Formal Languages Assignment 4

Rachel Hwang

October 31, 2013

- **Exercise 0.1**
  Design a context-free language for the set $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$ *that is, the set of strings of a's followed by b's followed by c's, such that there are either a different number of a's and b's or a different number of b's and c's, or both.*

  $G = \{\{S, A, B, C, D, E\},\ \{a, b, c, \epsilon\},\ P,\ S\}$, where P is as follows.

$$\begin{aligned} S &\to DC \mid AE \\ A &\to aA \mid \epsilon \\ B &\to Bb \mid b \\ C &\to Cc \mid \epsilon \\ D &\to aDb \mid aA \mid B \\ E &\to bEc \mid B \mid Cc \end{aligned}$$

  We handle each case – $i \neq j$ *or* $j \neq k$ – seperately. First, A, B and C each produce any number of a's, b's, and c's respectively (although at least one b, to avoid ever generating ϵ, which has 0 of each letter). The productions rules on D create a string with an unequal number of a's and b's, by first producing an equal number of a's and b's, then using A or B to add any number of only a's or only b's. DC thus creates an an unequal number of a's and b's concatenated with any number of c's. Similar to D, the productions rules on E create a string with an unequal number of b's and c's. AE thus creates any number of a's concatenated with an an unequal number of b's and c's. Note that all the mimimal elements a, b, c, ab, ac, bc are also covered in this grammar. Thus, this grammar generates precisely those strings in the given set.

- **Exercise 0.2**
  Design a context-free language for the set of all strings of a's and b's that are not of the form ww, that is, not equal to any string repeated.

  $G = \{\{S, A, B\},\ \{a, b\},\ P,\ S\}$, where P is as follows.

$$\begin{aligned} S &\to A \mid B \mid AB \mid BA \\ A &\to aAa \mid aAb \mid bAa \mid bAb \mid a \\ B &\to aBa \mid aBb \mid bBa \mid bBb \mid b \end{aligned} \tag{1}$$

The production rules for A produce all odd-length strings with a as the middle character and those of B produce all odd-length strings with b as the middle character. Between them, they produce all odd-length strings. Since we know that a string of odd length cannot be of the repeated form ww, all productions of A and B are valid.

However, for a string of some length $2k$, we know that there must be some pair of characters in the string c_i and c_j where index of c_i is $1 \leq i \leq k$, the index of c_j is $i + k$, and $c_i \neq c_j$. In other words, for some pair of characters in the same position in first and second string of k characters, these characters must be different. As long as this condition is met, the rest of the string can be arbitrary a's and b's. Given $c_i \neq c_j$, a valid string in this language is therefore $i - 1$ arbitrary symbols followed by c_i followed by $k - 1$ arbitrary symbols followed by c_j followed by $2k - j = k - i$ symbols. This means that a string that is that are not of the form ww has the format $(i{-}1\ chars)(c_i)(k{-}1\ chars)(c_j)(k{-}i\ chars)$, where $c_i \neq c_j$.

Since we know that c_i and c_j are in the same positions in their respective halves of the string, we know $k - 1 = (i - 1) + (k - i)$. An equivalent way to state the form of the string is therefore $(i - 1\ chars)(c_i)(i - 1\ chars)(k - i\ chars)(c_j)(k - i\ chars)$, where $c_i \neq c_j$ We can thus use the production rules for A and B to generate strings of this form by taking two different characters to use as c_i and c_j and 'growing' equal numbers of arbitrary characters of either side of each. This produces all even-length strings in our language. Thus, because this grammar generates all odd-length strings and only valid even-length strings, it generates precisely the strings in the given set.

- **Exercise 0.3**
  Show that every regular language is a context-free language. Hint: Construct a CFG by induction on the number of operators in the regular expression.

  Each regular language is described by some regular expression. Therefore, if we can show that for all regular expressions, there is an equivalent CFG, we know that each regular languages is described by some CFG, and is therefore a context-free language.

  The set of all regular expressions R over some alphabet Σ is inductively defined. As our basis, we have the empty set $\varnothing \in R$, the empty string $\epsilon \in R$, and $\forall a \in \Sigma$, $a \in R$. Next we have several operators: if $A, B \in R$, then $AB \in R$ (concatentation), $A + B \in R$ (alternation, the union between the describes sets), and $A^* \in R$ (Kleene star, the set of all strings made by concatenating a finite number of A). All regular expressions are produced using these constants and operations. Now we show that for all the sets described by a regular expression may be described by a CFG.

  First, the constants, the sets described by the minimal regular expressions, are simple to generate using a CFG.

  $$S \to \epsilon$$
  $$S \to a,\ where\ a \in \Sigma$$

  since this generates the empty string and all strings consisting of a single element in Σ. So for any minimal regular expression, there is an equivalent CFG. Now that we have shown the base

case, for every operator over a regular expression, we must find an equivalent modifcation of an equivalent CFA.

To represent concatenation over regular expressions A and B, given two CFG's which describe the same languages, $C_A = \{V_A, T_A, P_A, S_A\}$ and $C_B = \{V_B, T_B, P_B, S_B\}$, our new CFG C is $\{V_A \cup V_B \cup S,\ T_A \cup T_B,\ P_A \cup P_B \cup S,\ S\}$, where S is

$$S \to S_A S_B$$

since this means that we now have a CFG which only accepts strings which are concatenations of one string accepted by C_A with one from C_B, which is AB. In other words, the components of our new CFG C consist of unions of the C_A and C_B components, except the new S defined above.

Similarly, the alternation operator over C_A and C_B simply takes adds a new production rule S such that $C = \{V_A \cup V_B \cup S,\ T_A \cup T_B,\ P_A \cup P_B \cup S,\ S\}$, where S is

$$S \to S_A \mid S_B$$

since this generates both all strings from expression C_A and all those of C_B, which is $A \cup B$.

