CPSC 313 — Fall 2018
Assignment 2 — Context-Free Languages and Grammars

Jasmine Roebuck, 30037334, October 31, 2018

1. **Non-regular languages and the Pumping Lemma**

   Let $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, =\}$ and consider the language L of all strings over Σ that constitute a valid equation of the form $a + b = c$ where a, b, c are non-negative integers represented in base 10, without leading zeros. Some elements of L include $13 + 17 = 30$ and $99 + 0 = 99$, but not $13 + 17 = 29$ or $99 + 01 = 100$. Use the Pumping Lemma to prove that L is not regular.

   Solution.

   By way of contradiction, assume that L is regular. Then L satisfies the Pumping Lemma. Let p be the pumping length of L, and consider the string $s =$'$1^p + 1^p = 2^p$' $\in L$. Since $|s| = 3p + 2 > p$, s can be written as $s = xyz$ with strings x, y, z such that $y \neq \varepsilon$, $|xy| \leq p$, and $xy^iz \in L$ for all $i \geq 0$.

   Since $|xy| \leq p$ and s begins with p 1s, the substring xy must begin with 1s only. Then y consists of one or more 1s, since $y \neq \varepsilon$.

   Consider the string $w = xyyz = xy^2z$. By the Pumping Lemma, $w \in L$. But since xy consists of 1s only, and y consists of at least one 1, w can be written as '$1^{p+a} + 1^p = 2^p$', where $a = |y| \geq 1$. Since $a \geq 1$, $p + a \neq p$, and therefore '$1^{p+a} + 1^p = 2^p$' is no longer a valid equation, since $1^{p+a} + 1^p = 1^a 2^p \neq 2^p$. So $w \notin L$. This is a contradiction, therefore L cannot be a regular language.

2. **Regular languages are context-free**

   Formally prove that every regular language is context-free. Use the following ingredients in your proof:

   - the recursive definition of regular expressions;
   - the fact that L is a regular language if and only if $L = L(e)$ for some regular expression e;
   - induction on the length of a regular expression — recall that every regular expression is a string of length at least 1 consisting of elements in Σ as well as the symbols $\cup, *, (,), \varepsilon, \emptyset$;
   - the fact that context-free languages are closed under the regular operations; that is, if L_1 and L_2 are context-free languages, then $L_1 \cup L_2$, L_1L_2 and L_1^* are context-free. (You may use this result without proof.)

Solution.

To prove this, we must show that if a language L is regular, then L is also context-free, i.e. that is L can be described by a context-free grammar.

Assume that L is regular. L is a regular language if and only if $L = L(e)$ for some regular expression e, so L can therefore be represented with some regular expression e.

e is a regular expression if e is one of the following:

(a) $\emptyset$,

(b) ε,

(c) a for some $a \in \Sigma$,

(d) $e_1 \cup e_2$, where e_1 and e_2 are regular expressions,

(e) $e_1 e_2$, where e_1 and e_2 are regular expressions,

(f) e_1^*, where e_1 is a regular expression.

I will show that any language of a regular expression e can be generated by a context-free grammar by induction on the length of e. First I will show grammars for the first three cases (the base cases) of e.

If $e = \emptyset$, then the language is empty. $L(e)$ is generated by the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S\}$, $\Sigma = \{\}$, the start variable is S, and R consists of the rule $S \to S$.

If $e = \varepsilon$, $L(e)$ is generated by the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S\}$, $\Sigma = \{\varepsilon\}$, the start variable is S, and R consists of the rule $S \to \varepsilon$.

If $e = a$ for some $a \in \Sigma$, e is generated by the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S\}$, $\Sigma = \{a\}$, the start variable is S, and R consists of the rule $S \to a$.

In all three of these base cases, $L(e)$ is generated by a context-free grammar, and so $L(e)$ is a context-free language. If e is not one of the above base cases, then it must be one of d, e, or f from the above list. Assume e_1 and e_2 are regular expressions where $L(e_1)$ and $L(e_2)$ can be generated by some context-free grammar, i.e. $L(e_1)$ and $L(e_2)$ are context-free.

If $e = e_1 \cup e_2$, let S_1 be the start variable of the grammar for $L(e_1)$, and let S_2 be the start variable of the grammar for $L(e_2)$. Then $L(e) = L(e_1) \cup L(e_2)$, which can be generated by the grammar $S \to S_1 \mid S_2$. Since context-free languages are closed under union, and $L(e_1)$ and $L(e_2)$ are context-free, $L(e)$ is also context free.

