Derivation Trees and Regular Languages

We consider special derivations and ways to represent derivations. Then we connect CFGs and regular languages.

Leftmost & Rightmost Derivations

Strings α ***yields*** string β, written $\alpha \stackrel{*}{\Longrightarrow} \beta$, if it is possible to get from α to β using the productions. A ***derivation*** of β is the sequence of steps that gets to β. A ***leftmost*** derivation is where at each stage one replaces the leftmost variable. A ***rightmost derivation*** is defined similarly.

Derivation Trees

In a ***derivation tree***, the root is the start variable, all internal nodes are labeled with variables, while all leaves are labeled with terminals. The children of an internal node are labeled from left to right with the right-hand side of the production used.

Example

Recall the CFG for equal `0`'s and `1`'s:

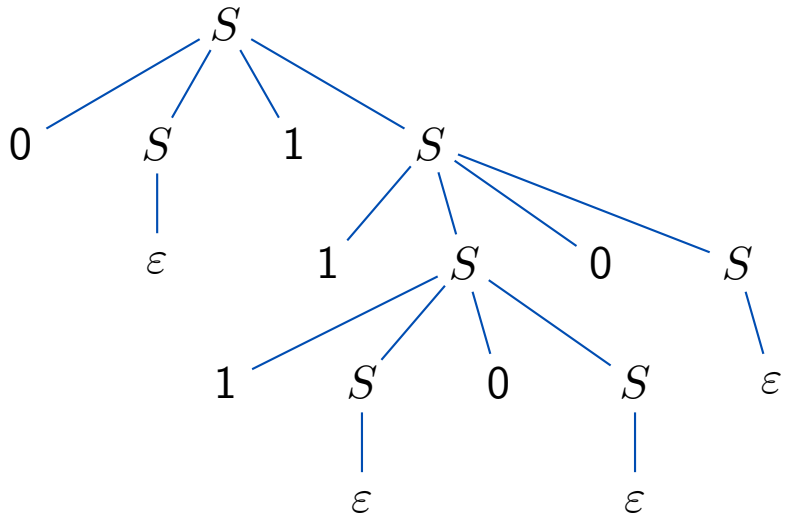
$$S \to \texttt{0}S\texttt{1}S \mid \texttt{1}S\texttt{0}S \mid \varepsilon$$

The derivation for `011100` given earlier was left-most:

$$S \Longrightarrow \texttt{0}S\texttt{1}S \Longrightarrow \texttt{01}S \Longrightarrow \texttt{011}S\texttt{0}S \Longrightarrow \texttt{0111}S\texttt{0}S\texttt{0}S$$
$$\Longrightarrow \texttt{01110}S\texttt{0}S \Longrightarrow \texttt{011100}S \Longrightarrow \texttt{011100}$$

Example Derivation Tree

Here is derivation tree for `011100` in the above grammar



Ambiguous Grammars

A grammar is ***unambiguous*** if there is a unique leftmost derivation for each string in the language. Equivalently, for each string there is a unique derivation tree.

For example, our grammar for equality is ambiguous:

$$S \to \texttt{0}S\texttt{1}S \mid \texttt{1}S\texttt{0}S \mid \varepsilon$$

(The string `0101` has two derivation trees.)

Grammars and Compilers

Grammars are used in compilers. A compiler checks that the input file is valid by essentially finding the derivation tree. This derivation tree is then used in finding code for the file or expression. If grammar is unambiguous, the derivation tree is always unique.

Grammar for Arithmetic

The language of arithmetical expressions using only multiplication and addition can be described as the following CFG with start variable E:

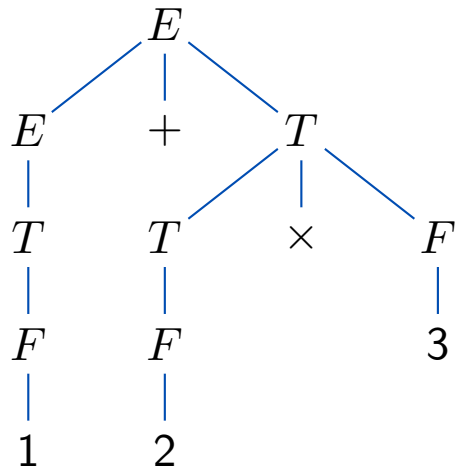
$$E \to E\texttt{+}T \mid T$$
$$T \to T\times F \mid F$$
$$F \to \texttt{(}\,E\,\texttt{)} \mid number$$

(Think of Expression, Factor and Term.)

This generates expressions such as `1+(3+2)`$\times$`5` and `1+2`$\times$`3`.

Derivation Tree

The derivation tree for 1+2×3:



Note that derivation tree automatically knows that multiplication takes precedence over addition.

Regular Languages Revisited

Theorem. *Every regular language is generated by a context-free grammar.*

One way to prove this is to provide algorithm to convert an RE to a CFG.

This can be achieved recursively. For example, if overall language is union of two pieces, one can write $S \to A \mid B$; and if the concatenation of two pieces, one can write $S \to CD$. If overall language is the star of a piece, say generated by E, then one can write $S \to ES \mid \varepsilon$.

Example CFG for RE

Consider RE $(\texttt{11}+\texttt{00})^*\texttt{11}$. At top level, it is *concatenation* of two pieces; the first piece is *or* of two parts. So:

$$S \to T\,U$$
$$U \to \texttt{11}$$
$$T \to T\,V \mid \varepsilon$$
$$V \to \texttt{00} \mid \texttt{11}$$

Summary

Each string in the language has a leftmost derivation and a derivation tree. If these are unique for all strings, then the grammar is called unambiguous.