7.2.4 Exercises for Section 7.2

Exercise 7.2.1: Use the CFL pumping lemma to show each of these languages not to be context-free:

* a) \( \{ a^i b^j c^k \mid i < j < k \} \).
  b) \( \{ a^i b^j c^k \mid i \leq n \} \).
  c) \( \{ 0^p \mid p \text{ is a prime} \} \). *Hint: Use the same ideas as in Example 4.3, which showed this language not to be regular.\)

*! d) \( \{ 0^{2j} \mid j = i^2 \} \).
  ! e) \( \{ a^i b^j c^k \mid n \leq i \leq 2n \} \).
  ! f) \( \{ awa^*w \mid w \text{ is a string of 0's and 1's} \} \). That is, the set of strings consisting of some string \( w \) followed by the same string in reverse, and then the string \( w \) again, such as 001000001.

Exercise 7.2.2: When we try to apply the pumping lemma to a CFL, the "adversary wins," and we cannot complete the proof. Show what goes wrong when we choose \( L \) to be one of the following languages:

  a) \( \{ 0^n1^n \} \).
  * b) \( \{ 0^n1^n \mid n \geq 1 \} \).
  * c) The set of palindromes over alphabet \( \{ 0, 1 \} \).

Exercise 7.2.3: There is a stronger version of the CFL pumping lemma known as Ogden's lemma. It differs from the pumping lemma we proved by allowing us to focus on any \( u \) "distinguished" positions of a string \( z \) and guaranteeing that the strings to be pumped have between 1 and \( n \) distinguished positions. The advantage of this ability is that a language may have strings consisting of two parts, one of which can be pumped without producing strings not in the language, while the other does produce strings outside the language when pumped. Without being able to insist that the pumping take place in the latter part, we cannot complete a proof of non-context-freeness. The formal statement of Ogden's lemma is: If \( L \) is a CFL, then there is a constant \( n \), such that if \( z \) is any string of length at least \( n \) in \( L \), in which we select at least \( n \) positions to be distinguished, then we can write \( z = uvwx \), such that:

1. \( uvx \) has at most \( n \) distinguished positions.
2. \( ex \) has at least one distinguished position.
3. For all \( i \), \( uv^iwx \) is in \( L \).

Prove Ogden's lemma. *Hint: The proof is really the same as that of the pumping lemma of Theorem 7.18 if we pretend that the non-distinguished positions of \( z \) are not present as we select \( n \) long path in the parse tree for \( z \)."

7.3 Closure Properties of Context-Free Languages

We shall now consider some of the operations on context-free languages that are guaranteed to produce a CFL. Many of these closure properties will parallel the theorems we had for regular languages in Section 4.2. However, there are some differences.

First, we introduce an operation called substitution, in which we replace each symbol in the strings of one language by an entire language. This operation, a generalization of the homomorphism that we studied in Section 4.2, is useful in proving some other closure properties of CFL's, such as the regular-expression operations: union, concatenation, and closure. We show that CFL's are closed under homomorphisms and inverse homomorphisms. Unlike the regular languages, the CFL’s are not closed under intersection or difference. However, the intersection or difference of a CFL and a regular language is always a CFL.

7.3.1 Substitutions

Let \( \Sigma \) be an alphabet, and suppose that for every symbol \( a \) in \( \Sigma \), we choose a language \( L_a \). These chosen languages can be over any alphabets, not necessarily \( \Sigma \) and not necessarily the same. This choice of languages defines a function \( s \) (a substitution) on \( \Sigma \), and we shall refer to \( L_s \) as \( s(a) \) for each symbol \( a \).

If \( w = a_1a_2 \cdots a_n \) is a string in \( \Sigma^* \), then \( s(w) \) is the language of all strings \( x_1x_2 \cdots x_n \) such that string \( x_i \) is in the language \( s(a_i) \), for \( i = 1, 2, \ldots, n \). Put another way, \( s(w) \) is the concatenation of the languages \( s(a_1)s(a_2) \cdots s(a_n) \).

We can further extend the definition of \( s \) to apply to languages: \( s(L) \) is the union of \( s(w) \) for all strings \( w \) in \( L \).

Example 7.22: Suppose \( s(0) = \{ a^i b^i \mid i \geq 1 \} \) and \( s(1) = \{ aa, bb \} \). That is, \( s \) is a substitution on alphabet \( \Sigma = \{ 0, 1 \} \). Language \( s(0) \) is the set of strings with one or more \( a \)'s followed by an equal number of \( b \)'s, while \( s(1) \) is the finite language consisting of the two strings \( aa \) and \( bb \).
Let \( w = 01 \). Then \( s(w) \) is the concatenation of the languages \( s(0)s(1) \). To be exact, \( s(w) \) consists of all strings of the forms \( a^n b^n a^n \) and \( a^n b^{n+2} \), where \( n \geq 1 \).