Similarly, if $e = e_1 e_2$, let S_1 be the start variable of the grammar for $L(e_1)$, and let S_2 be the start variable of the grammar for $L(e_2)$. Then $L(e) = L(e_1 e_2)$, which can be generated by the grammar $S \to S_1 S_2$. Since context-free languages are closed under concatenation, and $L(e_1)$ and $L(e_2)$ are context-free, $L(e)$ is also context free.

If $e = e_1^*$, let S_1 be the start variable of the grammar for $L(e_1)$. Then $L(e)$ can be generated by the grammar $S \to SS_1 \mid \varepsilon$. Since context-free languages are closed under the Kleene closure, and $L(e_1)$ is context-free, $L(e)$ is also context free.

Therefore, since L is regular and is represented by some regular expression e, a context-free grammar can always be constructed for $L(e)$. Hence L is also context-free.

3. **Designing context-free grammars and languages**

   (a) Design a context-free grammar for the language

$$L = \left\{a^{2i}b^j vc^j (ac)^i \mid i, j \geq 0, v \in \{a, b\}^*\right\}$$

over the alphabet $\Sigma = \{a, b, c, \}$. Your grammar must have at most 3 variables and at most 7 rules. Clearly state the variables, the terminals, the rules, and the start variable for your grammar. You need *not* formally prove your grammar correct, but you should give a brief, coherent, convincing explanation of its correctness (in case of errors, such an explanation may also secure you partial credit).

Solution.

L can be described by the below grammar $G = (V, \Sigma, R, S)$, where $V = \{S, A, B\}$, $\Sigma = \{a, b, c\}$, the start variable S is S, and R consists of the rules

$$\begin{aligned} S &\to aaSac \mid A \\ A &\to bAc \mid B \\ B &\to aB \mid bB \mid \varepsilon \end{aligned}$$

Any string $w \in L$ begins with $2i$ a's and ends with i (ac)'s, where $i \geq 0$. We can also write that w must begin with i (aa)'s and end with i (ac)'s. The number of occurrences of 'aa' at the start of w must match with the number of occurrences of 'ac' at the end of w, so we start with a rule $S \to aaSac$ which will guarantee this. We also include a rule $S \to A$ for when we are finished with the first rule, and to account for the cases where $i = 0$.

The inner substring of w is handled similarly. Let u be the substring $b^j vc^j$, where $j \geq 0$, and $v \in \{a, b\}^*$. We start producing this substring when we take the rule $S \to A$. The number of b's at the start of u must be equal to the number of c's at the end, so we add the rule $A \to bAc$ to guarantee this. Then we add the rule $A \to B$ for when we are finished with the previous rule, and to account for the cases where $j = 0$.

The only substring left to generate is v in the middle of w. The only restriction on v is that $v \in \{a, b\}^*$, which is to say that v is any substring made of symbols from the alphabet. We add the rules $B \to aB$ and $B \to bB$ to allow us to generate any substring of these symbols. Then we add the rule $B \to \varepsilon$, both to allow the string derivation to terminate and to allow for the cases where $v = \varepsilon$.

(b) Consider the context-free grammar $G = (V, \Sigma, R, S)$ where $V = \{S, A, B, C\}$, $\Sigma = \{a, b, c\}$ and R consists of the rules

$$
\begin{aligned}
S &\to ASA \mid B \\
A &\to a \mid b \\
B &\to BC \mid \varepsilon \\
C &\to a \mid b \mid c
\end{aligned}
$$

Describe $L(G)$. You need not formally prove your answer correct, but you should again give a brief, coherent, convincing explanation of how you obtained you answer (in case of errors, such an explanation may also secure you partial credit).

Solution.

$$L(G) = \{uvw, \mid u, w \in \{a, b\}^*, |u| = |w|, \text{ and } v \in \{a, b, c\}^*\}$$

The rule $S \to B$ can only be taken once as there is no way to return to S after taking this rule. Before this, the rule $S \to ASA$ can be taken any number of times, including zero. Taking this rule some n number of times followed by taking $S \to B$ once produces the string of variables $x = A^n BA^n$.

The only rules for A are $A \to a$ and $A \to b$, so A can only produce exactly one terminal. Therefore the length of the string is unchanged, and the string becomes uBw, where $u, w \in \{a, b\}^*$, and $|u| = |w| = n$.

The rule $B \to \varepsilon$ can only be taken once as it produces no more variables. So the rule $B \to BC$ can be taken some m number of times, including zero, followed by taking the rule $B \to \varepsilon$ once. Each iteration of the first rule adds one C to the string and leaves the number of Bs unchanged. Taking the second rule eliminates the B. Thus the string x becomes $uC^m w$.

Similar to A, the variable C can only produce exactly one terminal, so C^m in x must produce some m number of terminals from the alphabet. Therefore the string x becomes uvw, where $v \in \{a, b, c\}^*$, and the previous conditions on u and w still hold.