Chapter 3: Context-Free Languages

A. Context-free grammars

1. Introduction and motivation.
   - Recall the distinction between language recognizers (e.g., DFA) and language generators (e.g., regular expressions).
   - Recall that not all languages are regular. What is the next most general class of languages? Is there some more general FA that recognizes such languages?
   - This material on "formal languages" has applications both in (programming) language processing and in the theory of computation in general.
2. **Definition**. A **context-free grammar** is a quadruple $G = (V, \Sigma, R, S)$ where

   V is an alphabet,
   $\Sigma \subset V$ is a set of
   $R \subset (V - \Sigma) \times V^*$ is a finite set of
   $S \in V - \Sigma$ is the
3. Terminology and notation.
   - $V - \Sigma$ is the set of
   - A grammar is **context sensitive** if
   - We write $A \rightarrow u$ if $(A, u) \in R$.
   - For any $u, v \in V^*$, we write $u \Rightarrow v$ if v can be **derived** from u using one of the rules.
   - The (context-free) language generated by G is $L(G)$
4. Examples.

B. Regular grammars

1. **Definition**. A context-free grammar $G = (V, \Sigma, R, S)$ is **regular** iff

2. **Theorem 3.2.1**. A language is regular iff it is generated by a regular grammar.
3. Examples.

C. Pushdown automata

1. Introduction and motivation.
   - Recall that a language is regular iff it is accepted by a DFA.
   - A typical requirement in recognizing a nonregular CFL is "remembering" some arbitrary amount of information.
   - What about a FA for CFLs? Some motivational examples:

2. **Definition**. A **pushdown automaton** (PDA) is a 6-tuple $(K, \Sigma, \Gamma, \Delta, s, F)$ where

   K is a finite set of states,
   Σ is an alphabet of input symbols,
   Γ is an alphabet of stack symbols,
   $\Delta \subset (K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$ is a
   $s \in K$ is the
   $F \subseteq K$ is a set of

3. Terminology and notation.
   - **Applying the transition** $((p, u, \beta), (q, \gamma))$ means
   - A **configuration** of a PDA is a member of
   - A machine M **accepts** a string $w \in \Sigma^*$ iff
   - The language $L(M)$ accepted by M is defined in the usual way.

4. Examples.

5. **Lemma 3.4.3**. Each CFL is accepted by a PDA.
6. **Theorem 3.4.1**. The class of languages accepted by PDAs is exactly the class of CFLs.
7. Remarks:

8. Example.

D. Properties of context-free languages

1. **Theorem 3.5.1**. The CFLs are closed under union, concatenation, and Kleene star.
2. **Theorem 3.5.2**. The intersection of a CFL with a regular language is a CFL.
3. **Theorem 3.5.3** (a pumping theorem). Let $L(G)$ by a CFL. Then $\exists K$, depending on G, such that any $w \in L(G)$ with $|w| > K$ can be written $w = uvxyz$ where at least one of v or y is nonempty and $uv^nxy^nz \in L(G), n \geq 0$.
4. **Theorem 3.5.4**. $\{a^nb^nc^n \mid n \geq 0\}$ is not context-free.
5. **Theorem 3.5.5**. The CFLs are not closed under intersection or complementation.
6. **Theorem 3.5.8**. There are algorithms for answering the following questions about CFLs:

E. Parsing

1. Parse trees.
   - A convenient representation of a derivation.
   - Start symbol is the root, nonterminal symbols are interior nodes, terminal symbols are the leaves.
   - A left-to-right traversal of the leaves corresponds to the input.
   - Often the job of a parser is to build a parse tree.
2. Deterministic context-free languages.
   - Recognized by deterministic PDAs.
   - May require the ability to indicate end of input with a special symbol.
   - Theorem 3.6.1 characterizes a set of languages that are *not* deterministic context-free.
3. Top-down parsing
   - Idea: construct a parse tree from the top down. PDA typically finds a leftmost derivation.
   - **LL(k) grammars**: a set of CFGs that permit deterministic left-to-right top-down recognition with a k-symbol lookahead.
   - **Recursive descent parsers**: a famous technique for implementing a PDA for an LL(k) grammar.
   - A deterministic PDA for Pascal0:
   - *Sometimes*, a grammar can be modified so that a deterministic PDA can be easily constructed from it.
   - Removing "left recursion" from the expression language $L(G_E)$:

   - A PDA for $L(G_E)$.
4. Bottom-up parsing
   - Idea: construct a parse tree from the bottom up. PDA typically finds a rightmost derivation (in reverse).
   - **LR(k) grammars**: a set of CFGs that permit deterministic left-to-right bottom-up recognition with a k-symbol lookahead.
   - **Precedence parsers** (shift/reduce parsers): a famous technique for implementing a PDA for an LR(k) grammar.
   - A bottom-up PDA for a grammar $G = (V, \Sigma, R, S)$:

   - A bottom-up parser for $L(G_E)$.