CS301: Theory of Computation Sections 2.1 & 2.3 - Context-Free Grammars/Languages

Daniel White

Gettysburg College
Computer Science

Spring 2024

Chapter 1 introduced two different, though equivalent, methods of describing languages: *regular expressions* and *finite automata*.

Many languages can be described in this way, but some simple languages cannot since memory is a bottleneck.

Chapter 2 presents *context-free grammars* and their equivalent machine counterpart, *pushdown automata*.

The set of languages recognized by this new machinery — the context-free languages — will subsume the regular languages.

The following is an example of a context-free grammar, G_1.

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

- Grammars consist of a collection of **substitution rules**, also known as **productions**.
- Each rule is comprised of a **variable** and a string, separated by a right-facing arrow.
- The strings consist of variables and **terminals**, the latter of which are analogous to the members of an alphabet.

The grammar G_1 generates all strings of the form $0^n\#1^n$ through the following **derivation**:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow \cdots \Rightarrow 0^nB1^n \Rightarrow 0^n\#1^n$$

Definition (pg. 103)

All strings generated by a context-free grammar constitute the **language of the grammar**. We write $L(G)$ for the language of the grammar G. Any language that can be generated by some context-free grammar is called a **context-free language (CFL)**.

exercise: Show that there is a CFL which is not regular.

Definition 2.2 (pg. 104)

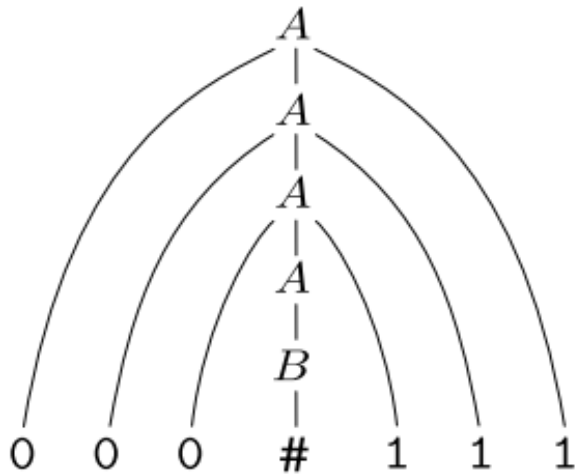
A **context-free grammar** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

exercise: Create a context-free grammar G_0 that generates all balanced parenthetical expressions.

homework: 2.3 & 2.4ad

One can illustrate the derivation of a string using a **parse tree**. For example, the parse tree of 000#111 using G_1 is given below.



exercise: Generate ()(()()) using G_0 and show the parse tree.

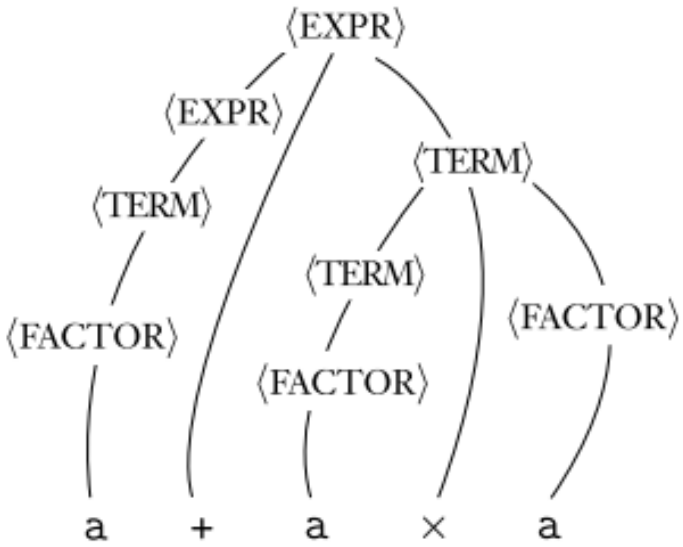
Consider the grammar $G_4 = (V, \Sigma, R, \langle \texttt{EXPR} \rangle)$, where

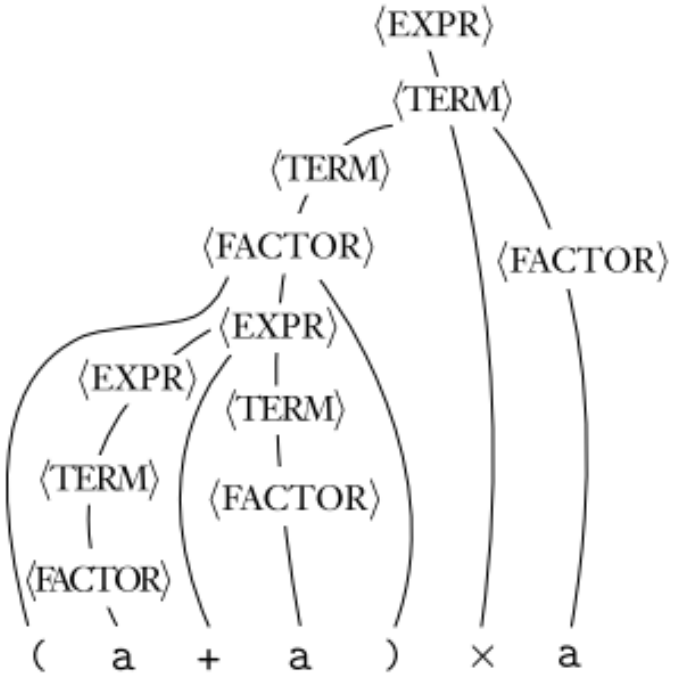
$$V = \{\langle \texttt{EXPR} \rangle, \langle \texttt{TERM} \rangle, \langle \texttt{FACTOR} \rangle\} \text{ and}$$
$$\Sigma = \{\texttt{a}, +, \times, (,)\},$$

and the rules are:

