8. CONTEXT-FREE AND NON-CONTEXT-FREE LANGUAGES

8.1 The Pumping Lemma for Context-Free Languages

- Languages can be shown to be non-context-free by using a pumping lemma that is similar to the one for regular languages.

- Suppose that a derivation in a context-free grammar G involves using a variable A more than once, in the following way:

  $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$

  where $v, w, x, y, z \in \Sigma^*$. Then

  $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vw^2Ay^2z \Rightarrow^* vw^3Ay^3z \Rightarrow^* \ldots$

  Since x can be derived from each of these A's, it must be the case that all the strings vxz, $vwxyz$, vw^2xy^2z, … are in $L(G)$.

- The pumping lemma will follow from the fact that this duplication of variables occurs in the derivation of every sufficiently long string in $L(G)$.

The Pumping Lemma for Context-Free Languages (Continued)

- The discussion will be simpler if it can be assumed that derivations are represented by *binary* trees, where no node has more than two children. To guarantee this property, the grammar can be converted into Chomsky normal form.

- Definitions:

  A *path* in a nonempty binary tree consists either of a single node or of a node, one of its descendants, and all the nodes in between.
  The *length* of a path is the number of nodes it contains.
  The *height* of a binary tree is the length of the longest path.

- **Lemma 8.1.** For any $h \geq 1$, a binary tree having more than 2^{h-1} leaf nodes must have height greater than h.

  Proof: The contrapositive statement (if the height is no more than h, the number of leaf nodes is no greater than 2^{h-1}) can be proved by induction on h.

  For the basis step, note that a binary tree with height ≤ 1 has no more than one node and therefore no more than one leaf node.

  In the induction step, suppose that $k \geq 1$ and that any binary tree of height $\leq k$ has no more than 2^{k-1} leaf nodes.

  Now let T be a binary tree with height $\leq k + 1$. If T has no more than one node, the result is clear. Otherwise, the left and right subtrees of T both have height $\leq k$, and so by the induction hypothesis each has 2^{k-1} or fewer leaf nodes.

  The number of leaf nodes in T is the sum of the numbers in the two subtrees and is therefore no greater than $2^{k-1} + 2^{k-1} = 2^k$.

The Pumping Lemma for Context-Free Languages (Continued)

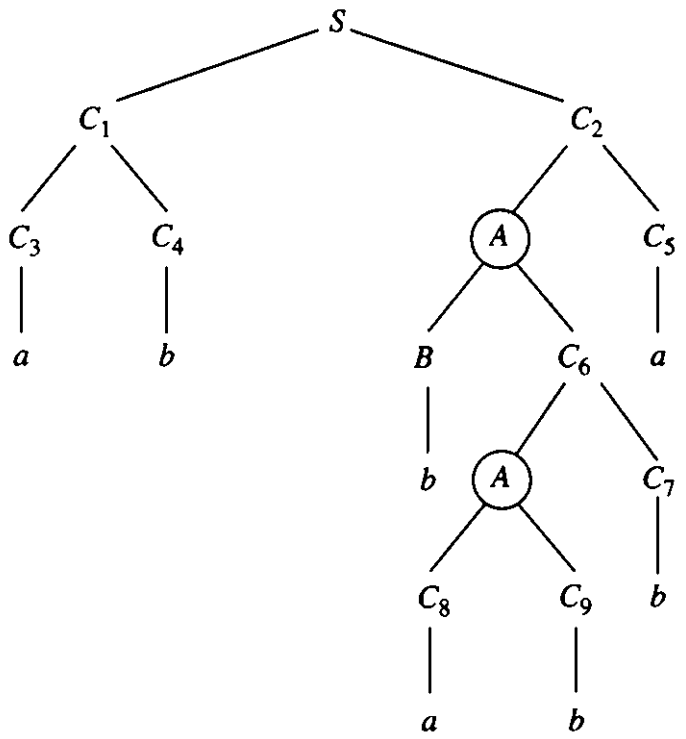
- **Theorem 8.1.** Let $G = (V, \Sigma, S, P)$ be a context-free grammar in Chomsky normal form, with a total of p variables. Any string u in $L(G)$ with $|u| \geq 2^{p+1}$ can be written as $u = vwxyz$, for some strings v, w, x, y, and z satisfying

  $|wy| > 0$
  $|wxy| \leq 2^{p+1}$
  for any $m \geq 0$, $vw^m xy^m z \in L(G)$

  Proof: Lemma 8.1 shows that any derivation tree for u must have height at least $p + 2$. (It has more than 2^p leaf nodes, and therefore its height is $> p + 1$.)