Now suppose \( L = L(0^k) \); that is, the set of all strings of 0's. Then \( s(L) = (s(0))^* \). This language is the set of all strings of the form

\[ a^{n_1} b^{n_2} a^{n_3} b^{n_4} \cdots a^{n_k} b^{n_k} \]

for some \( k \geq 0 \) and any sequence of choices of positive integers \( n_1, n_2, \ldots, n_k \).

It includes strings such as \( \varepsilon, aabbbaabb, \) and \( aabbbaabbab \).

**Theorem 7.23:** If \( L \) is a context-free language over alphabet \( \Sigma \), and \( s \) is a substitution on \( \Sigma \) such that \( s(a) \) is a CFL for each \( a \) in \( \Sigma \), then \( s(L) \) is a CFL.

**Proof:** The essential idea is that we may take a CFG for \( L \) and replace each terminal \( a \) by the start symbol of a CFG for language \( s(a) \). The result is a single CFG that generates \( s(L) \). However, there are a few details that must be gotten right to make this idea work.

More formally, start with grammars for each of the relevant languages, say \( G = (V, \Sigma, P, S) \) for \( L \) and \( G_a = (V_a, T_a, P_a, S_a) \) for each \( a \) in \( \Sigma \). Since we can choose any names we wish for variables, let us make sure that the sets of variables are disjoint; that is, there is no symbol \( A \) that is in two or more of \( V \) and any of the \( V_a \)'s. The purpose of this choice of names is to make sure that when we combine the productions of the various grammars into one set of productions, we cannot get accidental mixing of the productions from two grammars and thus have derivations that do not resemble the derivations in any of the given grammars.

We construct a new grammar \( G' = (V', T', P', S) \) for \( s(L) \), as follows:

- \( V' \) is the union of \( V \) and all the \( V_a \)'s for \( a \) in \( \Sigma \).
- \( T' \) is the union of all the \( T_a \)'s for \( a \) in \( \Sigma \).
- \( P' \) consists of:
  1. All productions in any \( P_a \), for \( a \) in \( \Sigma \).
  2. The productions of \( P \), but with each terminal \( a \) in their bodies replaced by \( S_a \) everywhere \( a \) occurs.

Thus, all parse trees in grammar \( G' \) start out like parse trees in \( G \), but instead of generating a yield in \( \Sigma^* \), there is a frontier in the tree where all nodes have labels that are \( S_a \) for some \( a \) in \( \Sigma \). Then, depending on which node is a parse tree of \( G_a \), whose yield is a terminal string that is in the language \( s(a) \).

The typical parse tree is suggested in Fig. 7.8.

Now, we must prove that this construction works, in the sense that \( G' \) generates the language \( s(L) \). Formally:

- A string \( w \) is in \( L(G') \) if and only if \( w \) is in \( s(L) \).

**7.3. Closure Properties of Context-Free Languages**

There are several familiar closure properties, which we studied for regular languages, that we can show for CFL's using Theorem 7.23. We shall list them all in one theorem.

![Figure 7.8: A parse tree in \( G' \) begins with a parse tree in \( G \) and finishes with many parse trees, each in one of the grammars \( G_a \).](image)
7.3. Closure Properties of Context-Free Languages

7.3.4 Intersection With a Regular Language

The CFL's are not closed under intersection. Here is a simple example that proves they are not.

Example 7.26: We learned in Example 7.19 that the language

\[ L = \{0^n1^n \mid n \geq 1 \} \]

is not a context-free language. However, the following two languages are context-free:

\[ L_1 = \{0^i1^j \mid i \geq 1, j \geq 1 \} \]
\[ L_2 = \{0^i2^j \mid i \geq 1, j \geq 1 \} \]

A grammar for \( L_1 \) is:

\[
S \rightarrow AB \\
A \rightarrow 0A1 \mid 01 \\
B \rightarrow 2B \mid 2
\]

In this grammar, \( A \) generates all strings of the form \( 0^n1^n \), and \( B \) generates all strings of \( 2^n \). A grammar for \( L_2 \) is:

\[
S \rightarrow AB \\
A \rightarrow 0A \mid 0 \\
B \rightarrow 1B2 \mid 12
\]

It works similarly, but with \( A \) generating any string of \( 0^n \)'s, and \( B \) generating matching strings of \( 1^n \)'s and \( 2^n \)'s.

