# YOURNAME – SUNETID

# CS143 Spring 2024 – Written Assignment 2

Due Monday, April 29, 2024 11:59 PM PDT

This assignment covers context free grammars and parsing. 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. Assignments can be submitted electronically through Gradescope as a PDF by 11:59 PM PDT. Please review the the course policies for more information: `https://web.stanford.edu/class/cs143/policies/`. A LaTeX template for writing your solutions is available on the course website. If you need to draw parse trees in LaTeX, you may use the `forest` package: `https://ctan.org/pkg/forest`.

1. Give a context-free grammar (CFG) for each of the following languages. Any grammar is acceptable—including ambiguous grammars—as long as it has the correct language. The start symbol should be $S$.

   (a) The set of all strings over the alphabet $\{3, 4, *, -\}$ representing valid products of integers where the expression evaluates to some positive odd value.

   Example Strings in the Language:

   3     3*433     -3*3*-3343

   Strings not in the Language:

   $\varepsilon$     --3     -3*3*-3344

   **Solution:**

   $$S \to S * S \mid P \mid N * N \mid N * S * N$$
   $$P \to 4P \mid 3P \mid 3$$
   $$N \to -P$$

   (b) The set of all strings over the alphabet $\{$`a`, `b`, `[`, `]`, `,`$\}$ representing nested lists of $a$ and $b$'s where each list has an even length. Nested lists are defined as comma seperated sequences of elements enclosed with a pair of square brackets [], where an element may be an $a$, $b$, or another list. Example Strings in the Language:

   $[]$     $[a, [[b, a], a]]$     $[[], [a, b], [a, a], [[], [b, b]]]$

   Strings not in the Language:

   $\varepsilon$     $b$     $[a, b, ]$     $[a$     $[a, [[b], a]]$

   **Solution:**

   $$S \to [T]$$
   $$T \to \varepsilon \mid U$$
   $$U \to E, E \mid E, E, U$$
   $$E \to S \mid a \mid b$$

(c) The set of all strings over the alphabet $\{0, 1\}$ where the number of 1's is more than the number of 0's.

Example Strings in the Language:

1   101   001111

Strings not in the Language:

$\varepsilon$   0   01100101

**Solution:**

$$
\begin{aligned}
S &\to T1T \\
T &\to T1T0T \mid T1T \mid T0T1T \mid \varepsilon
\end{aligned}
$$

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

Example Strings in the Language:

0   100011   $\varepsilon$

Strings not in the Language:

1   010   101

**Solution:**

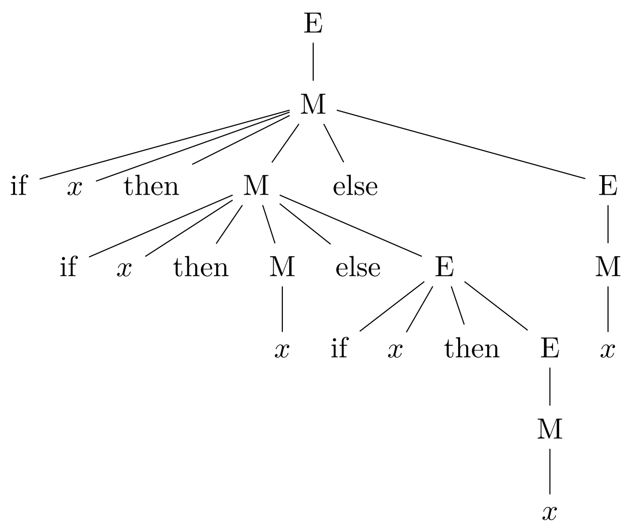
$$
\begin{aligned}
S &\to TZU \\
T &\to 1T0 \mid \varepsilon \\
U &\to 0U1 \mid \varepsilon \\
Z &\to Z0 \mid \varepsilon
\end{aligned}
$$

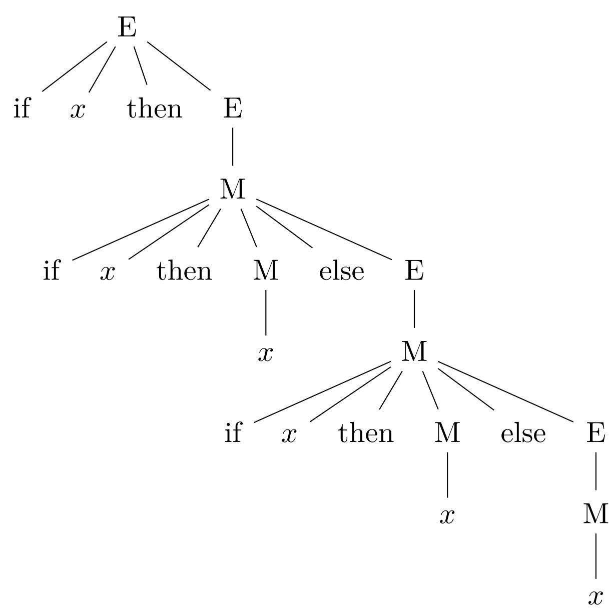
2. Consider the following grammar for if-then-else expressions that involve a variable $x$:

$$E \to \text{if } x \text{ then } E \mid M$$
$$M \to \text{if } x \text{ then } M \text{ else } E \mid x$$

Is this grammar ambiguous or not? If yes, give an example of an expression with two different parse trees and draw the two parse trees. If not, explain why that is the case.