  Consider a path of maximum length and look at the bottom portion, consisting of a leaf node and the $p + 1$ nodes above it. Each of these $p + 1$ nodes corresponds to a variable, so some variable A must appear twice in this portion of the path.

  Let x be the portion of u derived from the A closest to the leaf, and let $t = wxy$ be the portion of u derived from the other A. If v and z represent the beginning and ending portions of u, then $u = vwxyz$, as in the following figure:



$u = (ab)(b)(ab)(b)(a)$
$= v \ w \ x \ y \ z$

The Pumping Lemma for Context-Free Languages (Continued)

The A closest to the root in this portion of the path is the root of a binary derivation tree for wxy. Since the path has maximum length, this tree has height $< p + 2$, and so by the lemma $|wxy| \leq 2^{p+1}$.

The node containing this A has two children, both corresponding to variables. Let B denote the one that is not an ancestor of the other A. Since x is derived from that other A, the string of terminals derived from B does not overlap x. It follows that either w or y is nonnull, and therefore $|wy| > 0$.

The derivation

$S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$

follows if the first A is the one in the higher node and the second the one in the lower.

- The pumping lemma becomes easier to use if it is stated in terms of a language rather than a grammar.

- **Theorem 8.1a (The Pumping Lemma for Context-Free Languages).** Let L be a CFL. Then there is an integer n so that for any $u \in L$ satisfying $|u| \geq n$, there are strings $v, w, x, y,$ and z satisfying

  $u = vwxyz$ (8.1)
  $|wy| > 0$ (8.2)
  $|wxy| \leq n$ (8.3)
  For any $m \geq 0$, $vw^m xy^m z \in L$ (8.4)

  Proof: First, find a context-free grammar in Chomsky normal form that generates $L - \{\Lambda\}$. If p is the number of variables in this grammar and $n = 2^{p+1}$, the result is an immediate consequence of Theorem 8.1.

- The pumping lemma for context-free languages is used in the same way as the pumping lemma for regular languages. A language L is assumed to be context-free, and then a string u is chosen. The theorem says that there exist strings $v, w, x, y,$ and z satisfying the four properties; the goal is then to show that *every* choice of v, w, x, y, z satisfying the properties leads to a contradiction.

Applying the Pumping Lemma

- Intuitively, it would seem that a PDA could not accept strings of the form $a^i b^i c^i$, because matching the *a*'s and *b*'s would require emptying the stack. The pumping lemma can be used to prove that the language of such strings is not a CFL.

- **Example 8.1:** The Pumping Lemma Applied to $\{a^i b^i c^i\}$

  Let

  $L = \{a^i b^i c^i \mid i \geq 1\}$

  Suppose for the sake of contradiction that L is context-free, and let n be the integer in Theorem 8.1a. Let $u = a^n b^n c^n$.

  Suppose v, w, x, y, and z are any strings satisfying conditions (8.1)–(8.4). Since $|wxy| \leq n$, the string wxy can contain at most two distinct types of symbol (*a*'s, *b*'s, and *c*'s), and since $|wy| > 0$, w and y together contain at least one.

  The string vw^2xy^2z contains additional occurrences of the symbols in w and y; therefore, it cannot contain equal numbers of all three symbols. However, according to (8.4), $vw^2xy^2z \in L$. This is a contradiction, so L cannot be a CFL.

  The same proof can be used to show that the following language is not a CFL:

  $L_1 = \{u \in \{a, b, c\}^* \mid n_a(u) = n_b(u) = n_c(u)\}$

Applying the Pumping Lemma (Continued)

- **Example 8.2:** The Pumping Lemma Applied to $\{ss \mid s \in \{a, b\}^*\}$

  Let

  $L = \{ss \mid s \in \{a, b\}^*\}$

  Suppose L is a context-free language, and let n be the integer in Theorem 8.1a. Choose $u = a^n b^n a^n b^n$. Suppose that v, w, x, y, and z are any strings satisfying (8.1)–(8.4).

  Because of condition (8.2), wxy can overlap at most two of the four contiguous groups of symbols.