However, \( L = L_1 \cap L_2 \). To see why, observe that \( L_1 \) requires that there be the same number of \( 0 \)'s and \( 1 \)'s, while \( L_2 \) requires the numbers of \( 1 \)'s and \( 2 \)'s to be equal. A string in both languages must have equal numbers of all three symbols and thus be in \( L \).

If the CFL's were closed under intersection, then we could prove the false statement that \( L \) is context-free. We conclude by contradiction that the CFL's are not closed under intersection. \( \square \)

On the other hand, there is a weaker claim we can make about intersection. The context-free languages are closed under the operation of "intersection with a regular language." The formal statement and proof is in the next theorem.

Theorem 7.27: If \( L \) is a CFL and \( R \) is a regular language, then \( L \cap R \) is a CFL.
7.3. CLOSURE PROPERTIES OF CONTEXT-FREE LANGUAGES

Let \( P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \delta_0) \). We leave these inductions as exercises. Since \((q, p)\) is an accepting state of \( P' \) if and only if \( p \) is an accepting state of \( P \), and \( p \) is an accepting state of \( A \), we conclude that \( P' \) accepts \( w \) if and only if both \( P \) and \( A \) do; i.e., \( w \) is in \( L \cap R \). □

Example 7.28: In Fig. 6.6 we designed a PDA called \( F \) to accept by final state the set of strings of \( i \)'s and \( c \)'s that represent minimal violations of the rule regarding how \( if \)'s and else's may appear in C programs. Call this language \( L \). The PDA \( F \) was defined by

\[
P_F = \{ (p, q, r), (i, c), (Z, X_0), \delta_F, p, X_0, \{ r \} \}
\]

where \( \delta_F \) consists of the rules:

1. \( \delta_F(q, c, X_0) = \{ (q, ZX_0) \} \).
2. \( \delta_F(q, i, Z) = \{ (q, ZZ) \} \).
3. \( \delta_F(q, c, Z) = \{ (q, c) \} \).
4. \( \delta_F(q, c, X_0) = \{ (r, c) \} \).

Now, let us introduce a finite automaton

\[
A = \{ (s, t), (i, c), \delta_A, s, \{ s, t \} \}
\]

that accepts the strings in the language of \( \Gamma \epsilon^* \), that is, all strings of \( i \)'s followed by \( c \)'s. Call this language \( R \). Transition function \( \delta_A \) is given by the rules:

a) \( \delta_A(s, i) = s \).

b) \( \delta_A(s, c) = t \).

c) \( \delta_A(t, c) = t \).

Strictly speaking, \( A \) is not a DFA, as assumed in Theorem 7.27, because it is missing a dead state for the case that we see input \( i \) when in state \( t \). However, the same construction works even for an NFA, since the PDA that we construct is allowed to be nondeterministic. In this case, the constructed PDA is actually deterministic, although it will “die” on certain sequences of input.

We shall construct a PDA

\[
P = \{ \{ p, q, r \} \times \{ s, t \}, \{ i, c \}, \{ Z, X_0 \}, \delta, \{ p, s \}, X_0, \{ r \} \}
\]

The transitions of \( \delta \) are listed below and indexed by the rule of PDA \( F \) (a number from 1 to 4) and the rule of DFA \( A \) (an letter a, b, or c) that gives rise to the rule. In the case that the PDA \( F \) makes an \( \epsilon \)-transition, there is no rule of \( A \) used. Note that we construct these rules in a “lazy” way, starting with the state of \( P \) that is the start states of \( F \) and \( A \), and constructing rules for other states only if we discover that \( P \) can enter that pair of states.
were always a CFL when \( L_1 \) and \( L_2 \) are, it would follow that \( \Sigma^* - L \) was always a CFL when \( L \) is. However, \( \Sigma^* - L \) is \( \bar{L} \) when we pick the proper alphabet \( \Sigma \). Thus, we would contradict (2) and we have proved by contradiction that \( L_1 - L_2 \) is not necessarily a CFL. □

### 7.3.5 Inverse Homomorphism

Let us review from Section 4.2.4 the operation called “inverse homomorphism.” If \( h \) is a homomorphism, and \( L \) is any language, then \( h^{-1}(L) \) is the set of strings \( w \) such that \( h(w) \) is in \( L \). The proof that regular languages are closed under inverse homomorphism was suggested in Fig. 4.6. There, we showed how to design a finite automaton that processes its input symbols \( a \) by applying a homomorphism \( h \) to it, and simulating another finite automaton on the sequence of inputs \( h(a) \).