**Solution:** The grammar is ambigous. To see why, consider the expression "if $x$ then if $x$ then $x$ else if $x$ then $x$ else $x$". This expression has two parse trees under this grammar, shown below:



3. (a) Left factor the following grammar:

$$
\begin{aligned}
S &\to T \mid T + T \mid T * T \\
T &\to Ta \mid Tb \mid cU \\
U &\to U0 \mid U1 \mid \varepsilon
\end{aligned}
$$

**Solution:**

$$
\begin{aligned}
S &\to TS' \\
S' &\to \varepsilon \mid +T \mid *T \\
T &\to TT' \mid cU \\
T' &\to a \mid b \\
U &\to UU' \mid \varepsilon \\
U' &\to 0 \mid 1
\end{aligned}
$$

(b) Eliminate left recursion from the following grammar:

$$
\begin{aligned}
S &\to S + S \mid (S) \mid T \\
T &\to UUb \mid Ta \\
U &\to TTc \mid c
\end{aligned}
$$

**Solution:**

$$
\begin{aligned}
S &\to (S)S' \mid TS' \\
S' &\to +SS' \mid \varepsilon \\
T &\to cUbT' \\
T' &\to TcUbT' \mid aT' \mid \varepsilon \\
U &\to TTc \mid c
\end{aligned}
$$

4. Consider the following CFG, where the set of terminals is $\{a, b, c, <, >\}$:

$$
\begin{aligned}
S &\to T < U \mid b > U \\
T &\to aS < S \mid cU \mid > b \\
U &\to > Ta \mid < Sb
\end{aligned}
$$

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

**Solution:**

- $S : \{a, b, c, >\}$
- $T : \{a, c, >\}$
- $U : \{<, >\}$

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

**Solution:**

- $S : \{a, b, <, \$\}$
- $T : \{a, <\}$
- $U : \{a, b, <, \$\}$

(c) Construct the LL(1) parsing table for the grammar. Where applicable, list all possible productions for every parse table cell.

**Solution:**

| | a | b | c | $<$ | $>$ | \$ |
|---|---|---|---|---|---|---|
| $S$ | $T < U$ | $b > U$ | $T < U$ | | $T < U$ | |
| $T$ | $aS < S$ | | $cU$ | | $> b$ | |
| $U$ | | | | $< Sb$ | $> Ta$ | |

(d) Show the sequence of stack, input, and action configurations that occur during an LL(1) parse of the string "$> b <>> ba$". At the beginning of the parse, the stack should contain a single $S$. The acceptable actions include: "out <production>", "match <terminal>", "accept", and "error".

**Solution:**

| Stack | Input | Action |
|---:|---:|---|
| $S\$$ | $> b <>> ba\$$ | output $S \to T < U$ |
| $T < U\$$ | $> b <>> ba\$$ | output $T \to > b$ |
| $> b < U\$$ | $> b <>> ba\$$ | match $>$ |
| $b < U\$$ | $b <>> ba\$$ | match $b$ |
| $< U\$$ | $<>> ba\$$ | match $<$ |
| $U\$$ | $>> ba\$$ | output $U \to > Ta$ |
| $> Ta\$$ | $>> ba\$$ | match $>$ |
| $Ta\$$ | $> ba\$$ | output $T \to > b$ |
| $> ba\$$ | $> ba\$$ | match $>$ |
| $ba\$$ | $ba\$$ | match $b$ |
| $a\$$ | $a\$$ | match $a$ |
| $\$$ | $\$$ | accept |

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

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

You want to implement G using an SLR(1) parser (note that we have already added the S' → 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)
**Solution:** Here is the first state of the LR(0) machine:

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

We have that FOLLOW$(A)$ = $\{a, c\}$ and FOLLOW$(B)$ = $\{b, c\}$. We have a reduce-reduce conflict between producion 5 ($A \to \varepsilon$) and production 7 ($B \to \varepsilon$), so the grammar is not SLR(1).

(b) Show modifications to production 4 (A → Ac) and production 6 (B → Bc) that make the grammar SLR(1) while having the same language as the original grammar G. Explain the intuition behind this result.
**Solution:** 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 set of either $A$ nor $B$. Since the follow sets are now disjoint, the reduce-educe 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.