  Case 1: w or y contains at least one a from the first group of a's. Since $|wxy| \le n$, neither w nor y can contain any symbols from the second half of u. Then

  $vw^0xy^0z = a^i b^j a^n b^n$

  where $i < n$ and $1 \le j \le n$. The midpoint of this string is somewhere within the substring a^n, and therefore it is impossible for it to be of the form ss.

  Case 2: wxy contains no a's from the first group but w or y contains at least one b from the first group of b's. Then

  $vw^0xy^0z = a^n b^i a^j b^n$

  where $i < n$ and $1 \le j \le n$. The midpoint is somewhere in the substring $b^i a^j$, and as before, the string cannot be in L.

  Case 3: wxy contains no symbols from the first half of u but at least one a. Similar to case 2.

  Case 4: x or y contains at least one b from the second group of b's. Similar to case 1.

  The same argument works for the languages $\{a^i b^i a^i b^i \mid i \ge 0\}$ and $\{a^i b^j a^i b^j \mid i, j \ge 0\}$. A similar proof shows that $\{scs \mid s \in \{a, b\}^*\}$ also fails to be context-free.

Applying the Pumping Lemma (Continued)

- Potential trouble-spots in using the pumping lemma:

  Choosing the string u. A bad choice may not lead to a contradiction.
  Deciding on the cases to consider. Some effort may be necessary to avoid having a large number of cases. Also, it is important to ensure that every case leads to a contradiction.
  Choosing the value of m to use in order to obtain a contradiction. Sometimes choosing $m = 0$ will not work but choosing $m > 1$ will, and vice-versa.

- **Example 8.3:** A Third Application of the Pumping Lemma

  Let

  $L = \{x \in \{a, b, c\}^* \mid n_a(x) < n_b(x) \text{ and } n_a(x) < n_c(x)\}$

  Suppose L is a CFL, and let n be the integer in Theorem 8.1a. Let $u = a^n b^{n+1} c^{n+1}$. If (8.1)–(8.4) hold for strings v, w, x, y, and z, the string wxy can contain at most two distinct symbols.

  Case 1: w or y contains at least one a. Then wy cannot contain any c's, so vw^2xy^2z contains at least as many a's as c's and cannot be in L.

  Case 2: Neither w nor y contains an a. Then vw^0xy^0z still contains n a's. Since wy contains one of the other two symbols, vw^0xy^0z contains fewer occurrences of that symbol than u does and therefore is not in L.

  There is a contradiction in either case, so L cannot be context-free.

  The same argument can be used to show that $\{a^i b^j c^k \mid i < j \text{ and } i < k\}$ is not context-free.

Applying the Pumping Lemma (Continued)

- **Example 8.4:** The Set of Legal C Programs Is Not a CFL

  Context-free grammars can describe much of the syntax of high-level languages. However, there are some rules in these languages that depend on context.

  Consider the set L of all legal C programs. The pumping lemma can be used to show that L is not a context-free language. The proof uses the fact that, in C, a variable must be declared before it is used.

  Assume that L is a CFL, and let n be the integer in Theorem 8.1a. Let u be the following C program:

  `main(){int aa…a;aa…a;aa…a;}`

  All three identifiers are assumed to be $\mathtt{a}^n$.

  According to the pumping lemma, $u = vwxyz$, where (8.2)–(8.4) are satisfied. In particular, vw^0xy^0z should be a valid C program. However, this is impossible.

  Case 1: wy contains the blank or any of the symbols before it. Then *vxz* still contains at least most of the first occurrence of the identifier, and without `main(){int` and the blank, it cannot be syntactically correct.

  Case 2: wxy is a substring of "$\mathtt{a}^n\mathtt{;}\ \mathtt{a}^n\mathtt{;}\ \mathtt{a}^n\mathtt{;}\}$".

  Case 2a: wy contains either the final semicolon or bracket. Then the string *vxz* is illegal.

  Case 2b: wy contains one of the two intermediate semicolons, and possibly portions of one or both of the identifiers on either side. Then *vxz* has two identifiers, which are now not the same.

  Case 2c: wy contains only a portion of one of the identifiers and nothing else. Then *vxz* still has a variable declaration and two subsequent expressions consisting of an identifier, but the three identifiers are not all the same.

  In each case *vxz* is not a legal C program, and therefore L is not a context-free language.

  The reason that the program contains three occurrences of $\mathtt{a}^n$ is that otherwise *wy* might be just the first semicolon, which does not lead to a contradiction.

Ogden's Lemma

- Although the pumping lemma provides some information about the strings w and y that are pumped, it says little about their location within u.

