COMP30026 Models of Computation

Context-Free Languages

Harald Søndergaard

Lecture 16

Semester 2, 2017

A Bit of History

Finite-state machines go back to McCulloch and Pitts (1943), who wanted to model the working of neurons and synapses.

The formalism that we use today is from Moore (1956).

Kleene (1956) established the connections between regular expressions and finite-state automata.

We now turn to context-free grammars.

These go back to Post's "productions" and Chomsky's grammar formalism (1956).

Chomsky, a linguist, proposed a range of formalisms in grammar form for the description of natural language syntax.

Context-Free Grammars in Computer Science

Exactly how computer scientists ended up adopting context-free grammars is lost in the fog of history.

Backus used them to describe Algol 58 syntax, and they gained popularity with the Algol 60 Report, edited by Naur (1963).

They are frequently referred to as Backus-Naur Formalism (BNF).

Standard tools for parsing owe much to this formalism, which indirectly has helped make parsing a routine task.

It is extensively used to specify syntax of programming languages, document formats (XML's document-type definition), etc.

Pushdown automata are to context-free grammars what finite-state automata are to regular languages.

Context-Free Grammars

We have already used the formalism of context-free grammars. To specify the syntax of regular expressions we gave a grammar, much like

$$
\begin{array}{lcl}
R & \rightarrow & 0 \\
R & \rightarrow & 1 \\
R & \rightarrow & \epsilon \\
R & \rightarrow & \emptyset \\
R & \rightarrow & R \cup R \\
R & \rightarrow & R\ R \\
R & \rightarrow & R^*
\end{array}
$$

Hence a grammar is a set of substitution rules, or productions. We have the shorthand notation

$$R \quad \rightarrow 0 \mid 1 \mid \epsilon \mid \emptyset \mid R \cup R \mid R\ R \mid R^*$$

Derivations, Sentences and Sentential Forms

A simpler example is this grammar G:

$$\begin{array}{lcl} A & \rightarrow & 0\ A\ 1\ 1 \\ A & \rightarrow & \epsilon \end{array}$$

Using the two rules as a rewrite system, we get derivations such as

$$\begin{array}{lcl} A & \Rightarrow & 0A11 \\ & \Rightarrow & 00A1111 \\ & \Rightarrow & 000A111111 \\ & \Rightarrow & 000111111 \end{array}$$

A is called a variable. Other symbols (here 0 and 1) are terminals. We refer to a valid string of terminals (such as `000111111`) as a sentence. The intermediate strings that mix variables and terminals are sentential forms.

Context-Free Languages

Clearly a grammar determines a formal language.

The language of G is written $L(G)$.

$$L(G) = \{0^n 1^{2n} \mid n \geq 0\}$$

A language which can be generated by some context-free grammar is a context-free language (CFL).

It should be clear that some of the languages that we found not to be regular are context-free, for example

$$\{0^n 1^n \mid n \geq 1\}$$

Context-Free Grammars Formally

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

1. V is a finite set of variables,
2. Σ is a finite set of terminals,
3. R is a finite set of rules, each consisting of a variable (the left-hand side) and a sentential form (the right-hand side),
4. S is the start variable.

The binary relation $\Rightarrow$ on sentential forms is defined as follows.

Let u, v, and w be sentential forms. Then $uAw \Rightarrow uvw$ iff $A \rightarrow v$ is a rule in R. That is, $\Rightarrow$ captures a single derivation step.

Let $\stackrel{*}{\Rightarrow}$ be the reflexive transitive closure of $\Rightarrow$.

$$L(G) = \{s \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} s\}$$

A Context-Free Grammar for Numeric Expressions

Here is a grammar with three variables, 14 terminals, and 15 rules:

$$
\begin{array}{lcl}
E & \rightarrow & T \mid T + E \\
T & \rightarrow & F \mid F * T \\
F & \rightarrow & 0 \mid 1 \mid \ldots \mid 9 \mid (\ E\ )
\end{array}
$$

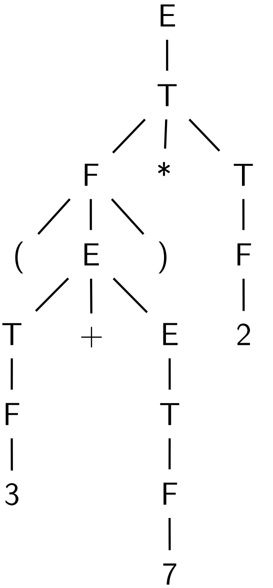
When the start variable is unspecified, it is assumed to be the variable of the first rule.

An example sentence in the language is

```
(3 + 7) * 2
```

The grammar ensures that * binds tighter than +.

Parse Trees



A **parse tree** for

`(3 + 7) * 2`

Parse Trees

There are different derivations leading to the sentence `(3 + 7) * 2`, all corresponding to the parse tree above. They differ in the order in which we choose to replace variables. Here is the leftmost derivation:

$$
\begin{array}{rcl}
E & \Rightarrow & T \\
 & \Rightarrow & F \ * \ T \\
 & \Rightarrow & (\ E \ ) \ * \ T \\
 & \Rightarrow & (\ T \ + \ E \ ) \ * \ T \\
 & \Rightarrow & (\ F \ + \ E \ ) \ * \ T \\
 & \Rightarrow & (\ 3 \ + \ E \ ) \ * \ T \\
 & \Rightarrow & (\ 3 \ + \ T \ ) \ * \ T \\
 & \Rightarrow & (\ 3 \ + \ F \ ) \ * \ T \\
 & \Rightarrow & (\ 3 \ + \ 7 \ ) \ * \ T \\
 & \Rightarrow & (\ 3 \ + \ 7 \ ) \ * \ F \\
 & \Rightarrow & (\ 3 \ + \ 7 \ ) \ * \ 2
\end{array}
$$

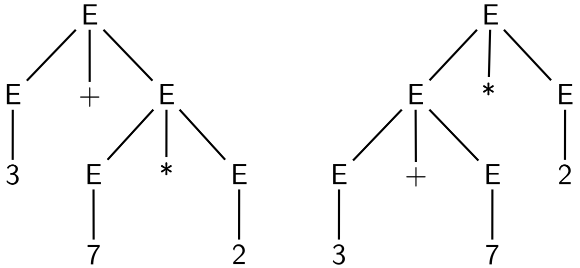
$$
\begin{array}{rcl}
E & \rightarrow & T \mid T + E \\
T & \rightarrow & F \mid F * T \\
F & \rightarrow & 0 \mid 1 \mid \ldots \mid 9 \mid (\ E \ )
\end{array}
$$

Ambiguity

Consider the grammar

$$E \rightarrow E + E \mid E * E \mid (E) \mid 0 \mid 1 \mid \ldots \mid 9$$

This grammar allows not only different derivations, but different parse trees for `3 + 7 * 2`:



Accidental vs Inherent Ambiguity

A grammar that has different parse trees for some sentence is ambiguous.

Sometimes we can find a better grammar (as in our example) which is not ambiguous, and which generates the same language.

However, this is not always possible: There are CFLs that are inherently ambiguous.