CHAPTER 14

Algorithms and Decision Procedures for Context-Free Languages

Many questions that we could answer when asked about regular languages are unanswerable for context-free ones. But a few important questions can be answered and we have already presented a useful collection of algorithms that can operate on context-free grammars and PDAs. We'll present a few more here.

14.1 The Decidable Questions

Fortunately, the most important questions (i.e., the ones that must be answerable if context-free grammars are to be of any practical use) are decidable.

14.1.1 Membership

We begin with the most fundamental question, "Given a language L and a string w, is w in L?" Fortunately this question can be answered for every context-free language. By Theorem 12.1, for every context-free language L, there exists a PDA M such that M accepts L. But we must be careful. As we showed in Section 12.4, PDAs are not guaranteed to halt. So the mere existence of a PDA that accepts L does not guarantee the existence of a procedure that decides it (i.e., always halts and says yes or no appropriately).

It turns out that there are two alternative approaches to solving this problem, both of which work:

- Use a grammar: Using facts about every derivation that is produced by a grammar in Chomsky normal form, we can construct an algorithm that explores a finite number of derivation paths and finds one that derives a particular string w iff such a path exists.

- Use a PDA: While not all PDAs halt, it is possible, for any context-free language L, to craft a PDA M that is guaranteed to halt on all inputs and that accepts all strings in L and rejects all strings that are not in L.

Using a Grammar to Decide

We begin by considering the first alternative. We show a straightforward algorithm for deciding whether a string w is in a language L:

decideCFLusingGrammar(L: CFL, w: string) =

1. If L is specified as a PDA, use *PDAtoCFG*, presented in the proof of Theorem 12.2, to construct a grammar G such that $L(G) = L(M)$.
2. If L is specified as a grammar G, simply use G.
3. If $w = \varepsilon$ then if S_G is nullable (as defined in the description of *removeEps* in Section 11.7.4) then accept, otherwise reject.
4. If $w \neq \varepsilon$ then:
   - 4.1. From G, construct G' such that $L(G') = L(G) - \{\varepsilon\}$ and G' is in Chomsky normal form.
   - 4.2. If G derives w, it does so in $2 \cdot |w| - 1$ steps. Try all derivations in G of that number of steps. If one of them derives w, accept. Otherwise reject.

The running time of *decideCFLusingGrammar* can be analyzed as follows: We assume that the time required to build G' is constant, since it does not depend on w. Let $n = |w|$. Let g be the search-branching factor of G', defined to be the maximum number of rules that share a left-hand side. Then the number of derivations of length $2n - 1$ is bounded by g^{2n-1}, and it takes at most $2n - 1$ steps to check each one. So the worst-case running time of *decideCFLusingGrammar* is $O(n2^n)$. In Section 15.3.1, we will present techniques that are substantially more efficient. We will describe the CKY algorithm, which, given a grammar G in Chomsky normal form, decides the membership question for G in time that is $O(n^3)$. We will then describe an algorithm that can decide the question in time that is linear in n if the grammar that is provided meets certain requirements.

THEOREM 14.1 Decidability of Context-Free Languages

Theorem: Given a context-free language L (represented as either a context-free grammar or a PDA) and a string w, there exists a decision procedure that answers the question, "Is $w \in L$?"

Proof: The following algorithm, *decideCFL*, uses *decideCFLusingGrammar* to answer the question:

decideCFL(L: CFL, w: string) =

1. If *decideCFLusingGrammar*(L, w) accepts, return *True* else return *False*.

Using a PDA to Decide ●

It is also possible to solve the membership problem using PDAs. We take a two-step approach. We first show that, for every context-free language L, it is possible to build a PDA that accepts $L - \{\varepsilon\}$ and that has no ε-transitions. Then we show that every PDA with no ε-transitions is guaranteed to halt.

THEOREM 14.2 Elimination of ε-Transitions

Theorem: Given any context-free grammar $G = (V, \Sigma, R, S)$, there exists a PDA M such that $L(M) = L(G) - \{\varepsilon\}$ and M contains no transitions of the form $((q_1, \varepsilon, \alpha), (q_2, \beta))$. In other words, every transition reads exactly one input character.

Proof: The proof is by a construction that begins by converting G to Greibach normal form. Recall that, in any grammar in Greibach normal form, all rules are of the form $X \rightarrow a\,A$, where $a \in \Sigma$ and $A \in (V - \Sigma)^*$. Now consider again the algorithm *cfgtoPDAtopdown*, which builds, from any context-free grammar G, a PDA M that, on input w, simulates G deriving w, starting from S. $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

1. The start-up transition $((p, \varepsilon, \varepsilon), (q, S))$, which pushes the start symbol onto the stack and goes to state q.
2. For each rule $X \rightarrow s_1 s_2 \ldots s_n$ in R, the transition $((q, \varepsilon, X), (q, s_1 s_2 \ldots s_n))$, which replaces X by $s_1 s_2 \ldots s_n$. If $n = 0$ (i.e., the right-hand side of the rule is ε), then the transition $((q, \varepsilon, X), (q, \varepsilon))$.
3. For each character $c \in \Sigma$, the transition $((q, c, c), (q, \varepsilon))$, which compares an expected character from the stack against the next input character and continues if they match.