- A generalization of the pumping lemma known as Ogden's lemma makes it possible to designate certain positions of u as "distinguished" and to guarantee that the pumped portions include at least some of these distinguished positions.

- Ogden's lemma is sometimes more convenient than the pumping lemma, and occasionally it can be used when the pumping lemma fails.

- **Theorem 8.2 (Ogden's Lemma).** Suppose L is a context-free language. Then there is an integer n so that if u is any string in L of length n or greater, and any n or more positions of u are designated as "distinguished," then there are strings v, w, x, y, and z satisfying

  $u = vwxyz$ (8.5)
  the string wy contains at least one distinguished position (8.6)
  the string wxy contains no more than n distinguished positions (8.7)
  the string x contains at least one distinguished position (8.8)
  for every $m \geq 0$, $vw^m xy^m z \in L$ (8.9)

  Proof: As in the proof of the pumping lemma, let $n = 2^{p+1}$, where p is the number of variables in a Chomsky normal form grammar generating $L - \{\Lambda\}$. Suppose u is a string in L in which n or more positions are marked as distinguished.

  The strings v, w, x, y, and z will be obtained as before, by selecting a path in a derivation tree for u on which some variable A occurs twice. However, the path will be chosen in a particular way this time.

  The path starts at the root node. At each interior node N in the path, the path is extended downward by selecting the child of N having the larger number of distinguished positions among its descendants (breaking ties arbitrarily).

  An interior node on this path is said to be a *branch point* if both its children have at least one distinguished descendant. Because of the way the path is chosen, every branch point below the top one has at least half as many distinguished descendants as the branch point above it in the path.

Ogden's Lemma (Continued)

- **Lemma 8.2.** If the binary tree consisting of a node on the path and its descendants has h or fewer branch points, its leaf nodes include no more than 2^h distinguished nodes.

  Proof: The proof is virtually identical to that of Lemma 8.1. The reason the statement involves 2^h rather than 2^{h-1} is that the bottom-most branch point has two distinguished descendants rather than one.

- *Conclusion of proof of Theorem 8.2:* Since there are n distinguished leaf nodes in the tree, and $n > 2^p$, Lemma 8.2 implies that there must be more than p branch points in the chosen path.

  Consider the $p + 1$ branch points farthest down in the path, and the subtree whose root is the topmost such node. Since there are only p variables in the grammar, at least two branch points are labeled with the same variable A.

  The strings v, w, x, y and z are now defined in terms of these two nodes exactly as in the proof of the pumping lemma.

  Property (8.6) follows from the definition of branch point, and (8.7) is an immediate consequence of Lemma 8.2. Property (8.8) is clearly true because the bottom A is a branch point. Properties (8.5) and (8.9) follow as in the proof of the pumping lemma.

- Note that the ordinary pumping lemma is identical to the special case of Theorem 8.2 in which *all* the positions of u are distinguished.

Applying Ogden's Lemma

- **Example 8.5:** Using Ogden's Lemma on $\{a^i b^i c^j \mid j \neq i\}$

  Let $L = \{a^i b^i c^j \mid i, j \geq 0 \text{ and } j \neq i\}$. Suppose L is a context-free language, and let n be the integer in the statement of Theorem 8.2. One choice for the string u is

  $u = a^n b^n c^{n+n!}$

  Let the first n positions of u be the distinguished positions, and suppose that v, w, x, y, and z satisfy (8.5)–(8.9).

  First, if either w or y contains two distinct symbols, then vw^2xy^2z will no longer have the form $a^*b^*c^*$, which would be a contradiction. Second, w or y must consist of a's because wy contains at least one distinguished position.

  Unless w consists of a's and y consists of the same number of b's, the string vw^2xy^2z gives a contradiction, because it has different numbers of a's and b's.

  Suppose that $w = a^j$ and $y = b^j$. Let $k = n!/j$, which is an integer, and let $m = k + 1$. Then the number of a's in vw^mxy^mz is

  $n + (m - 1) * j = n + k * j = n + n!$

  which is the same as the number of c's. This is a contradiction, and therefore L cannot be context-free.

  With just the pumping lemma, there would be no way to rule out the possibility that w and y contained only c's.