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

exercise: Generate the strings $\texttt{a} + \texttt{a} \times \texttt{a}$ and $(\texttt{a} + \texttt{a}) \times \texttt{a}$ using G_4 by showing their parse trees.


⟨EXPR⟩
⟨EXPR⟩
⟨TERM⟩
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
⟨FACTOR⟩
⟨FACTOR⟩
a + a × a


⟨EXPR⟩
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
⟨FACTOR⟩
⟨EXPR⟩
⟨EXPR⟩
⟨TERM⟩
⟨TERM⟩
⟨FACTOR⟩
⟨FACTOR⟩
(a + a) × a

homework: 2.1

Grammars may generate the same string in several different ways. Such strings will have multiple parse trees, and thus multiple meanings.

This may be undesirable for applications such as programming languages.

exercise: Consider the following grammar, G_5, and show that it generates the string $a + a \times a$ ambiguously.

$$E \to E + E \mid E \times E \mid (E) \mid a$$

exercise: Can you argue that either of the parse trees generated above are themselves ambiguous? Can you argue against that?

To "generate a string ambiguously", we mean that the string has two different parse trees, not two different derivations.

To concentrate on structure only, we consider **leftmost derivations**, which requires that we replace the leftmost variable at each step of the derivation.

Definition 2.7 (pg. 108)

A string w is derived **ambiguously** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is **ambiguous** if it generates some string ambiguously.

exercise: Let G be the following grammar.

$$S \to A \mid I \mid E$$
$$I \to \texttt{if condition then } S$$
$$E \to \texttt{if condition then } S \texttt{ else } S$$
$$A \to \texttt{a} := 1$$

G is a natural-looking grammar for a fragment of a programming language, but G is ambiguous.

- Show that G is ambiguous.
- Give a new unambiguous grammar for the same language.

homework: 2.8

Despite the claim (to be proved) that context-free languages properly contain the regular languages, there are still many languages left uncaptured by context-free grammars.

We introduced the pumping lemma previously as a tool for showing a language is not regular. Fortunately, a version of this exists in the context-free setting.

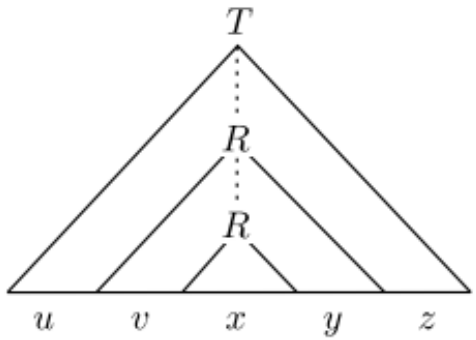
Theorem 2.34 (pg. 125)

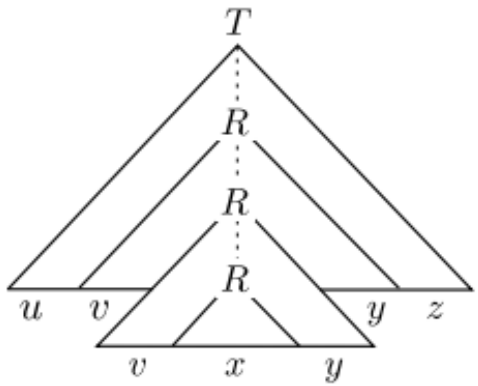
If A is a CFL, then there is a number p, the **pumping length**, where, if $s \in A$ has length at least p, then s may be divided into $s = uvxyz$ satisfying the conditions

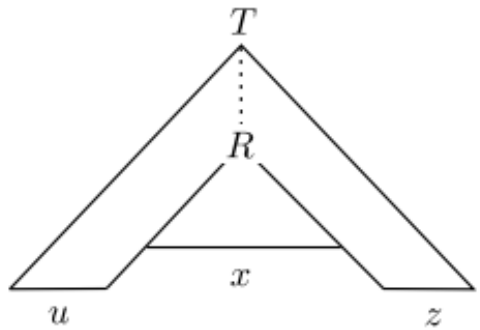
1. for each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

Proof idea. Let G be a CFG generating CFL A and let $s \in A$ be sufficiently long. Since s is very long, we are guaranteed that a path down a parse tree of s contains a repeated variable, R ...

exercise: Finish this proof idea. *(pg. 126)*


T
R
R
u
v
x
y
z


T
R
R
R
u
v
y
z
v
x
y


T
R
x
u
z

exercise: Use the pumping lemma to show that the language $B = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.

exercise: Use the pumping lemma to show that the language $D = \{ww \mid w \in \{0,1\}^*\}$ is not context-free.

homework: 2.30bc & example 2.37