CS 241 — Lecture 13

Bartosz Antczak *Instructor: Kevin Lanctot* *February 14, 2017*

Recall: Context-Free Grammar

Since DFA's, NFA's, and regular expressions are finite, they won't suffice in reading the syntax. We'll need something more powerful, and from that stems **context-free grammar**, which is an approach to interpreting a sequence of tokens and determining if the syntax of a program is correct.

13.1 Examples of Context-free Grammar

Context-free Grammar (CFG) involves a set of rules that we can use to manipulate words found in a language.

13.1.1 Example 1

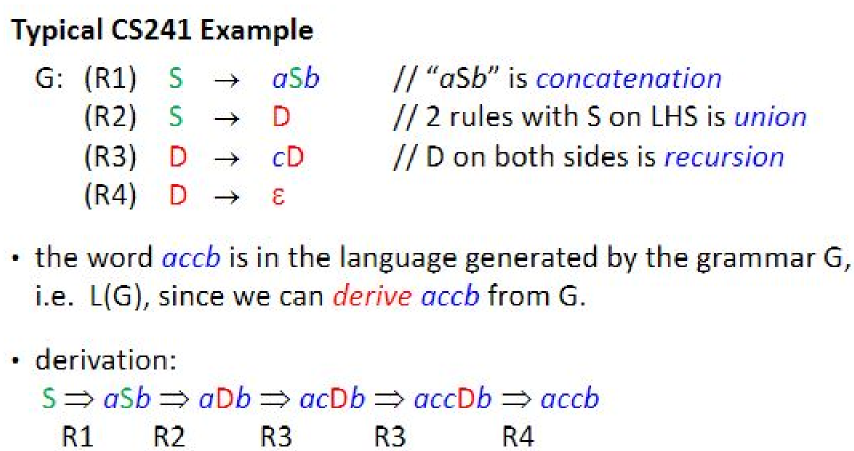

Figure 13.1: We can derive the word "accb" using this particular CFG. This means that "accb" is a valid word in the language. Courtesy of Prof. Lanctot's slides.

13.1.2 Example 2

Consider the language $\Sigma = \{a, b\}$ whose language consists of words that start with one 'a', followed by an arbitrary amount of 'b's (i.e., $\{a, ab, abb, abbb, \cdots\}$). The rules of the CFG that reads this language are:

- **(R1)**: $S \rightarrow aB$
- **(R2)**: $B \rightarrow bB$
- **(R3)**: $B \rightarrow \varepsilon$

Say we want to derive abb, we would do the following:

$$S \rightarrow_{(R1)} aB \rightarrow_{(R2)} abB \rightarrow_{(R2)} abbB \rightarrow_{(R3)} abb$$

13.1.3 Example 3

Let's create a CFG that access accepts words with balanced parentheses. For example, valid words are

$$\varepsilon,\ (),\ (()),\ ()(),\ (()()),\cdots$$

The rules in the grammar are:

- **(R1)**: $S \to (S)$
- **(R2)**: $S \to S\ S$
- **(R3)**: $S \to \varepsilon$

From these, let's derive some words:

1. Derive (()):
$$S \to_{(R1)} (S) \to_{(R1)} ((S)) \to_{(R3)} (())$$

2. Derive (()()):

$$S \to_{(R1)} (S) \to_{(R2)} (SS) \to_{(R1)} ((S)S) \to_{(R3)} (()S) \to_{(R1)} (()(S)) \to_{(R3)} (()())$$

13.1.4 Example 4

For $\Sigma = \{a, b\}$, a CFG that contains an even number of a's:

- **(R1)**: $S \to bS$
- **(R2)**: $S \to Sb$
- **(R3)**: $S \to aSa$ (a's are generated in pairs, which grantees that we have an even number)
- **(R4)**: $S \to \varepsilon$

13.1.5 Example 5

Binary Expressions

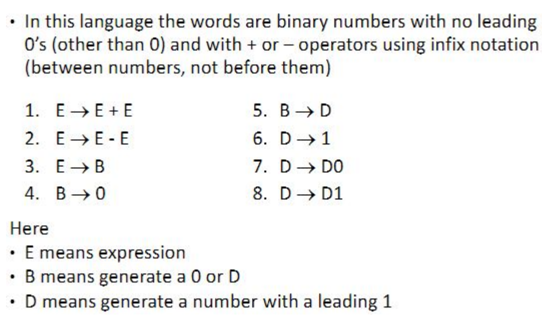

Figure 13.2: Courtesy of Prof. Lanctot's slides.

13.2 Parse Trees

Also called a *derivation tree*. It visualizes an entire derivation at once. The tree is structured such that:

- The **internal nodes** are the non-terminals (e.g., E, B, D)
- The **root** of the tree is the start symbol (e.g., E)
- The **children** of a node are given by derivation rules
- The **leaf nodes** are the terminals and show their value (e.g., $1, 0, +1$)

Ambiguous Grammar

Because grammar can be ambiguous, we can have multiple parse trees for the same expression.
The way we'll be processing order in a parse tree is with a **post-order** with a **depth first** traversal.

Unambiguous Grammar

To make CFG unambiguous, we must change our rules:

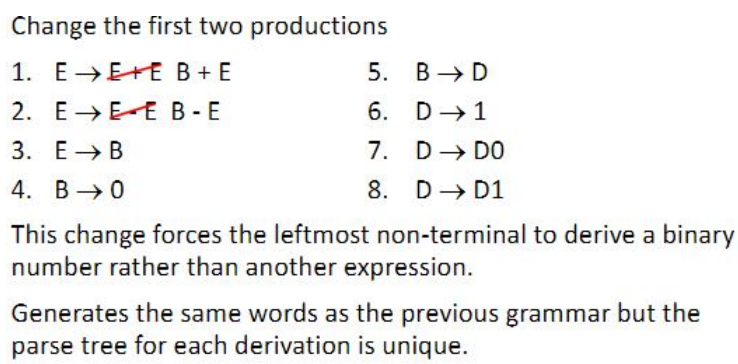

Figure 13.3: Courtesy of Prof. Lanctot's slides.