Applying Ogden's Lemma (Continued)

- **Example 8.6:** Using Ogden's Lemma when the Pumping Lemma Fails

  Let $L = \{a^p b^q c^r d^s \mid p = 0 \text{ or } q = r = s\}$. L satisfies the properties of the pumping lemma, but can be shown to be non-context-free by using Ogden's lemma.

  Suppose n is any positive integer and u is any string in L with $|u| \geq n$, say $u = a^p b^q c^r d^s$. It is possible to choose strings v, w, x, y, and z that satisfy (8.1)–(8.4).

  Case 1: If $p = 0$, then there are no restrictions on the numbers of b's, c's, or d's, and the choices $w = b$, $v = x = y = \Lambda$ work.

  Case 2: If $p > 0$, then $q = r = s$, and the choices $w = a$, $v = x = y = \Lambda$ work.

  Theorem 8.2 can be used to show that L is not a CFL. Suppose L is a CFL, let n be the integer in the theorem, let $u = ab^n c^n d^n$, and designate all but the first position of u as distinguished.

  Suppose that v, w, x, y, and z satisfy (8.5)–(8.9). Then wy must contain one of the symbols b, c, or d and cannot contain all three. Therefore, vw^2xy^2z has one a and does not have equal numbers of b's, c's, and d's, so it is not in L.

8.2 Intersections and Complements of Context-Free Languages

- According to Theorem 6.1, the set of context-free languages is closed under union, concatenation, and Kleene *. Unlike the regular languages, however, the set of context-free languages is *not* closed under intersection and complement.

- **Theorem 8.3.** There are CFLs L_1 and L_2 so that $L_1 \cap L_2$ is not a CFL, and there is a CFL L so that L' is not a CFL.

  Proof: Example 8.3 showed that

  $L = \{a^i b^j c^k \mid i < j \text{ and } i < k\}$

  is not context-free. However, L is the intersection of the context-free languages

  $L_1 = \{a^i b^j c^k \mid i < j\}$

  and

  $L_2 = \{a^i b^j c^k \mid i < k\}$

  L_1 is generated by the grammar with productions

  $S \to ABC \quad A \to aAb \mid \Lambda \quad B \to bB \mid b \quad C \to cC \mid \Lambda$

  Similarly, L_2 is generated by the CFG with productions

  $S \to AC \quad A \to aAc \mid B \quad B \to bB \mid \Lambda \quad C \to cC \mid c$

  The second statement in the theorem follows from the first and the formula

  $L_1 \cap L_2 = (L_1' \cup L_2')'$

  If complements of CFLs were always CFLs, then for any CFLs L_1 and L_2, the languages L_1' and L_2' would be CFLs, so would their union, and so would its complement.

- **Example 8.7:** A CFL Whose Complement Is Not a CFL

  The second part of the proof of Theorem 8.3 can be used to find an example of a CFL whose complement is not a CFL. Let L_1 and L_2 be the languages defined in the first part of the proof. Then $L_1 \cap L_2 = (L_1' \cup L_2')'$ is not a CFL.

  Because the union of CFLs is a CFL, at least one of the three languages L_1, L_2, $L_1' \cup L_2'$ is a CFL whose complement is not a CFL.

  If a string is not in L_1, then either it is not an element of R, the language $\{a\}^*\{b\}^*\{c\}^*$, or or it has the form $a^i b^j c^k$ for some $i \geq j$. In other words,

  $L_1' = R' \cup \{a^i b^j c^k \mid i \geq j\}$

  The language R' is regular because R is, and therefore R' is context-free. The second language involved in the intersection can be expressed as the concatenation

  $\{a^i b^j c^k \mid i \geq j\} = \{a^m \mid m \geq 0\}\{a^j b^j \mid j \geq 0\}\{c^k \mid k \geq 0\}$

  each factor of which is a CFL. Therefore, L_1' is a CFL. A similar argument shows that L_2' is also a CFL. The conclusion is that $L_1' \cup L_2'$, or

  $R' \cup \{a^i b^j c^k \mid i \geq j \text{ or } i \geq k\}$

  is a CFL whose complement is not a CFL. (In fact, the second part alone is also an example.)

Intersections and Complements of Context-Free Languages (Continued)

- Theorem 3.4 showed how to build an FA M whose states are pairs (p, q) of states in other other FAs M_1 and M_2, respectively. This allows M to keep track of both machines at once. By choosing an appropriate set of accepting states, M can be made to accept the intersection of the languages accepted by M_1 and M_2.

