St. Francis Xavier University
Department of Computer Science

CSCI 356: Theory of Computing
Lecture 2: Context-Free Languages
Fall 2023

1 Context-Free Grammars

Recall that, in our discussion on regular languages, we introduced the notion of a regular expression. This expression essentially performed a kind of pattern matching to accept text in a certain form and reject all other text not of that form.

We can take this idea of matching patterns in text and modify it to work not just for individual words within the text, but for the structure and composition of the entire text itself. This ability comes in the form of *grammars*, which provide us with a set of *rules* that we can follow to produce words that belong to a certain language. If a grammar produces all and only those words belonging to a certain language, then we say the grammar *generates* that language.

The idea of using grammars with languages is nothing new; linguists have been using grammars to study natural languages for centuries, dating as far back as the fourth century BCE with the work of the Indian grammarian Pāṇini. Only with the advent of computer science itself has the notion of grammars been applied to formal languages and programming languages, starting with the work of American linguist Noam Chomsky in the 1950s.

If you look at the specification manual for any programming language, you will likely find tucked away somewhere in the documentation a grammar for that language. This grammar, which could number into the tens of pages, describes precisely what the structure of a program written in that language should look like. In fact, this grammar is exactly what the compiler relies on to check for syntax errors in your program!

As an example, let's consider one small excerpt from the grammar given in the third edition of the *Java Language Specification*:

```
Statement:
    Block
    assert Expression [ : Expression] ;
    if ParExpression Statement [else Statement]
    for ( ForControl ) Statement
    while ParExpression Statement
    do Statement while ParExpression ;
    try Block ( Catches | [Catches] finally Block )
    switch ParExpression { SwitchBlockStatementGroups }
    synchronized ParExpression Block
    return [Expression] ;
    throw Expression ;
    break [Identifier]
    continue [Identifier]
    ;
    StatementExpression ;
    Identifier   :   Statement
```

This part of the Java grammar checks statement blocks such as assignments, if-else blocks, for loops, and so on. All of the words written with an `Uppercase` letter or written in `CamelCase` correspond to rules, and all of the words written in `lowercase` correspond to language keywords. For example, the `if` rule on the fourth line checks that every if-else block in a program conforms to the syntax that the compiler expects: the block begins with the keyword `if` together with some parenthesized expression, followed by some statement or sequence of instructions, and ending with an optional `else` block.

1.1 Definition

The Java grammar is an example of a *context-free grammar*. Such a grammar consists of a set of rules that we can use, in this instance, to generate valid programs in Java. These rules take on a very general form: observe, for example, that we can replace the word `Statement` by a number of combinations of keywords and other rules, as specified by each line of the grammar underneath the `Statement` label. This is where the name "context-free" comes from: we don't care about the context surrounding a particular bit of text when we replace that text with something else.

Before we look at some more examples, let's formalize the notion of a context-free grammar. To construct a grammar, we need only four elements.

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

- V is a finite set of elements called *nonterminal symbols*;
- Σ is a finite set of elements called *terminal symbols*, where $\Sigma \cap V = \emptyset$;
- R is a finite set of *rules*, where each rule consists of a nonterminal on the left-hand side and a combination of nonterminals and terminals on the right-hand side; and
- $S \in V$ is the *start nonterminal*.

In a context-free grammar, the set of nonterminal symbols V correspond to parts of a word that we have yet to "fill in" with terminal symbols from Σ. The set of rules R tell us how we can perform this "filling in". If we have a rule of the form $A \to \alpha$, then we can replace any instance of the symbol A in our word with whatever symbols make up α. The start nonterminal S is self-explanatory; it is the first thing in our word that we "fill in".

Returning to our Java grammar example, we can see that (for example) some of the nonterminals in the grammar include `Statement`, `Block`, `Identifier`, and `ParExpression`, while some of the terminals include `if`, `while`, `for`, and `;` (semicolon).

Importantly, we have in our definition of a context-free grammar that $\Sigma \cap V = \emptyset$; that is, the set of terminals and the set of nonterminals must be disjoint. This is to prevent the grammar from confusing terminals and nonterminals, and this is exactly why the Java language designers used uppercase letters in their nonterminals and lowercase letters in their terminals.

1.2 Language of a Context-Free Grammar

The sequence of rule applications we follow beginning with the start nonterminal S and ending with a completed word containing symbols from Σ is called a *derivation*. Each word of the form $(V \cup \Sigma)^*$ produced during a derivation is sometimes referred to as a *sentential form*.

