COMP30026 Models of Computation

Context-Free Languages

Bach Le / Anna Kalenkova

Lecture Week 8 Part 2

Semester 2, 2021

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).

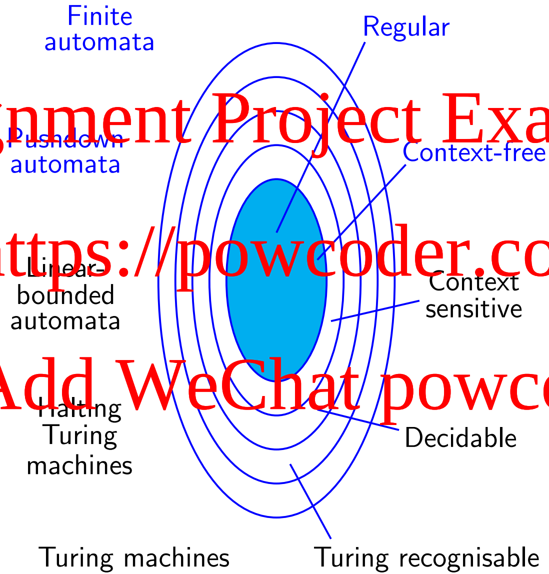
Kleene (1956) established the connections between regular expressions and finite-state automata.

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.

Machines vs Languages



Context-Free Grammars in Computer Science

In the 60's, computer scientists started adopting context-free free grammars to describe syntax of programming languages.

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, data formats (XML, JSON), 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

$$
\begin{array}{lcl}
R & \rightarrow & 0 \\
R & \rightarrow & 1 \\
R & \rightarrow & \text{eps} \\
R & \rightarrow & \text{empty} \\
R & \rightarrow & R \cup R \\
R & \rightarrow & R\ R \\
R & \rightarrow & R^*
\end{array}
$$

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

$$R \rightarrow 0 \mid 1 \mid \text{eps} \mid \text{empty} \mid R \cup R \mid R\ R \mid R^*$$

A simpler example is this grammar G:

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

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

$$\begin{array}{rcl} A & \Rightarrow & 0A0 \\ & \Rightarrow & 00A00 \\ & \Rightarrow & 001A100 \\ & \Rightarrow & 00100100 \end{array}$$

A is called a variable. Other symbols (here 0 and 1) are terminals. We refer to a valid string of terminals (such as 00100100) 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 $L(G)$.

$$L(G) = \{ww^R \mid w \in \{0,1\}^*\}$$

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 string in $(V \cup \Sigma)^*$ (the right-hand side),
4. S is the start variable.

The binary relation $\Rightarrow$ on $(V \cup \Sigma)^*$ is defined as follows.

Let $u, v, w \in (V \cup \Sigma)^*$. Then $uAw \Rightarrow uvw$ iff $A \to v$ is a rule in R.
That is, $\Rightarrow$ captures a single derivation step.

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

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

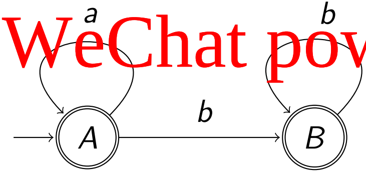
Right/Left Regular Grammars (Not Examinable)

Right regular grammar:

$$\begin{array}{lcl} A & \rightarrow & a\ A \\ A & \rightarrow & \epsilon \\ A & \rightarrow & b\ B \\ B & \rightarrow & b\ B \\ B & \rightarrow & \epsilon \end{array}$$

Left regular grammar:

$$\begin{array}{lcl} A & \rightarrow & A\ b \\ A & \rightarrow & \epsilon \\ A & \rightarrow & B\ a \\ B & \rightarrow & B\ a \\ B & \rightarrow & \epsilon \end{array}$$



A Context-Free Grammar for Numeric Expressions

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

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

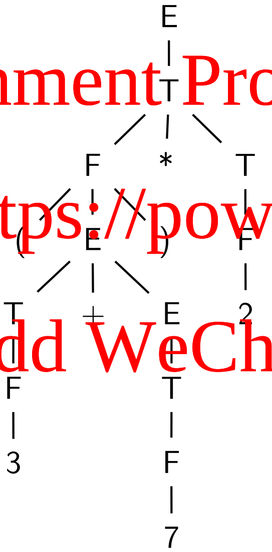
When the start variable is unspecified, it is assumed to be the variable of the first rule.

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:

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

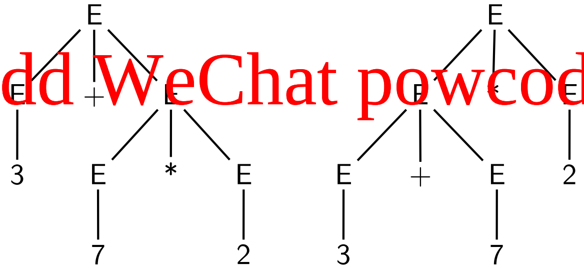
$$
\begin{aligned}
E &\rightarrow T \mid T + E \\
T &\rightarrow F \mid F * T \\
F &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \mid (\, E \,)
\end{aligned}
$$

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.