The start-up transition, plus all the transitions generated in step 2, are ε-transitions. But now suppose that G is in Greibach normal form. If G contains the rule $X \rightarrow c s_2 \ldots s_n$ (where $c \in \Sigma$ and s_2 through s_n are elements of V-Σ), it is not necessary to push c onto the stack, only to pop it with a rule from step 3. Instead, we collapse the push and the pop into a single transition. So we create a transition that can be taken only if the next input character is c. In that case, the string $s_2 \ldots s_n$ is pushed onto the stack.

Now we need only find a way to get rid of the start-up transition, whose job is to push S onto the stack so that the derivation process can begin. Since G is in Greibach normal form, any rules with S on the left-hand side must have the form $S \rightarrow c s_2 \ldots s_n$. So instead of reading no input and just pushing S, M will skip pushing S and instead, if the first input character is c, read it and push the string $s_2 \ldots s_n$.

Since terminal symbols are no longer pushed onto the stack, we no longer need the transitions created in step 3 of the original algorithm.

So $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains:

1. The start-up transitions: For each rule $S \rightarrow cs_2 \ldots s_n$, the transition $((p, c, \varepsilon), (q, s_2 \ldots s_n))$.
2. For each rule $X \rightarrow cs_2 \ldots s_n$ (where $c \in \Sigma$ and s_2 through s_n are elements of $V - \Sigma$), the transition $((q, c, X), (q, s_2 \ldots s_n))$.

The following algorithm builds the required PDA:

cfgtoPDAnoeps(*G*: context-free grammar) =

1. Convert G to Greibach normal form, producing G'.
2. From G' build the PDA M described above.

THEOREM 14.3 Halting Behavior of PDAs Without ε-Transitions

Theorem: Let M be a PDA that contains no transitions of the form $((q_1, \varepsilon, s_1), (q_2, s_2))$, i.e., no ε-transitions. Consider the operation of M on input $w \in \Sigma^*$. M must halt and either accept or reject w. Let $n = |w|$. We make three additional claims:

a. Each individual computation of M must halt within n steps.

b. The total number of computations pursued by M must be less than or equal to b^n, where b is the maximum number of competing transitions from any state in M.

c. The total number of steps that will be executed by all computations of M is bounded by nb^n.

Proof:

a. Since each computation of M must consume one character of w at each step and M will halt when it runs out of input, each computation must halt within n steps.

b. M may split into at most b branches at each step in a computation. The number of steps in a computation is less than or equal to n. So the total number of computations must be less than or equal to b^n.

c. Since the maximum number of computations is b^n and the maximum length of each is n, the maximum number of steps that can be executed before all computations of M halt is nb^n.

So a second way to answer the question, "Given a context-free language L and a string w, is w in L?" is to execute the following algorithm:

decideCFLusingPDA(*L*: CFL, *w*: string) =

1. If L is specified as a PDA, use *PDAtoCFG*, as presented in the proof of Theorem 12.2, to construct a grammar G such that $L(G) = L(M)$.
2. If L is specified as a grammar G, simply use G.

3. If $w = \varepsilon$ then if S_G is nullable (as defined in the description of *removeEps* in Section 11.7.4) then accept, otherwise reject.
4. If $w \neq \varepsilon$ then:
   - 4.1. From G, construct G' such that $L(G') = L(G) - \{\varepsilon\}$ and G' is in Greibach normal form.
   - 4.2. From G' construct, using *cfgtoPDAnoeps*, the algorithm described in the proof of Theorem 14.2, a PDA M' such that $L(M') = L(G')$ and M' has no ε-transitions.
   - 4.3. By Theorem 14.3, all paths of M' are guaranteed to halt within a finite number of steps. So run M' on w. Accept if M' accepts and reject otherwise.

The running time of *decideCFLusingPDA* can be analyzed as follows: We will take as a constant the time required to build M', since that can be done once. It need not be repeated for each string that is to be analyzed. Given M', the time required to analyze a string w is then the time required to simulate all paths of M' on w. Let $n = |w|$. From Theorem 14.3, we know that the total number of steps that will be executed by all paths of M is bounded by nb^n, where b is the maximum number of competing transitions from any state in M'. But is that number of steps required? If one state has a large number of competing transitions but the others do not, then the average branching factor will be less than b, so fewer steps will be necessary. But if b is greater than 1, the number of steps still grows exponentially with n. The exact number of steps also depends on how the simulation is done. A straightforward depth-first search of the tree of possibilities will explore b^n steps, which is less than nb^n because it does not start each path over at the beginning. But it still requires time that is $\mathcal{O}(b^n)$. In Section 15.2.3, we present an alternative approach to top-down parsing that runs in time that is linear in n if the grammar that is provided meets certain requirements.

14.1.2 Emptiness and Finiteness

While many interesting questions are not decidable for context-free languages, two others, in addition to membership are: emptiness and finiteness.

