CS241 – Week 7 Tutorial

Languages: Context-Free Grammars

Spring 2017

Context-Free Grammars Review

A Context-Free Grammar $G = (N, T, P, S)$ consists of:

- N: the set of nonterminal symbols (or "nonterminals") (sometimes also called V).
- T: the alphabet, or the set of terminal symbols (or "terminals") (sometimes also called Σ).
- P: the set of production rules (or "productions") (sometimes also called R).
- S: the start nonterminal (or "start symbol").

We usually denote individual terminals by lowercase letters near the start of the alphabet $(a, b, c, \dots)$, nonterminals by uppercase letters $(A, B, C, \dots)$, strings of terminals by lowercase letters near the end of the alphabet (u, v, w, x, y, z), and (possibly empty) strings of terminals and nonterminals by Greek letters $(\alpha, \beta, \gamma, \dots)$. We usually simply represent a grammar by its set of production rules, with N and Σ implicitly defined by which terminals and nonterminals appear in the production rules. S is usually named S in the grammar, and is usually given the first production in the grammar as a reminder that it's the start nonterminal.

A production is a statement of the form $A \to \gamma$, where A is a nonterminal and γ is a string of terminals and nonterminals as stated above. A production means "anywhere in our string where we have an A, we can replace it with γ."

A language is context-free if and only if it is defined by some context-free grammar G. A word is in $L(G)$, the language specified by G, if we can take the string "S" and successively apply production rules until we arrive at w. For example, suppose G is the following grammar:

$$S \to 0S1$$
$$S \to 01$$

The word 000111 is in G since we can obtain 000111 by the following sequence of replacements:

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

Where $\Rightarrow$ means "the right hand side is obtained by applying a single production rule to the left hand side." This is called a *derivation* of 000111. Sometimes it's convenient to say that something can be derived after arbitrarily many (0 or more) production rules are applied, in which case we can use $\overset{*}{\Rightarrow}$, such as:

$$S \overset{*}{\Rightarrow} 000111$$

In general, $\Rightarrow$ is more convenient to show people why a word is in the language (it's the equivalent of "showing your work"), whereas $\overset{*}{\Rightarrow}$ is more convenient for stating facts or requirements (such as "this only holds for grammars where $S \overset{*}{\Rightarrow} \epsilon$").

While we can replace any nonterminal when using $\Rightarrow$, usually we prefer to consistently replace either the leftmost or rightmost nonterminal first at each step. These are called *left derivations* and *right derivations* (or leftmost and rightmost derivations) respectively. The individual steps (such as $00S11$ above) are called *left-sentential forms* (or *right-sentential forms*).

Context-Free Grammar Problems

Derivations

Let G be the following grammar:

$$S \to aAb$$
$$S \to bAa$$
$$A \to aSa$$
$$A \to bSb$$
$$A \to c$$

1. Show that the word *abacbbb* is in $L(G)$ by giving a derivation.
2. Which of the following are valid derivations of words in $L(G)$?
   (a) $S \Rightarrow aAb \Rightarrow aaAab \Rightarrow aacab$
   (b) $S \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aacbb$
   (c) $S \Rightarrow bAa \Rightarrow bca$
   (d) $A \Rightarrow aSa \Rightarrow aaAba \Rightarrow aacba$
   (e) $S \Rightarrow aaSab \Rightarrow aaaAbab \Rightarrow aaacbab$

Designing Grammars

1. Write context-free grammars for the following languages:
   (a) The language $L = \{\text{my}, \text{name}, \text{is}, \text{inigo}, \text{montoya}\}$.
   (b) The language of all words over $\Sigma = \{a, b, c\}$ not beginning with a b.
   (c) The language defined by the regular expression $(0|1)^*((00)^*1)^*010$.
   (d) The language $L = \{a^n b^n c^m d^m : n, m \in \mathbb{N}\}$, where a^n means "n copies of a in a row."
   (e) The language of palindromes over $\Sigma = \{a, b\}$. A *palindrome* is a word or phrase which is spelled the same forwards and backwards, for example "ABBA," "racecar," or "Able was I ere I saw Elba."
2. Argue that every regular language is context-free by describing how to convert any regular expression into a context-free grammar.

Parse Trees and Ambiguous Grammars

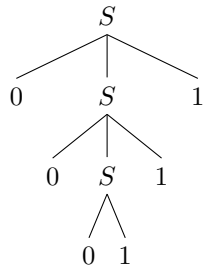
While a grammar and a derivation completely describes how a word is obtained from the grammar, derivations can be difficult to understand and work with. Derivations also put emphasis on things like the order in which we chose to expand two nonterminals, something we usually don't care about. To solve these problems, we usually choose to prove that words are in the language specified by a grammar by using a different construction, known as a *parse tree*.

In a parse tree, each internal node is a nonterminal, and each leaf node is a terminal or ε. The children of a nonterminal represent the right-hand side of the production it was expanded by. For example, recall the grammar and derivation from the definition of a derivation:

$$\begin{aligned} S &\to 0S1 \\ S &\to 01 \end{aligned}$$

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

We can instead prove that $000111 \in L(G)$ via the following parse tree:



Unlike with derivations, where there will usually be many valid derivations for each word, there is often only one parse tree for each word. Grammars with only one parse tree for each word are called *unambiguous*, while those with multiple parse trees for a word are called *ambiguous*. To prove a grammar is ambiguous we only need to find a word which has two parse trees and show them. Proving a grammar is unambiguous is more challenging and is beyond the scope of this course.

Parse Tree Problems

Let G be the following grammar:

$$\begin{aligned} S &\to AB & S &\to CD \\ A &\to aAb & A &\to \epsilon \\ B &\to cB & B &\to \epsilon \\ C &\to aC & C &\to \epsilon \\ D &\to bDc & D &\to \epsilon \end{aligned}$$

1. Find parse trees for the following words:
   (a) *abcccc*
   (b) *aabc*
   (c) *bbbbcccc*
2. Prove that G is ambiguous by finding two parse trees for the word *aaabbbccc*.