5 Context free grammars

We will now introduce *context free grammars* (CFGs) as a formalism to define languages. CFGs are more general than regular expressions, i.e. there are more languages definable by CFGs (called *type-2-languages*). We will define the corresponding notion of automata, the *push down automata* (PDA) later.

5.1 What is a context-free grammar?

A context-free grammar $G = (V, \Sigma, S, P)$ is given by

- A finite set V of variables or nonterminal symbols.
- A finite set Σ of symbols or terminal symbols. We assume that the sets V and Σ are disjoint.
- A start symbol $S \in V$.
- A finite set $P \subseteq V \times (V \cup T)^*$ of productions. A production (A, α), where $A \in V$ and $\alpha \in (V \cup T)^*$ is a sequence of terminals and variables, is written as $A \to \alpha$.

As an example we define a grammar for the language of arithmetical expressions over a (using only $+$ and $*$), i.e. elements of this language are $a + (a * a)$ or $(a + a) * (a + a)$. However words like $a + +a$ or $)(a$ are not in the language.
We define $G = (\{E, T, F\}, \{(,), a, +, *\}, E, P)$ where P is given by:

$$\begin{aligned} P = \{ & E \to T \\ & E \to E + T \\ & T \to F \\ & T \to T * F \\ & F \to a \\ & F \to (E)\} \end{aligned}$$

To save space we may combine all the rules with the same left hand side, i.e. we write

$$\begin{aligned} P = \{ & E \to T \mid E + T \\ & T \to F \mid T * F \\ & F \to a \mid (E) \end{aligned}$$

5.2 The language of a grammar

How do we check whether a word $w \in \Sigma^*$ is in the language of the grammar? We start with the start symbol E and use one production $V \to \alpha$ to replace the nonterminal symbol by the α until we have no nonterminal symbols left - this

is called *a derivation.* I.e. in the example G:

$$\begin{aligned} E & \Rightarrow_G E + T \\ & \Rightarrow_G T + T \\ & \Rightarrow_G F + T \\ & \Rightarrow_G a + T \\ & \Rightarrow_G a + F \\ & \Rightarrow_G a + (E) \\ & \Rightarrow_G a + (T) \\ & \Rightarrow_G a + (T * F) \\ & \Rightarrow_G a + (F * F) \\ & \Rightarrow_G a + (a * F) \\ & \Rightarrow_G a + (a * a) \end{aligned}$$

Note that $\Rightarrow_G$ here stands for the relation *derives in one step* and has nothing to do with implication. In the example we have always replaced the leftmost non-terminal symbol (hence it is called a leftmost derivation) but this is not necessary.
Given any grammar $G = (V, \Sigma, S, P)$ we define the relation *derives in one step*

$$\begin{aligned} \Rightarrow_G & \subseteq (V \cup T)^* \times (V \cup T)^* \\ \alpha V \gamma \Rightarrow_G \alpha\beta\gamma & :\iff V \to \beta \in P \end{aligned}$$

The relation *derives* is defined as[2]

$$\begin{aligned} \Rightarrow_G^* & \subseteq (V \cup T)^* \times (V \cup T)^* \\ \alpha_0 \Rightarrow_G^* \alpha_n & :\iff \alpha \Rightarrow_G \alpha_1 \Rightarrow_G \dots \alpha_{n-1} \Rightarrow_G \alpha_n \end{aligned}$$

this includes the case $\alpha \Rightarrow_G^* \alpha$ because n can be 0.
We now say that the language of a grammar $L(G) \subseteq \Sigma^*$ is given by all words (over Σ) which can be derived in any number of steps, i.e.

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$$

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

5.3 More examples

Some of the languages which we have shown not to be regular are actually context-free.
The language $\{0^n 1^n \mid n \in \mathbb{N}\}$ is given by the following grammar

$$G = (\{S\}, \{0, 1\}, S, \{S \to \epsilon \mid 0S1\})$$

Also the language of palindromes

$$\{ww^R \mid w \in \{\mathrm{a}, \mathrm{b}\}^*\}$$

[2] $\Rightarrow_G^*$ is the *transitive-reflexive closure* of $\Rightarrow_G^*$.

turns out to be context-free;

$$G = (\{S\}, \{\text{a}, \text{b}\}, S, \{S \to \epsilon \mid \text{a}S\text{a} \mid \text{b}S\text{b}\})$$