THEOREM 14.4 Decidability of Emptiness and Finiteness

Theorem: Given a context-free language L, there exists a decision procedure that answers each of the following questions:

1. Given a context-free language L, is $L = \emptyset$?
2. Given a context-free language L, is L infinite?

Since we have proven that there exists a grammar that generates L iff there exists a PDA that accepts it, these questions will have the same answers whether we ask them about grammars or about PDAs.

Proof:

1. Let $G = (V, \Sigma, R, S)$ be a context-free grammar that generates L. $L(G) = \emptyset$ iff S is unproductive (i.e., not able to generate any terminal strings). The following algorithm exploits the procedure *removeunproductive*, defined in Section 11.4, to remove all unproductive nonterminals from G. It answers the question, "Given a context-free language L, is $L = \emptyset$?"

   decideCFLempty(G: context-free grammar) =

   1. Let G' = *removeunproductive* (G).
   2. If S is not present in G' then return *True* else return *False*.

2. Let $G = (V, \Sigma, R, S)$ be a context-free grammar that generates L. We use an argument similar to the one that we used to prove the context-free Pumping Theorem. Let n be the number of nonterminals in G. Let b be the branching factor of G. The longest string that G can generate without creating a parse tree with repeated nonterminals along some path is of length b^n. If G generates no strings of length greater than b^n, then $L(G)$ is finite. If G generates even one string w of length greater than b^n, then, by the same argument we used to prove the Pumping Theorem, it generates an infinite number of strings since $w = uvxyz$, $|vy| > 0$, and $\forall q \geq 0$ (uv^qxy^qz is in L). So we could try to test to see whether L is infinite by invoking *decideCFL*(L, w) on all strings in Σ^* of length greater than b^n. If it returns *True* for any such string, then L is infinite. If it returns *False* on all such strings, then L is finite.

   But, assuming Σ is not empty, there is an infinite number of such strings. Fortunately, it is necessary to try only a finite number of them. Suppose that G generates even one string of length greater than $b^{n+1} + b^n$. Let t be the shortest such string. By the Pumping Theorem, $t = uvxyz$, $|vy| > 0$, and uxz (the result of pumping vy out once) $\in L$. Note that $|uxz| < |t|$ since some nonempty vy was pumped out of t to create it. Since, by assumption, t is the shortest string in L of length greater than $b^{n+1} + b^n$, $|uxz|$ must be less than or equal to $b^{n+1} + b^n$. But the Pumping Theorem also tells us that $|vxy| \leq k$ (i.e., b^{n+1}), so no more than b^{n+1} strings could have been pumped out of t. Thus we have that $b^n < |uxz| \leq b^{n+1} + b^n$. So, if L contains any strings of length greater than b^n, it must contain at least one string of length less than or equal to $b^{n+1} + b^n$. We can now define *decideCFLinfinite* to answer the question, "Given a context-free language L, is L infinite?":

   decideCFLinfinite(G: context-free grammar) =

   1. Lexicographically enumerate all strings in Σ^* of length greater than b^n and less than or equal to $b^{n+1} + b^n$.
   2. If, for any such string w, *decideCFL*(L, w) returns *True* then return *True*. L is infinite.
   3. If, for all such strings w, *decideCFL*(L, w) returns *False* then return *False*. L is not infinite.

14.1.3 Equality of Deterministic Context-Free languages

THEOREM 14.5 Decidability of Equivalence for Deterministic Context-Free Languages

Theorem: Given two *deterministic* context-free languages L_1 and L_2, there exists a decision procedure to determine whether $L_1 = L_2$.

Proof: This claim was not proved until 1997 and the proof [Sénizergues 2001] is beyond the scope of this book, but see □.

14.2 The Undecidable Questions

Unfortunately, we will prove in Chapter 22 that there exists no decision procedure for many other questions that we might like to be able to ask about context-free languages, including:

- Given a context-free language L, is $L = \Sigma^*$?
- Given a context-free language L, is the complement of L context-free?
- Given a context-free language L, is L regular?
- Given two context-free languages L_1 and L_2, is $L_1 = L_2$? (Theorem 14.5 tells us that this question is decidable for the restricted case of two deterministic context-free languages. But it is undecidable in the more general case.)
- Given two context-free languages L_1 and L_2, is $L_1 \subseteq L_2$?
- Given two context-free languages L_1 and L_2, is $L_1 \cap L_2 = \emptyset$?
- Given a context-free language L, is L inherently ambiguous?
- Given a context-free grammar G, is G ambiguous?

14.3 Summary of Algorithms and Decision Procedures for Context-Free Languages

Although we have presented fewer algorithms and decision procedures for context-free languages than we did for regular languages, there are many important ones, which we summarize here:

- Algorithms that transform grammars:
  - *removeunproductive*(G: context-free grammar): Construct a grammar G' that contains no unproductive nonterminals and such that $L(G') = L(G)$.
  - *removeunreachable*(G: context-free grammar): Construct a grammar G' that contains no unreachable nonterminals and such that $L(G') = L(G)$.