- This construction does not work for PDAs, because the machine M would not able to keep track of the stacks of M_1 and M_2 simultaneously. One machine could be pushing when the other is popping, for example.

- However, the construction can be used if one of the machines is a PDA and the other is an FA.

Intersections and Complements of Context-Free Languages (Continued)

- **Theorem 8.4.** If L_1 is a context-free language and L_2 is a regular language, then $L_1 \cap L_2$ is a context-free language.

  Proof: Let $M_1 = (Q_1, \Sigma, \Gamma_1, q_1, Z_1, A_1, \delta_1)$ be a PDA accepting the language L_1 and let $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be an FA recognizing L_2. Define a PDA $M = (Q, \Sigma, \Gamma, q_0, Z_1, A, \delta)$ as follows:

  $Q = Q_1 \times Q_2 \quad q_0 = (q_1, q_2) \quad A = A_1 \times A_2$

  For $p \in Q_1$, $q \in Q_2$, and $Z \in \Gamma$,

  (1) $\delta((p, q), a, Z) = \{((p', q'), \alpha) \mid (p', \alpha) \in \delta_1(p, a, Z) \text{ and } \delta_2(q, a) = q'\}$

  for every $a \in \Sigma$; and

  (2) $\delta((p, q), \Lambda, Z) = \{((p', q'), \alpha) \mid (p', \alpha) \in \delta_1(p, \Lambda, Z)\}$

  Showing that M accepts $L_1 \cap L_2$ depends on proving the following statement: For any $n \geq 0$ and any $p \in Q_1$, $q \in Q_2$, $y, z \in \Sigma^*$, and $\alpha \in \Gamma^*$,

  $(q_1, yz, Z_1) \vdash_{M_1}^{n} (p, z, \alpha) \quad \text{and} \quad \delta_2^*(q_2, y) = q$

  if and only if

  $((q_1, q_2), yz, Z_1) \vdash_{M}^{n} ((p, q), z, \alpha)$

  where $\vdash^n$ refers to a sequence of n moves. The "only if" direction shows that $L(M_1) \cap L(M_2) \subseteq L(M)$, and the converse shows the opposite inclusion.

  The two parts of the proof are both by induction on n and are very similar. The "only if" part is shown; the other direction is left as an exercise.

  For the basis step, suppose that

  $(q_1, yz, Z_1) \vdash_{M_1}^{0} (p, z, \alpha) \quad \text{and} \quad \delta_2^*(q_2, y) = q$

  This means that $y = \Lambda$, $p = q$, $\alpha = Z_1$, and $q = q_2$. In this case, it is clear that

  $((q_1, q_2), yz, Z_1) \vdash_{M}^{0} ((p, q), z, \alpha) = ((q_1, q_2), yz, Z_1)$

Intersections and Complements of Context-Free Languages (Continued)

Now suppose that $k \geq 0$ and that the statement is true for $n = k$, and assume that

$(q_1, yz, Z_1) \vdash_{M_1}^{k+1} (p, z, \alpha)$ and $\delta_2^*(q_2, y) = q$

The goal is to show that

$((q_1, q_2), yz, Z_1) \vdash_M^{k+1} ((p, q), z, \alpha)$

Consider the last move in the sequence of $k + 1$ moves of M_1. If it is a Λ-transition, then

$(q_1, yz, Z_1) \vdash_{M_1}^{k} (p', z, \beta) \vdash_{M_1} (p, z, \alpha)$

for some $p' \in Q_1$ and some $\beta \in \Gamma^*$. In this case the inductive hypothesis implies that

$((q_1, q_2), yz, Z_1) \vdash_M^{k} ((p', q), z, \beta)$

and from (2) we have

$((p', q), z, \beta) \vdash_M ((p, q), z, \alpha)$

which implies the result. Otherwise $y = y'a$ for some $a \in \Sigma$, and

$(q_1, y'az, Z_1) \vdash_{M_1}^{k} (p', az, \beta) \vdash_{M_1} (p, z, \alpha)$

Let $q' = \delta_2^*(q_2, y')$. Then the induction hypothesis implies that

$((q_1, q_2), y'az, Z_1) \vdash_M^{k} ((p', q'), az, \beta)$

and from (1) we have

$((p', q'), az, \beta) \vdash_M ((p, q), z, \alpha)$

Again the result follows.

