Exercise 5.4.2: Prove that the grammar of Exercise 5.4.1 generates all and only the strings of a's and b's such that every prefix has at least as many a’s as b’s.

*! Exercise 5.4.3: Find an unambiguous grammar for the language of Exercise 5.4.1.

!! Exercise 5.4.4: Some strings of a’s and b’s have a unique parse tree in the grammar of Exercise 5.4.1. Give an efficient test to tell whether a given string is one of these. The test “try all parse trees to see how many yield the given string” is not adequately efficient.

! Exercise 5.4.5: This question concerns the grammar from Exercise 5.1.2, which we reproduce here:

\[
\begin{align*}
S & \rightarrow A1B \\
A & \rightarrow 0A \mid \epsilon \\
B & \rightarrow 0B \mid 1B \mid \epsilon
\end{align*}
\]

a) Show that this grammar is unambiguous.

b) Find a grammar for the same language that is ambiguous, and demonstrate its ambiguity.

*! Exercise 5.4.6: Is your grammar from Exercise 5.1.5 unambiguous? If not, redesign it to be unambiguous.

Exercise 5.4.7: The following grammar generates prefix expressions with operands x and y and binary operators +, −, and ∗:

\[E \rightarrow +EE \mid ∗EE \mid −EE \mid x \mid y\]

a) Find leftmost and rightmost derivations, and a derivation tree for the string **-xyxy.

!! b) Prove that this grammar is unambiguous.

5.5 Summary of Chapter 5

✦ Context-Free Grammars: A CFG is a way of describing languages by recursive rules called productions. A CFG consists of a set of variables, a set of terminal symbols, and a start variable, as well as the productions. Each production consists of a head variable and a body consisting of a string of zero or more variables and/or terminals.

✦ Derivations and Languages: Beginning with the start symbol, we derive terminal strings by repeatedly replacing a variable by the body of some production with that variable in the head. The language of the CFG is the set of terminal strings we can so derive; it is called a context-free language.
the input (second component) in each ID is also legal.

2. If a computation is legal for a PDA \( P \), then the computation formed by adding the same additional stack symbols below the stack in each ID is also legal.

3. If a computation is legal for a PDA \( P \), and some tail of the input is not consumed, then we can remove this tail from the input in each ID, and the resulting computation will still be legal.

Intuitively, data that \( P \) never looks at cannot affect its computation. We formalize points (1) and (2) in a single theorem.

**Theorem 6.5**: If \( P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) \) is a PDA, and \( (q, x, \alpha) \xrightarrow{w}^* (p, y, \beta) \), then for any strings \( w \) in \( \Sigma^* \) and \( \gamma \) in \( \Gamma^* \), it is also true that

\[
(q, x w, \alpha \gamma) \xrightarrow{w}^* (p, y w, \beta \gamma)
\]

Note that if \( \gamma = \epsilon \), then we have a formal statement of principle (1) above, and if \( w = \epsilon \), then we have the second principle.

**Proof**: The proof is actually a very simple induction on the number of steps in the sequence of ID’s that take \( (q, x w, \alpha \gamma) \) to \( (p, y w, \beta \gamma) \). Each of the moves in the sequence \( (q, x, \alpha) \xrightarrow{w}^* (p, y, \beta) \) is justified by the transitions of \( P \) without using \( w \) and/or \( \gamma \) in any way. Therefore, each move is still justified when these strings are sitting on the input and stack. \( \square \)

Incidentally, note that the converse of this theorem is false. There are things that a PDA might be able to do by popping its stack, using some symbols of \( \gamma \), and then replacing them on the stack, that it couldn’t do if it never looked at \( \gamma \). However, as principle (3) states, we can remove unused input, since it is not possible for a PDA to consume input symbols and then restore those symbols to the input. We state principle (3) formally as:

**Theorem 6.6**: If \( P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) \) is a PDA, and

\[
(q, x w, \alpha) \xrightarrow{w}^* (p, y w, \beta)
\]

then it is also true that \( (q, x, \alpha) \xrightarrow{w}^* (p, y, \beta) \). \( \square \)

### 6.1.5 Exercises for Section 6.1

**Exercise 6.1.1**: Suppose the PDA \( P = (\{q, p\}, \{0, 1\}, \{Z_0, X\}, \delta, q, Z_0, \{p\}) \) has the following transition function:

1. \( \delta(q, 0, Z_0) = \{(q, XZ_0)\} \).
One might wonder why we did not introduce for finite automata a notation like the ID’s we use for PDA’s. Although a FA has no stack, we could use a pair \((q, w)\), where \(q\) is the state and \(w\) the remaining input, as the ID of a finite automaton.

While we could have done so, we would not glean any more information from reachability among ID’s than we obtain from the \(\delta\) notation. That is, for any finite automaton, we could show that \(\delta(q, w) = p\) if and only if \((q, wx) \rightarrow^* (p, x)\) for all strings \(x\). The fact that \(x\) can be anything we wish without influencing the behavior of the FA is a theorem analogous to Theorems 6.5 and 6.6.

\begin{enumerate}
  \item \(\delta(q, 0, X) = \{(q, XX)\}\).
  \item \(\delta(q, 1, X) = \{(q, X)\}\).
  \item \(\delta(q, e, X) = \{(p, e)\}\).
  \item \(\delta(p, e, X) = \{(p, e)\}\).
  \item \(\delta(p, 1, X) = \{(p, XX)\}\).
  \item \(\delta(p, 1, Z_0) = \{(p, e)\}\).
\end{enumerate}

Starting from the initial ID \((q, w, Z_0)\), show all the reachable ID’s when the input \(w\) is:

* a) 01.
  b) 0011.
  c) 010.

\section{6.2 The Languages of a PDA}

We have assumed that a PDA accepts its input by consuming it and entering an accepting state. We call this approach “acceptance by final state.” There is a second approach to defining the language of a PDA that has important applications. We may also define for any PDA the language “accepted by empty stack,” that is, the set of strings that cause the PDA to empty its stack, starting from the initial ID.

These two methods are equivalent, in the sense that a language \(L\) has a PDA that accepts it by final state if and only if \(L\) has a PDA that accepts it by empty stack. However, for a given PDA \(P\), the languages that \(P\) accepts