For any nonterminal A and terminals u, w, and v, if we have a rule $A \to w$ in our grammar and some step of our derivation takes us from uAv to uwv, then we say that uAv *yields* uwv and we write $uAv \Rightarrow uwv$. We can represent a sequence of "yields" relations using similar notation; given words x and y, if $x = y$ or if there exists a sequence $x_1, x_2, \ldots, x_k$ where $k \geq 0$ such that

$$x \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_k \Rightarrow y,$$

then we write $x \Rightarrow^* y$. This is very similar to the Kleene star notation, where the star indicates zero or more "yields" relations taking us from x to y.

With this, we can define the *language of a grammar* G over an alphabet Σ by $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$. In other terms, the language of a grammar contains all words that can be derived by that grammar beginning with the start nonterminal S.

Example 2. Consider the context-free grammar where $V = \{S, A\}$, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and R contains two rules:

$$S \to \mathtt{a}A\mathtt{b}$$
$$A \to \mathtt{a}A\mathtt{b} \mid \epsilon$$

Using this context-free grammar, we can generate words like

$$S \Rightarrow \mathtt{a}A\mathtt{b} \Rightarrow \mathtt{a}\,\epsilon\,\mathtt{b} = \mathtt{ab},$$
$$S \Rightarrow \mathtt{a}A\mathtt{b} \Rightarrow \mathtt{a}\,\mathtt{a}A\mathtt{b}\,\mathtt{b} \Rightarrow \mathtt{aa}\,\epsilon\,\mathtt{bb} = \mathtt{aabb}, \text{ and}$$
$$S \Rightarrow \mathtt{a}A\mathtt{b} \Rightarrow \mathtt{a}\,\mathtt{a}A\mathtt{b}\,\mathtt{b} \Rightarrow \mathtt{aa}\,\mathtt{a}A\mathtt{b}\,\mathtt{bb} \Rightarrow \mathtt{aaa}\,\epsilon\,\mathtt{bbb} = \mathtt{aaabbb},$$

and so on. For each step, the highlighted symbols indicate which symbols were added at that step. We get things started by replacing the S nonterminal by $\mathtt{a}A\mathtt{b}$, and from there we may replace the A nonterminal as many times as we like.

This context-free grammar generates all words over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ where the number of $\mathtt{a}$s is equal to the number of $\mathtt{b}$s, where there is at least one $\mathtt{a}$ and one $\mathtt{b}$, and where all $\mathtt{a}$s come before any $\mathtt{b}$s in the word. Thus, the language of this grammar is $L_{\mathtt{a}=\mathtt{b}} = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 1\}$.

Observe that the rule A in Example 2 included a vertical bar. This is simply a shorthand for writing multiple rules where each rule contains A on the left-hand side. Writing $A \to \mathtt{a}A\mathtt{b} \mid \epsilon$ is therefore equivalent to writing

$$A \to \mathtt{a}A\mathtt{b}$$
$$A \to \epsilon$$

There are very few limitations we must abide by when we write rules for a context-free grammar. All we need to ensure is that the left-hand side of each rule consists of exactly one nonterminal by itself. The right-hand side of each rule can contain any combination of terminals and nonterminals, including the empty word ϵ.

Example 3. Consider the context-free grammar where $V = \{S\}$, $\Sigma = \{\mathtt{(}, \mathtt{)}\}$, and R contains one rule:

$$S \to \mathtt{(}S\mathtt{)} \mid SS \mid \epsilon$$

This rule allows us to surround an occurrence of S with parentheses, to "duplicate" an occurrence of S, or to replace some occurrence of S with ϵ, effectively removing that occurrence of S from the derivation.

Using this context-free grammar, we can generate a word like

$$S \Rightarrow SS \Rightarrow \mathtt{(}S\mathtt{)}S \Rightarrow \mathtt{(}\,\mathtt{(}S\mathtt{)}\,\mathtt{)}S \Rightarrow \mathtt{((}\,\epsilon\,\mathtt{))}S \Rightarrow \mathtt{(())}S \Rightarrow \mathtt{(())}\,\mathtt{(}S\mathtt{)} \Rightarrow \mathtt{(())(}\,\epsilon\,\mathtt{)} = \mathtt{(())()}.$$

Again, the highlighted symbols indicate which symbols were added at a given step.

This context-free grammar generates all words over the alphabet $\Sigma = \{\mathtt{(}, \mathtt{)}\}$ where each word contains *balanced parentheses*: every opening parenthesis is matched by a closing parenthesis, and each pair of parentheses is correctly nested. We can express the language of the grammar as

$$L_{()} = \{w \in \{\mathtt{(}, \mathtt{)}\}^* \mid \text{all prefixes of } w \text{ contain no more } \mathtt{)}\text{s than } \mathtt{(}\text{s, and } |w|_{\mathtt{(}} = |w|_{\mathtt{)}}\}.^1$$

[1] The language of balanced parentheses is also known as the *Dyck language*.