# Automata Theory (CSE4085-01)

## Homework # 02 – Solution

**2.4** **b.** $S \to 0R0 \mid 1R1 \mid \varepsilon$
$R \to 0R \mid 1R \mid \varepsilon$

**c.** $S \to 0 \mid 1 \mid 00S \mid 01S \mid 10S \mid 11S$

**e.** $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

**f.** $S \to S$

**2.9** A CFG $G$ that generates $A$ is given as follows:

$G = (V, \Sigma, R, S)$, $V = \{S, E_{\mathtt{ab}}, E_{\mathtt{bc}}, C, A\}$, and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. The rules are:

$$\begin{aligned} S &\to E_{\mathtt{ab}}C \mid AE_{\mathtt{bc}} \\ E_{\mathtt{ab}} &\to \mathtt{a}E_{\mathtt{ab}}\mathtt{b} \mid \varepsilon \\ E_{\mathtt{bc}} &\to \mathtt{b}E_{\mathtt{bc}}\mathtt{c} \mid \varepsilon \\ C &\to C\mathtt{c} \mid \varepsilon \\ A &\to A\mathtt{a} \mid \varepsilon \end{aligned}$$

Initially substituting $E_{\mathtt{ab}}C$ for $S$ generates any string with an equal number of a's and b's followed by any number of c's. Initially substituting $E_{\mathtt{bc}}$ for $S$ generates any string with an equal number of b's and c's prepended by any number of a's.

The grammar is ambiguous. Consider the string $\varepsilon$. On the one hand, it can be derived by choosing $E_{\mathtt{ab}}$C with each of $E_{\mathtt{ab}}$ and $C$ yielding $\varepsilon$. On the other hand, $\varepsilon$ can be derived by choosing $AE_{\mathtt{bc}}$ with each of $A$ and $E_{\mathtt{bc}}$ yielding $\varepsilon$. In general, any string $\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k$ with $i = j = k$ can be derived ambiguously in this grammar.

**2.18** $S$ can generate a string of $T$s. Each $T$ can generate strings in $\{\mathtt{a}^m b^m \mid m \geq 1\}$. Here are two different leftmost derivations of ababab.

$S \Rightarrow SS \Rightarrow SSS \Rightarrow TSS \Rightarrow \mathtt{ab}SS \Rightarrow \mathtt{ab}TS \Rightarrow \mathtt{abab}S \Rightarrow \mathtt{abab}T \Rightarrow \mathtt{ababab}$.

$S \Rightarrow SS \Rightarrow TS \Rightarrow \mathtt{ab}S \Rightarrow \mathtt{ab}SS \Rightarrow \mathtt{ab}TS \Rightarrow \mathtt{abab}S \Rightarrow \mathtt{abab}T \Rightarrow \mathtt{ababab}$.

The ambiguity arises because $S$ can generate a string of $T$s in multiple ways. We can prevent this behavior by forcing $S$ to generate each string of $T$s in a single way by changing the first rule to be $S \to TS$ instead of $S \to SS$. In the modified grammar, a leftmost derivation will repeatedly expand $T$ until it is eliminated before expanding $S$, and no other option for expanding variables is possible, so only one leftmost derivation is possible for each generated string.

**2.43** Assume $B$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = \texttt{0}^p\texttt{1}^{2p}\texttt{0}^p$. Because $s \in B$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. We consider several cases.

i) If both $v$ and $y$ contain only 0's (or only 1's), then $uv^2xy^2z$ has unequal numbers of 0s and 1s and hence won't be in $B$.

ii) If $v$ contains only 0s and $y$ contains only 1s, or vice versa, then $uv^2xy^2z$ isn't a palindrome and hence won't be in $B$.

iii) If both $v$ and $y$ contain both 0s and 1s, condition 3 is violated so this case cannot occur.

iv) If one of $v$ and $y$ contain both 0s and 1s, then $uv^2xy^2z$ isn't a palindrome and hence won't be in $B$.

Hence $s$ cannot be pumped and contradiction is reached. Therefore $B$ isn't context-free.

**2.44** Assume $C$ is context-free and get its pumping length $p$ from the pumping lemma. Let $s = \texttt{1}^p\texttt{3}^p\texttt{2}^p\texttt{4}^p$. Because $s \in C$, it can be split $s = uvxyz$ satisfying the conditions of the lemma. By condition 3, $vxy$ cannot contain both 1s and 2s, and cannot contain both 3s and 4s. Hence $uv^2xy^2z$ doesn't have equal number of 1s and 2s or of 3s and 4s, and therefore won't be a member of $C$, so $s$ cannot be pumped and contradiction is reached. Therefore $C$ isn't context-free.

**2.56** Let $M_A$ be a DFA that recognizes $A$, and $M_B$ be a DFA that recognizes $B$. We construct a PDA recognizing $A \diamond B$. This PDA simulates $M_A$ on the first part of the string pushing every symbol it reads on the stack until it guesses that it has reached the middle of the input. After that it simulates $M_B$ on the remaining part of the string popping the stack for every symbol it reads. If the stack is empty at the end of the input and both $M_A$ and $M_B$ accepted, the PDA accepts. If something goes wrong, for example, popping when the stack is empty, or getting to the end of the input prematurely, the PDA rejects on that branch of the computation.