Finally, the Kleene star operator on C_A yields $C = \{V_A \cup S,\ T_A \cup \epsilon,\ P_A \cup S,\ S\}$, where S is

$$S \to S_A S \mid \epsilon$$

since this generates a string of C_A concatenated with itself zero or more times, which is A^*.

As stated above, all regular expressions are inductively defined using some minimal cases and three operators. We have shown ways to create equivalent CFG's for all the minimal cases and for each of those regular expression operators, found equivalent modifications for a CFG. Therefore, for any regular expression we can create, we can create an equivalent CFG.

As an example, let's find the CFG equivalent to the regular expression $a(a + b)^*$. We first find the CFG's for the minimal regular expressions a and b, which are simply $C_a = \{\{A\},\ \{a\},\ P_a, \{A\}\}$ and $C_b = \{\{B\},\ \{b\},\ P_b, \{B\}\}$ where

$$P_a = \{A \to a\}$$
$$P_b = \{B \to b\}$$

Now to find the CFG for $(a + b)$, let's call it C_1 we apply the modification defined above for alternation on C_a and C_b to get $C_1 = \{\{A, B, S_1\},\ \{a, b\},\ P_1, \{S_1\}\}$ where

$$P_1 = \{S_1 \to A \mid B$$
$$A \to a$$
$$B \to b\}$$

To find the CFG for $(a+b)^*$, let's call it C_2, we apply the modification defined above for the kleene star on C_1 to get $C_2 = \{\{A, B, S_1, S_2\},\ \{a, b, \epsilon\},\ P_2, \{S_2\}\}$ where

$$\begin{aligned} P_2 = \{&S_2 \to S_1 S_2 \mid \epsilon \\ &S_1 \to A \mid B \\ &A \to a \\ &B \to b\} \end{aligned}$$

Finally, to find the CFG for $a(a+b)^*$, let's call it C_3, we apply the modification defined above for concatenation to C_a and C_2 to get $C_3 = \{\{A, B, S_1, S_2, S_3\},\ \{a, b, \epsilon\},\ P_3, \{S_3\}\}$ where

$$\begin{aligned} P_3 = \{&S_3 \to A S_2 \\ &S_2 \to S_1 S_2 \mid \epsilon \\ &S_1 \to A \mid B \\ &A \to a \\ &B \to b\} \end{aligned}$$

- **Exercise 0.4**
  A CFG is said to be right-linear if each production body has at most one variable, and that variable is at the right end. That is, all productions of a right-linear grammar are of the form $A \to wB$ or $A \to w$, where A and B are variables and w some string of zero or more terminals.

  1. *Show that every right-linear grammar generates a regular language. Hint: Construct an ϵ-NFA that simulates leftmost derivations, using its state to represesnt the lone variable in the current left-sentential form.*

     Given a right-linear grammar G, to show that $L(G)$ is regular, we want to construct some NFA $M = \{Q,\ \Sigma,\ \delta,\ q_0,\ F\}$ which describes it. Let $G = \{V,\ T,\ P,\ S\}$. Assume that $V = \{V_0, \ldots V_n\}$ and $S = V_0$. First, Σ of M will be equal to T. This is easy to see because $L(M)$ must use precisely the same characters as $L(G)$. M's states Q will consist of $V_0, \ldots V_n$, with some additional auxiliary states as described below. Our start state $q_0 = V_0$. F will consist of a single final state $f \notin V$. The transition function δ must be based on the productions of G.

     Because G is a right-linear grammar, all productions will be of the form (1) $V_i \to a_1 \ldots a_k V_j$, some k number of terminals followed by a lone variable, or else (2) $V_i \to a_1 \ldots a_k$, just some k number of terminals. In case (1), our transition function give be $\hat{\delta}\{V_i, a_1 \ldots a_k\} = V_j$. For each of the $k > 1$ characters we will add an auxiliary state to Q, where $\delta\{V_i, a_1\} = Aux_1$ and $\delta\{Aux_{k-1}, a_k\} = V_j$. In case (2), our transition function will give $\hat{\delta}\{V_i, a_1 \ldots a_k\} = f$.

     Given that definition of M, we can observe that $V_0 \xrightarrow{*} w$ if and only if $\hat{\delta}\{V_i, w\} = f$. This means that $V_0 \xrightarrow{*} w$ if and only if w is accepted by M. We have constructed an NFA which exactly describes the $L(G)$, therefore G must be regular.

2. *Show that every regular language has a right-linear grammar. Hint: Start with a DFA and let the variables of the grammar represent states.*

   This problem is very similar to Exercise 0.4.1, just in reverse. Because every regular language can be described by some DFA, we want to show that for any DFA M, we can construct a right-linear grammar G for $L(M)$. Let $M = \{Q,\ \Sigma,\ \delta,\ q_0,\ F\}$ and $G = \{V,\ T,\ P,\ S\}$. First, the terminals $T = \Sigma$, since in order to describe the same language, G and M must use the same set of characters. Let'a assume that $Q = \{q_0, \ldots q_n\}$. Our set of variables $V = \{q_0, \ldots q_n\}$ and our start variable $S = q_0$.

   We can define P, describing δ, by creating a production rule for every $\delta\{q_i, a_i\} = q_j$. For every such δ expression, the production rule will be $q_i \to a_i q_j$, a single character a followed by a lone variable q_j. If q_j is a final state, the production rule will be $q_i \to a_i$, simply a single character a_i.

   Given this definition of G, we can observe that w is accepted by M if and only if $q_0 \overset{*}{\to} w$. Since we have successfully constructed a G for M, this shows that for any regular language $L(M)$, there is some right-linear grammar.