \mid j \leq i + k\}$.
Example Strings in the Language:

00000 000111100 ϵ

Strings not in the Language:

1 000111101

Answer:

$$\begin{aligned}
S &\to TU \\
T &\to ITJ \mid \epsilon \\
U &\to JUK \mid \epsilon \\
J &\to 1 \mid \epsilon \\
I &\to I0 \mid 0 \\
K &\to 0K \mid 0
\end{aligned}$$

(d) The set of all strings over the alphabet $\{[,],\{,\},,\}$ which are sets. We define a set to be a collection of zero or more comma-separated arrays enclosed in an open brace and a close brace. Similarly, we define an array to be a collection of zero or more comma-separated sets enclosed in an open bracket and a close bracket. **Note** that ”,” is in the alphabet.
Example Arrays:

$$[\{\},\{\}] \qquad [] \qquad [\{[],[]\}]$$

Example Sets:

$$\{[]\} \qquad \{\}, \qquad \{[\{\}],[]\}$$

Example Strings in the Language:

$$\{\} \qquad \{[],[\{[]\}]\} \qquad \{[\{\},\{\},\{\}],[]\}$$

Strings not in the Language:

$$[] \qquad \{\{\}\} \qquad \{[[]]\}$$

Answer:

$$\begin{aligned} S &\to \{T\} \mid \{\} \\ T &\to U,T \mid U \\ U &\to [V] \mid [] \\ V &\to S,V \mid S \end{aligned}$$

2. (a) Left factor the following grammar:

$$\begin{aligned}
S &\to I \mid I - J \mid I + K \\
I &\to (J - K) \mid (J) \\
J &\to K1 \mid K2 \\
K &\to K3 \mid \epsilon
\end{aligned}$$

Answer:

$$\begin{aligned}
S &\to IV \\
V &\to \epsilon \mid -J \mid +K \\
I &\to (JW \\
W &\to -K) \mid) \\
J &\to KZ \\
Z &\to 1 \mid 2 \\
K &\to K3 \mid \epsilon
\end{aligned}$$

(b) Eliminate left recursion from the following grammar:

$$\begin{aligned} S &\to STS \mid ST \mid T \\ T &\to Ta \mid Tb \mid U \\ U &\to T \mid c \end{aligned}$$

Answer:

$$\begin{aligned} S &\to TS' \\ S' &\to TSS' \mid TS' \mid \epsilon \\ T &\to cT' \\ T' &\to aT' \mid bT' \mid \epsilon \end{aligned}$$

3. Consider the following CFG, where the set of terminals is $\Sigma = \{a, b, \#, \%, !\}$:

$$\begin{aligned} S &\to \%aT \mid U! \\ T &\to aS \mid baT \mid \epsilon \\ U &\to \#aTU \mid \epsilon \end{aligned}$$

(a) Construct the FIRST sets for each of the nonterminals.
Answer:

- S: $\{\%, \#, !\}$
- T: $\{a, b, \epsilon\}$
- U: $\{\#, \epsilon\}$

(b) Construct the FOLLOW sets for each of the nonterminals.
Answer:

- S: $\{\#, !, \$\}$
- T: $\{\#, !, \$\}$
- U: $\{!\}$

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

Answer:

	a	b	#	%	!	\$
S			$S \to U!$	$S \to \%aT$	$S \to U!$	
T	$T \to aS$	$T \to baT$	$T \to \epsilon$		$T \to \epsilon$	$T \to \epsilon$
U			$U \to \#aTU$		$U \to \epsilon$	

(d) Show the sequence of stack, input and action configurations that occur during an LL(1) parse of the string "$\#abaa\%aba!$". At the beginning of the parse, the stack should contain a single S.

Answer:

Stack	Input	Action
$S\$$	$\#abaa\%aba!\$$	output $S \to U!$
$U!\$$	$\#abaa\%aba!\$$	output $U \to \#aTU$
$\#aTU!\$$	$\#abaa\%aba!\$$	match $\#$
$aTU!\$$	$abaa\%aba!\$$	match a
$TU!\$$	$baa\%aba!\$$	output $T \to baT$
$baTU!\$$	$baa\%aba!\$$	match b
$aTU!\$$	$aa\%aba!\$$	match a
$TU!\$$	$a\%aba!\$$	output $T \to aS$
$aSU!\$$	$a\%aba!\$$	match a
$SU!\$$	$\%aba!\$$	output $S \to \%aT$
$\%aTU!\$$	$\%aba!\$$	match $\%$
$aTU!\$$	$aba!\$$	match a
$TU!\$$	$ba!\$$	output $T \to baT$
$baTU!\$$	$ba!\$$	match b
$aTU!\$$	$a!\$$	match a
$TU!\$$	$!\$$	output $T \to \epsilon$
$U!\$$	$!\$$	output $U \to \epsilon$
$!\$$	$!\$$	match $!$
$\$$	$\$$	accept

4. What advantage does left recursion have over right recursion 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 initially reduce by $S \to \epsilon$ and then alternate between shifting an "a" on to the stack and reducing by $S \to Sa$. The stack space used is thus constant. In general, using left recursion in grammars for shift-reduce parsers helps limit the size of the stack.

5. Consider the following grammar G over the alphabet $\Sigma = \{a, b, c\}$:

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

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

(a) 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 5 ($A \to \epsilon$) and production 7 ($B \to \epsilon$), so the grammar is not SLR(1).

(b) Show modifications to production 4 ($A \rightarrow Ac$) and production 6 ($B \rightarrow Bc$) that make the grammar SLR(1) while having the same language as the original grammar G. Explain the intuition behind this result.

Answer: We change productions 4 and 6 to be right recursive, as follows:

$$A \rightarrow cA$$
$$B \rightarrow cB$$

As a result, c is no longer in the follow sets of either A nor 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.