CS 4510: Automata and Complexity 09/15/2022

Lecture 8: Context-Free Languages (CFLs)

Lecturer: Zvi Galil *Author: Austin Peng*

Contents

1 Context-Free Grammars (CFGs)

- A grammar consists of collection of **substitution rules**, called **productions**.
- Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow
- The symbol is called a **variable**.
- The string consists of variables and other symbols called **terminals**.
- The variables are often represented by capital letters. The terminals are often represented by lowercase letters, numbers, or special symbols.
- One variable is designated as the **start variable**.

You use a grammar to describe a language by generating each string of that language in the following manner.

1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

Example 1. Consider the grammar G_1:

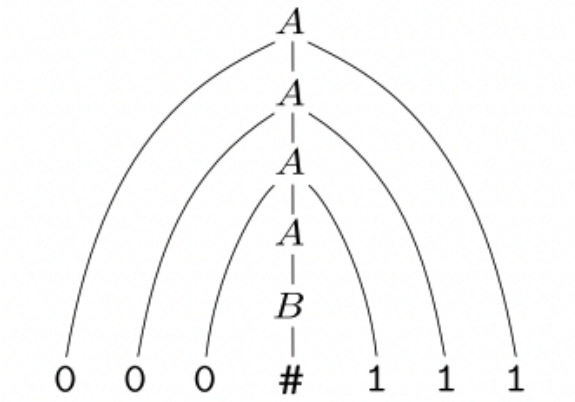
$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

- $0, 1, \#$ are terminals
- let A be the start symbol
- $A \to 0A1 \to 00A11 \to 000B111 \to 000\#111$ is a derivation
- $000\#111$ is a string of terminals

We can say that G_1 generates $0^3\#1^3$.

Define $L(G)$ to be all strings generated by G. We can prove $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$. Then $L(G_1)$ is a CFL because it is generated by a CFG. We say that $S = 000\#111 \in L(G_1)$.

The following is a parse tree that describes the derivation of S.



Pulling all the leaves from the tree (shown by thin lines), we can see the string that is generated. Every word that is generated/derived from the start state has a parse tree. Grammars and parse tress come from linguistics.

For example, English has a grammar and can be parsed.

1.1 Formal Definition Of A Context-Free Grammar

Definition 1. A **context-free grammar** is a 4-tuple (V, Σ, R, S), where:

- V is a finite set called the **variables**
- Σ is a finite set, disjoint from V, called the **terminals**
- R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, shown below

$$\text{variable} \rightarrow \text{string}$$

- $S \in V$ is the start variable

If u, v, w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv yields uwv, written $uAv \Rightarrow uwv$. Say that u derives v, written $u \Rightarrow v$, if $u = v$ or if a sequence $u_1, u_2, ..., u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow ... \Rightarrow u_k \Rightarrow v$$

Example 2. Consider the grammar $G_2 = (\{S\}, \{a, b\}R, S)$. The set of rules R is

$$S \rightarrow aSb \mid SS \mid \varepsilon$$

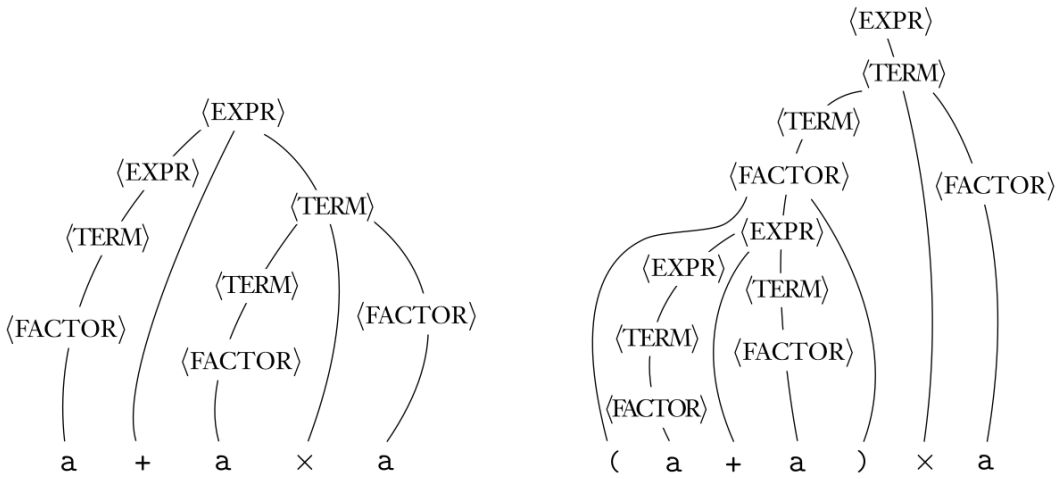
This grammar generates strings such as *abab*, *aaabbb*, and *aabaab*. Think of a as '(' and b as ')'. Viewed in this way, $L(G_3)$ is the language of all strings of properly nested (balanced) parentheses. Note that the language may contain the empty string ε.