We can prove this closure property of CFL's in much the same way, by using PDA's instead of finite automata. However, there is one problem that we face with PDA's that did not arise when we were dealing with finite automata. The action of a finite automaton on a sequence of inputs is a state transition, and thus looks, as far as the constructed automaton is concerned, just like a move that a finite automaton might make on a single input symbol. When the automaton is a PDA, in contrast, a sequence of moves might not look like a move on one input symbol. In particular, in \( n \) moves, the PDA can pop \( n \) symbols off its stack, while one move can only pop one symbol. Thus, the construction for PDA's that is analogous to Fig. 4.6 is somewhat more complex; it is sketched in Fig. 7.10. The key additional idea is that after input \( a \) is read, \( h(a) \) is placed in a “buffer.” The symbols of \( h(a) \) are used one at a time, and fed to the PDA being simulated. Only when the buffer is empty does the constructed PDA read another of its input symbols and apply the homomorphism to it. We shall formalize this construction in the next theorem.

#### Theorem 7.30

Let \( L \) be a CFL and \( h \) a homomorphism. Then \( h^{-1}(L) \) is a CFL.

**Proof:** Suppose \( h \) applies to symbols of alphabet \( \Sigma \) and produces strings in \( \Sigma^* \). We also assume that \( L \) is a language over alphabet \( T \). As suggested above, we start with a PDA \( P = (Q, T, \Gamma, b, q_0, Z_0, F) \) that accepts \( L \) by final state. We construct a new PDA

\[
P' = (Q', \Sigma', \Gamma', \delta', (q_0, \epsilon), Z_0, F \times \{\epsilon\})
\]

where:

1. \( Q' \) is the set of pairs \((q, x)\) such that:
   1. \( q \) is a state in \( Q \), and
   2. \( x \) is a suffix (not necessarily proper) of some string \( h(a) \) for some input symbol \( a \) in \( \Sigma \).

2. \( \Gamma' \) is the set of strings \( w \) over \( \Sigma \) such that:
   3. \( w \) is a prefix of some string \( h(a) \) for some input symbol \( a \) in \( \Sigma \).

3. \( \delta' \) is defined such that:
   4. \( \delta'(q, x, z) = \delta(q, b, z') \) if \( z = z' \epsilon \).

4. \( Z_0 \) is the initial stack symbol.

5. \( F \) is the set of final states.

This construction ensures that the new PDA simulates the original PDA for each input symbol, while also applying the homomorphism. Thus, \( h^{-1}(L) \) is a CFL.
7.3. CLOSURE PROPERTIES OF CONTEXT-FREE LANGUAGES

- \((q_0, h(w), Z_0) \xrightarrow{\epsilon} (p, \epsilon, \gamma)\) if and only if \((\langle q_0, \epsilon \rangle, w, Z_0) \xrightarrow{\epsilon} (\langle p, \epsilon \rangle, \epsilon, \gamma)\).

The proofs in both directions are inductions on the number of moves made by the two automata. In the "if" portion, one needs to observe that once the buffer of \(P'\) is nonempty, it cannot read another input symbol and must simulate \(P\) until the buffer has become empty (although when the buffer is empty, it may still simulate \(P\)). We leave further details as an exercise.

Once we accept this relationship between \(P'\) and \(P\), we note that \(P\) accepts \(h(w)\) if and only if \(P'\) accepts \(w\), because of the way the accepting states of \(P'\) are defined. Thus, \(L(P') = h^{-1}(L(P))\). 

7.3.6 Exercises for Section 7.3

Exercise 7.3.1: Show that the CFL's are closed under the following operations:
* a) min, defined in Exercise 4.2.6(c). Hint: Start with a CNF grammar for the language \(L\).
* b) The operation \(L/\alpha\), defined in Exercise 4.2.2. Hint: Again, start with a CNF grammar for \(L\).
* c) cycle, defined in Exercise 4.2.11. Hint: Try a PDA-based construction.

Exercise 7.3.2: Consider the following two languages:

\[ L_1 = \{a^m b^n c^m | m, n \geq 0\} \]
\[ L_2 = \{a^n b^m c^{2m} | n, m \geq 0\} \]

a) Show that each of these languages is context-free by giving grammars for each.

b) Is \(L_1 \cap L_2\) a CFL? Justify your answer.

!! Exercise 7.3.3: Show that the CFL's are not closed under the following operations:
* a) min, as defined in Exercise 4.2.6(a).
* b) max, as defined in Exercise 4.2.6(b).
* c) half, as defined in Exercise 4.2.8.
* d) alt, as defined in Exercise 4.2.7.

Exercise 7.3.4: The shuffle of two strings \(w\) and \(x\) is the set of all strings that one can get by interleaving the positions of \(w\) and \(x\) in any way. More precisely, \(shuffle(w, x)\) is the set of strings \(z\) such that

1. Each position of \(z\) can be assigned to \(w\) or \(x\), but not both.