Example CFLs

Discussion 1C

2-1-19

Disclaimer: The following are examples of PDAs and CFGs for different CFLs. However, the constructions as shown do not satisfy the requirements of a complete proof as I do not completely explain why $x$ is in the CFL if and only if $x$ is accepted by the PDA or can be derived from the CFG. Thus, the work shown here would not receive full credit on the homework. They are merely here to help you understand the intuition behind how PDAs and CFGs are constructed. Note also that there may be multiple CFGs or PDAs for any one language and that the PDAs and CFGs given for any one problem do not necessarily correspond to each other in terms of the intuition behind them.
1 \[ L = \{ w \in \{0, 1\}^* \mid w \text{ is of odd length} \} \]

1.1 PDA

**Intuition:** If \( w \) is odd, then \(|w| = 2n + 1\) for some \( n \), and \( w = abc \) where \(|a| = |c| = n\) and \( b \) is a single character. For the first \( n \) characters, we will push an arbitrary symbol \( x \) to the stack to count the size of \( n \). For the middle character, we will consume the character, but leave the stack alone. Then, for the last \( n \) characters, we will pop an \( x \) off the stack for each character. We accept if the stack is empty after consuming all input. This means that we had the same number of characters before and after the middle character, meaning that the string is odd. But how do we know what \( n \) is? After each input character, we will nondeterministically guess that we have reached the \( n^{th} \) character. If we guessed wrong, then that branch of computation will never reach an accepting state. However, if \( w \) is odd, then one branch of computation will guess correctly, meaning that the whole machine will accept since we only need one branch to accept.

**PDA:**

1.2 CFG

**Intuition:** Variable \( A \) will represent the base case odd string of either a single 0 or 1. Then, \( S \) can be constructed by adding 0’s or 1’s in pairs to this odd string. This will preserve the oddness or evenness of the resulting string.

**Grammar:**

\[
S \to 0S1 \mid 1S0 \mid 0S0 \mid 1S1 \mid A \\
A \to 1 \mid 0
\]
2 \quad L = \{w \in \{0, 1\}^* \mid w \text{ contains at least three 1’s}\}

2.1 PDA

Intuition: We will start by pushing three 1’s onto the stack. When we get a 0, we will do nothing. When we get a 1, we can either remove one of the 1’s on the stack, or do nothing. If the stack is empty, then we accept since this means we had to have at least three 1’s in the input in order to remove the three 1’s on the stack.

PDA:

2.2 CFG

Intuition: Variable A will represent any string. The set of strings with at least three 1’s can be represented by three 1’s with anything before, after, and between them. Note that this grammar is an ambiguous grammar for the language in that there may be multiple valid parse trees for a string in L.

Grammar:

\begin{align*}
S & \rightarrow A1A1A1A \\
A & \rightarrow 0A \mid 1A \mid \epsilon
\end{align*}
$L = \{ w \in \{0, 1\}^* \mid w \text{ has at least one 1 in the 2nd half} \}$

### 3.1 PDA

**Intuition:** We will start by pushing x’s to the stack on each input character. Then, we will nondeterministically guess when we’ve reached the middle of the string and transition to the next phase. From here, we will pop x’s off the stack for each character. We will also keep track of whether we’ve seen a 1 during this phase by changing state the first time we see a 1. If the stack is empty at the end of our input, then this means that we correctly guessed where the middle of the string was. Then, if we did see a 1 in the second phase, we saw it in the second half of the string so we should accept. In the case of an odd string, when we move to stage 2, we will subtract one x from the count of the first half of the characters so that the stack can be empty at the end of the input.

**PDA:**

![Diagram of PDA](image)

### 3.2 CFG

**Intuition:** Consider what would happen if we repeatedly peeled off a character from the start of the string and from the end of the string. If during one of these peelings, we peel off a 1 from the end of the string, then we know this 1 is in the second half of the string, so the string is in $L$. Then, whatever’s left of the string can be any string whatsoever. In our grammar, $A$ represents any string, while $S$ represents the idea of peeling off characters until we find a 1 in the second half.

**Grammar:**

\[
\begin{align*}
S & \rightarrow 0S0 \mid 1S0 \mid 0A1 \mid 1A1 \\
A & \rightarrow 0A \mid 1A \mid \epsilon
\end{align*}
\]