Example 3. Consider the grammar $G_3 = (V, \Sigma, R, \langle EXPR \rangle)$.

V is $\{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$ and Σ is $\{a, +, \times, (,)\}$

$$\begin{aligned} \langle EXPR \rangle &\to \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle \\ \langle TERM \rangle &\to \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle \\ \langle FACTOR \rangle &\to (\langle EXPR \rangle) \mid a \end{aligned}$$

The two strings $a + a \times a$ and $(a + a) \times a$ are both generated with grammar G_3. The parse trees are shown below:



1.2 Designing Context-Free Grammars

Many CFLs are unions of simpler CFLs.

Example 4. To get a grammar for the language $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$, first construct the grammar.

$S_1 \to 0S_1 1 \mid \varepsilon$ for language $\{0^n 1^n \mid n \geq 0\}$ $S_2 \to 1S_2 0 \mid \varepsilon$ for language $\{1^n 0^n \mid n \geq 0\}$

Then add the rule $S \to S_1 \mid S_2$ to give the grammar:

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to 0S_1 1 \mid \varepsilon \\ S_2 &\to 1S_2 0 \mid \varepsilon \end{aligned}$$

Constructing a CFG for a language that happens to be regular is easy if you can first construct a DFA for that language. You can convert any DFA into an equivalent CFG by:

1. Make a variable R_i for each state q_i of the DFA.
2. Add the rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is the transition in the DFA.
3. Add the rule $R_i \to \varepsilon$ if q_i is an accepting state of the DFA.
4. Make R_0 the start variable of the grammar, where q_0 is the start state of the machine

Certain CFLs contain strings with two substrings that are "linked" in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring.

Example 5. Consider the language $\{0^n1^n \mid n \geq 0\}$.

The machine would need to remember the number of 0s in order to verify that it equals the number of 1s. You can construct a CFG to handle this situation by using a rule of the form $R \to uRv$, which generates strings wherein the portion containing the u's correspond to the portion containing the v's.

In more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures. An example is Example 3, where the grammar generates arithmetic expressions. Any time a symbol a appears, an entire parenthesized expression might appear recursively instead. To do this, put the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear.

1.3 Nondeterminism With CFGs

CFGs are in some sense nondeterministic because at each step, we can choose:

1. which rule to apply
2. which variable to apply the rule

Example 6. Consider the string $0AB1A0$.

We can eliminate nondeterminism for (2) with leftmost derivation (applying the rule to the leftmost variable). However we cannot entirely eliminate nondeterminism because there can be multiple choices for the rule from a variable such as $A \to w_1 \mid w_2 \mid w_3 \mid w_4$.

1.4 Ambiguity

If the grammar generates the same string in several ways, we say that the string is derived **ambiguously**. If a grammar generates some string ambiguously, we say that the grammar is **ambiguous**.

More formally, if $w \in L(G)$, then $S \overset{*}{\Rightarrow} w$. Sometimes $S \overset{*}{\Rightarrow} w$ in more than 1 way, and if this happens with G, then G is ambiguous.

Example 7. Let $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$.

It can be shown that L is generated by a CFG, and L is inherently ambiguous. Consider the following grammar G_5 for L:

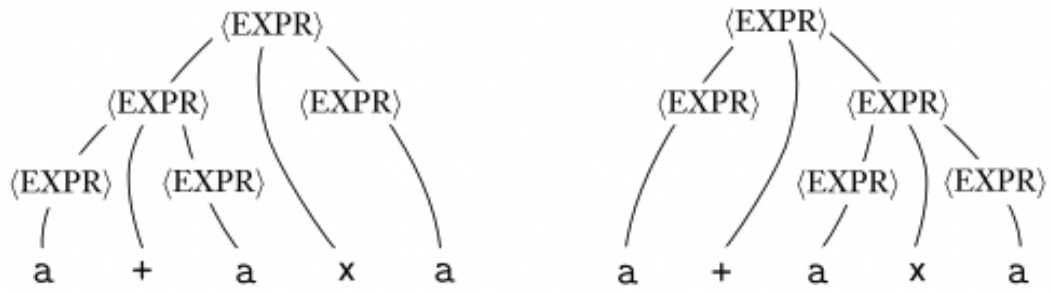
$$\begin{aligned} S &\to S_1 C \mid A S_2 \\ S_1 &\to a S_1 b \mid \varepsilon \\ S_2 &\to b S_2 c \mid \varepsilon \\ A &\to aA \mid \varepsilon \\ C &\to Cc \mid \varepsilon \end{aligned}$$

A string such as *aabbcc* can be generated through multiple different paths.

Example 8. Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$.

$$\langle EXPR \rangle \to \langle EXPR \rangle + \langle EXPR \rangle \mid \langle EXPR \rangle \times \langle EXPR \rangle \mid (\langle EXPR \rangle) \mid a$$

The following is two parse trees for the string $a + a \times a$ in grammar G_4.



Note that grammar G_4 does not capture the usual precedence relations ($\times$ before $+$).