Intersections and Complements of Context-Free Languages (Continued)

- Showing that the complement of a regular language is regular is easy: If the finite automaton $M = (Q, \Sigma, q_0, A, \delta)$ recognizes the language L, then the finite automaton $M' = (Q, \Sigma, q_0, Q - A, \delta)$ recognizes $\Sigma^* - L$.

- Applying the same construction to a PDA does not work, because PDAs are nondeterministic. For a particular input string x, a PDA may have a computation that accepts the string and a computation that rejects it. As a result, x would be accepted by M as well as by M'.

- What if the construction is applied to a *deterministic* PDA (DPDA)? It still does not work, unfortunately. There might be input strings that cause M to enter an infinite sequence of Λ-transitions. Any such string would be accepted neither by M nor by M'.

- However, this difficulty can be resolved and a DPDA can be constructed that recognizes $\Sigma^* - L$. It follows that the complement of a deterministic context-free language (DCFL) is a DCFL, and in particular that any context-free language whose complement is not a CFL cannot be a DCFL.

8.3 Decision Problems Involving Context-Free Languages

- The decision problems discussed in Chapter 5 can also be formulated for context-free languages.

- For some of the questions, essentially the same algorithms work; for others, the inherent nondeterminism of PDAs requires new algorithms; and for some, *no* algorithm is possible.

- The membership problem for context-free languages is tricky to solve using PDAs, which are nondeterministic. A simpler approach is to use the fact that a CFG without Λ-productions or unit productions requires at most $2n - 1$ steps for the derivation of a string of length n.

- **Decision algorithm for the membership problem. (Given a pushdown automaton *M* and a string *x*, does *M* accept *x*?)** Use the construction in the proof of Theorem 7.4 to find a CFG G generating the language recognized by M.

  If $x = \Lambda$, use Algorithm FindNull in Section 6.5 to determine whether the start symbol of G is nullable. Otherwise, eliminate Λ-productions and unit productions from G, using Algorithms 6.1 and 6.2.

  Examine all derivations with one step, all those with two steps, and so on, until either a derivation of x has been found or all derivations of length $2|x| - 1$ have been examined. If no derivation of x has been found, M does not accept x.

- Since Theorems 7.2 and 7.4 provide ways to convert from a CFG to a PDA and vice versa, any of these decision problems can be formulated in terms of either a CFG or a PDA.

Decision Problems Involving Context-Free Languages (Continued)

- Consider the decision problems corresponding to problems 2 and 3 in Chapter 5, but stated in terms of CFGs:

  1. Given a CFG G, does it generate any strings? (Is $L(G) = \emptyset$?)
  2. Given a CFG G, is $L(G)$ finite?

- Theorem 8.1 provides a way of answering both questions. First, transform G into Chomsky normal form. Let G' be the resulting grammar, p the number of variables in G', and $n = 2^{p+1}$.

  If G' generates any strings, it must generate one of length less than n. Otherwise, apply the pumping lemma to a string u of minimal length ($\geq n$) generated by G'; then $u = vwxyz$, for some strings v, w, x, y, and z with $|wy| > 0$ and $vxz \in L(G')$– and this contradicts the minimality of u.

  Similarly, if $L(G')$ is infinite, there must be a string $u \in L(G')$ with $n \leq |u| < 2n$; the proof is virtually identical to that of the corresponding result in Chapter 5.

- **Decision algorithms for problems 1 and 2. (Given a CFG G, is $L(G) = \emptyset$? Is $L(G)$ finite?)** First, test whether Λ can be generated from G, using the algorithm for the membership problem. If it can, then $L(G) \neq \emptyset$.

  In any case, let G' be a Chomsky-normal-form grammar generating $L(G) - \{\Lambda\}$, and let $n = 2^{p+1}$, where p is the number of variables in G'.

  For increasing values of i beginning with 1, test strings of length i for membership in $L(G)$. If for no $i < n$ is there a string of length i in $L(G')$, and $\Lambda \notin L(G)$, then $L(G') = L(G) = \emptyset$.

  If for no i with $n \leq i < 2n$ is there a string of length i in $L(G)$, then $L(G)$ is finite; if there is a string $x \in L(G)$ with $n \leq |x| < 2n$, then $L(G)$ is infinite.

- For some problems, such as the membership problem, there are much more efficient algorithms, including the ones by Cocke-Younger-Kasami and Earley.

- Many decision problems for CFGs and PDAs have no possible decision algorithm, including the problem of determining whether two CFGs generate any strings in common. These problems are said to be “unsolvable.”