

Limits of Computation

Week 2

Stephen Canon and Darius Pierce

Spring 2001

Contents

1	Language Classification	1
1.1	Review of Chomsky Normal Form	1
1.2	Decidability Problems for Context-free Languages	2
1.2.1	Notation	2
1.2.2	Decidable Problems	2
1.2.3	An Undecidable Problem	3
2	The Halting Problem (Coming on Friday . . .)	4

1 Language Classification

1.1 Review of Chomsky Normal Form

A context-free grammar is in *Chomsky normal form* if all of the rules are in one of the following forms:

$$A \longrightarrow BC \quad A \longrightarrow a \quad S \longrightarrow \epsilon$$

where a is any terminal and A, B and C are any variables - except that B and C may not be the *start variable*, S .

Theorem 1 *Any context-free grammar is equivalent to some grammar in Chomsky normal form.*

Proof: Given some context-free grammar, we will explicitly convert it into Chomsky normal form. Suppose we are given the grammar (V, Σ, R, S) . We begin by defining a new start symbol S_o , and adding the rule

$$S_o \longrightarrow S$$

This doesn't change the language that our grammar recognizes, but guarantees that our new start symbol, S_o , does not appear on the right hand side of any rule.

Next we need to eliminate all of the ϵ rules. If we find any rule of the form $A \rightarrow \epsilon$, we remove it; then for every rule with A on the right hand side, we add the same rule without A present. If, for example, we have rule $B \rightarrow uAv$, we add the rule $B \rightarrow uv$. If we have the rule $R \rightarrow A$, we add the rule $R \rightarrow \epsilon$, unless we've already removed that rule. We repeat this process until all ϵ rules except for (possibly) $S_o \rightarrow \epsilon$ have been removed.

If we have any unary rules (of the form $A \rightarrow B$), we add a rule of the form $A \rightarrow u_1u_2 \dots u_n$, for every expansion $B \rightarrow u_1u_2 \dots u_n$, and then delete the original rule. Again, we repeat this process until all unary rules have been removed.

At this point, all of our rules are in one of the following forms:

$$A \longrightarrow u_1 \dots u_n \quad A \longrightarrow a \quad S \longrightarrow \epsilon$$

The latter two are fine; we still need to convert the first form if either $n > 2$ or one of the u_i 's is a terminal. if $n > 2$, then we just add the rules

$$\begin{array}{ll} A \rightarrow u_1A_1 & A_1 \rightarrow u_2A_2 \\ A_i \rightarrow u_{i+1}A_{i+1} & A_{n-2} \rightarrow u_{n-1}u_n \end{array}$$

Finally, if any of the u_i 's are terminals, we replace them with the variable U_i and add the rules $U_i \rightarrow u_i$. At this point, our grammar is in Chomsky normal form.

Optional Exercise 1 Construct a context-free grammar that generates the language 0^n1^n , and convert it into Chomsky normal form.

1.2 Decidability Problems for Context-free Languages

1.2.1 Notation

Given some CFG G , we can represent G as a string that can be given as input to a turing machine. This string representation is denoted by $\langle G \rangle$ in the book. The important thing to take out of this is that given such a representation of a CFG, a turing machine can simulate the grammar.

Sipser also uses the notations A_{CFG} , E_{CFG} and EQ_{CFG} . These correspond, respectively, to the language composed of all pairs $(\langle G \rangle, w)$ where w is a string generated by the CFG G , the language of all strings $\langle G \rangle$ where the machine G generates a non-empty language, and the language of all pairs $(\langle G \rangle, \langle H \rangle)$ where G and H are CFG's that generate the same language.

1.2.2 Decidable Problems

Theorem 2 Given a CFG G and a string w , a turing machine can decide whether or not G generates w . Said formally, A_{CFG} is a decidable language.

Proof: If we have a grammar in Chomsky normal form, H , then it takes H exactly $2n - 1$ steps to generate an input of length n . (Unless $n = 0$, in which case it requires 1 step.) Thus, if G is in Chomsky-normal form, we can design a turing machine that will generate all derivations in G of length $2|w| - 1$. Since there are finitely many rules, this can be done in finite time and space. At that point, we only need to check all of our derivations against w ; if any one matches, we *accept*; if none do, we *reject*.

The problem is only slightly more complicated if G is not given to us in Chomsky normal form, as we have an *algorithm* to convert it. We need only modify our turing machine so that it first converts the grammar, then carries out the steps above.

Theorem 3 *Given a CFG G , a turing machine can decide whether G accepts any strings at all. Alternatively, E_{CFG} is decidable*

Proof: Intuitively, we want to “work our way backwards” from the terminals, and see if we can make our way to the top (start) node. In practice, we do exactly that. We begin by “marking” every terminal in the grammar. Then we scan through the grammar; if we find some variable that can be expanded into a string of symbols that have already been marked, we mark the variable. We then repeat this process until we can mark no more. If the start symbol is marked, we *accept*; if not, we *reject*.

Example: Suppose we wish to determine whether the CFG of problem 2.13 in the book is decidable. It has the rules:

$$\begin{array}{ll} S \rightarrow TT & S \rightarrow U \\ T \rightarrow 0T & T \rightarrow T0 \quad T \rightarrow 1 \\ U \rightarrow 0U00 & U \rightarrow 1T \end{array}$$

We begin by marking the terminals, 0 and 1. We then find the rule $T \rightarrow 1$, which allows us to mark T . Once T is marked, the rule $U \rightarrow 1T$ allows us to mark U and the rule $S \rightarrow TT$ allows us to mark S . So this particular CFG generates a non-empty language.

Corollary 4 *Given a two DFA's A and B , a turing machine can decide whether or not they recognize the same language - i.e. EQ_{DFA} is decidable.*

Proof: The set of languages recognized by DFA's is closed under union, intersection, and complementation. Furthermore, all of these operations can be carried out by our turing machine. We begin by constructing a new DFA, C , such that

$$L(C) = \left(L(A) \cap \overline{L(B)} \right) \cup \left(\overline{L(A)} \cap L(B) \right)$$

Thus $L(A) = L(B)$ if and only if $L(C)$ is empty. Since the set of languages generated by DFA's is a subset of the languages generated by CFG's, theorem 3 shows that we can decide whether or not $L(C)$ is empty, thus deciding whether $L(A) = L(B)$.

1.2.3 An Undecidable Problem

In corollary 4 we showed that the language EQ_{DFA} is decidable - that is, we can build a turing machine that decides if two DFA's generate the same language. By equivalence, we can do the same for NFA's or regular expressions. In order to prove that EQ_{DFA} is decidable, we used the fact that E_{DFA} is decidable. However, we cannot use this method to show that EQ_{CFG} is decidable, because the set of languages generated by CFG's is not closed under intersection or complementation. (See exercise 2.2 to see why not.) In fact, EQ_{CFG} *isn't* decidable, but we'll need to develop new proof techniques to show why.

2 The Halting Problem (Coming on Friday ...)