Context-Free Grammars + Languages

Alyssa Lytle

Fall 2025

1 Introduction

For this lecture we're finally going to stray *somewhat* from automata and regular languages. We're going to zoom out, in a sense, and talk about Context-Free Grammars (CFGs) and Languages (CFLs).

Just as regular expressions describe regular languages, **context-free grammars describe context-free languages**. However, context free languages can be even more powerful. In fact, the "class" of Context-Free Languages *contains* the set of Regular Languages.

CFLs are good for describing infinite sets of strings in a finite way.[Koz07] They are commonly used for describing the syntax programming languages.

Example 1

Here is an example of a Context-Free Grammar in Backus-Naur Form [Koz07]:

$$
\begin{aligned}
\texttt{<stmt>} &::= \texttt{<if-stmt>} \mid \texttt{<while-stmt>} \mid \texttt{<begin-stmt>} \mid \texttt{<assg-stmt>} \\
\texttt{<if-stmt>} &::= \textbf{if } \texttt{<bool-expr>} \textbf{ then } \texttt{<stmt>} \textbf{ else } \texttt{<stmt>} \\
\texttt{<while-stmt>} &::= \textbf{while } \texttt{<bool-expr>} \textbf{ do } \texttt{<stmt>} \\
\texttt{<begin-stmt>} &::= \textbf{begin } \texttt{<stmt-list>} \textbf{ end} \\
\texttt{<stmt-list>} &::= \texttt{<stmt>} \mid \texttt{<stmt>};\texttt{<stmt-list>} \\
\texttt{<assg-stmt>} &::= \texttt{<var>} := \texttt{<arith-expr>} \\
\texttt{<bool-expr>} &::= \texttt{<arith-expr>}\texttt{<compare-op>}\texttt{<arith-expr>} \\
\texttt{<compare-op>} &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
\texttt{<arith-expr>} &::= \texttt{<var>} \mid \texttt{<const>} \mid (\texttt{<arith-expr>}\texttt{<arith-op>}\texttt{<arith-expr>}) \\
\texttt{<arith-op>} &::= + \mid - \mid * \mid / \\
\texttt{<const>} &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
\texttt{<var>} &::= a \mid b \mid c \mid \ldots \mid x \mid y \mid z
\end{aligned}
$$

Definition 1: Context Free Grammar

A context-free grammar (CFG) is a quadruple

$$G = (N, \Sigma, P, S)$$

where

- N is a finite set (the *nonterminal* symbols or *variables*)
- Σ is a finite set (the *terminal* symbols) dijoint from N
- P is a finite subset of $N \times (N \cup \Sigma)^*$ (the *productions* or *rules*), and
- $S \in N$ (the *start symbol*) [Koz07]

Common Conventions Typically nonterminals are denoted with capital letters (e.g. $A, B, \ldots$) and terminals are denoted with lowercase letters (e.g. $a, b, \ldots$). Strings in $(N \cup \Sigma)^*$ are often denoted using greek letters (e.g. $\alpha, \beta, \gamma, \ldots$).

Think of productions kind of like transitions. Instead of a tuple representation like (A, α), they often have an arrow representation $A \to \alpha$.

To denote a set of productions with the same left-hand side, instead of listing them
$A \to \alpha_1$, $A \to \alpha_2$, $A \to \alpha_3$
You use the abbreviation
$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$

Example 2

The nonregular set $\{a^n b^n | n \geq 0\}$ can be represented as a CFL
$S \to aSb \mid \epsilon$

More specifically, in quadruple form: $G = (N, \Sigma, P, S)$, where

- $N = \{S\}$
- $\Sigma = \{a, b\}$
- $P = \{S \to aSb, S \to \epsilon\}$

Here's how you would derive the string a^3b^3 or $aaabbb$:

S	Start Symbol
aSb	Apply $S \to aSb$ from P
$aaSbb$	Apply $S \to aSb$ from P
$aaaSbbb$	Apply $S \to aSb$ from P
$aaabbb$	Apply $S \to \epsilon$ from P

The more common way to write this is:

$$S \xrightarrow[G]{1} aSb \xrightarrow[G]{1} aaSbb \xrightarrow[G]{1} aaaSbbb \xrightarrow[G]{1} aaabbb$$

Or $S \xrightarrow[G]{4} aaabbb$

In English you'd say "This string is derivable from the start symbol in 4 steps."

Definition 2: Sentential Form + Sentence

A string in $(N \cup \Sigma)^*$ derivable from the start symbol S is called a *sentential form*. A sentential form is called a *sentence* if it consists only of terminal symbols. [Koz07] (In other words, a sentence would be in Σ^*.)

Derivable Strings As we showed in the above example,

- $\alpha \xrightarrow[G]{1} \beta$ if β can be derived from α over one step in the grammar G.
- $\alpha \xrightarrow[G]{n} \beta$ if β can be derived from α over n steps in the grammar G.

Moreover

- $\alpha \xrightarrow[G]{*} \beta$ if $\alpha \xrightarrow[G]{n} \beta$ for some $n \geq 0$

  This allows us to define the language of a grammar in the following way:

  Definition 3

  The *language of the grammar* G is $\{w \in \Sigma^* \mid S \xrightarrow[G]{*} w\}$

1.1 Converting a DFA into a CFG

There is an easy step-by-step way to convert a DFA $M = (Q, \Sigma, \delta, s, F)$ into a CFG:

- For each $q_i \in Q$, make a nonterminal R_i
- For all transitions $\delta(q_i, x) = q_j$, $(x \in \Sigma)$, add the rule $R_i \to xR_j$
- If q_i is an accept state, add the rule $R_i \to \epsilon$
- If q_0 is the start state of the machine, make R_0 the start variable.

References

[Koz07] Dexter C Kozen. *Automata and computability*. Springer Science & Business Media, 2007.