We also note that

Theorem 5.1 *All regular languages are context-free.*

We don't give a proof here - the idea is that regular expressions can be translated into (special) context-free grammars, i.e. a^*b^* can be translated into

$$G = (\{A, B\}, \{\text{a}, \text{b}\}, \{A \to aA \mid B, B \to bB \mid \epsilon)$$

(Extensions of) context free grammars are used in computer linguistics to describe real languages. As an example consider

$$\Sigma = \{\text{the}, \text{dog}, \text{cat}, \text{which}, \text{bites}, \text{barks}, \text{catches}\}$$

we use the grammar $G = (\{S, N, NP, VI, VT, VP\}, \Sigma, S, P)$ where

$$\begin{aligned} S &\to NP\ \ VP \\ N &\to \text{cat} \mid \text{dog} \\ NP &\to \text{the } N \mid NP \text{ which } VP \\ VI &\to \text{barks} \mid \text{bites} \\ VT &\to \text{bites} \mid \text{catches} \\ VP &\to VI \mid VT\ NP \end{aligned}$$

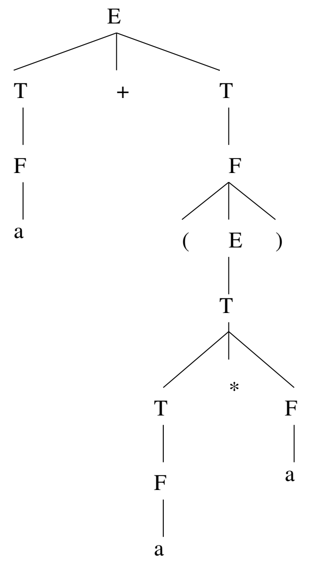
which allows us to derive interesting sentences like

the dog which catches the cat which bites barks

An important example for context-free languages is the syntax of programming languages. We have already mentioned appendix of [GJSB00] which uses a formalism slightly different from the one introduced here. Another example is the toy language used in the compilers course ([Bac02], see *Mini-Triangle Concrete and Abstract Syntax*
However, note that not all syntactic aspects of programming languages are captured by the context free grammar, i.e. the fact that a variable has to be declared before used and the type correctness of expressions are not captured.

5.4 Parse trees

With each derivation we also associate a *derivation tree*, which shows the structure of the derivation. As an example consider the tree associated with the derivation of $a + (a * a)$ given before:



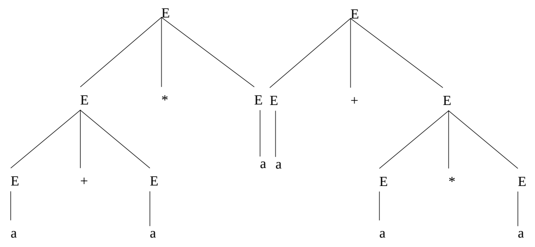
The top of the tree (called its root) is labelled with start symbol, the other *nodes* are labelled with nonterminal symbols and the leaves are labelled by terminal symbols. The word which is derived can be read off from the leaves of the tree. The important property of a parse tree is that the ancestors of an internal node correspond to a production in the grammar.

5.5 Ambiguity

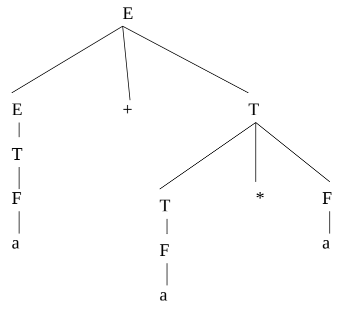
We say that a grammar G is *ambiguous* if there is a word $w \in L(G)$ for which there is more than one parse tree. This is usually a bad thing because it entails that there is more than one way to interpret a word (i.e. it leads to semantical ambiguity).
As an example consider the following alternative grammar for arithmetical expressions: We define $G' = (\{E\}, \{(,), a, +, *\}, E, P')$ where P' is given by:

$$P' = \{E \to E + E \mid E * E \mid a \mid (E)\}$$

This grammar is shorter and requires only one variable instead of 4. Moreover it generates the same language, i.e. we have $L(G) = L(G')$. But it is ambigous: Consider $a + a * a$ we have the following parse trees:



Each parse tree correspond to a different way to read the expression, i.e. the first one corresponds to $(a+a)*a$ and the second one to $a+(a*a)$. Depending on which one is chosen an expression like $2+2*3$ may evaluate to 12 or to 8. Informally, we agree that $*$ binds more than $+$ and hence the 2nd reading is the intended one.
This is actually achieved by the first grammar which only allows the 2nd reading:


E
E
+
T
T
F
a
T
*
F
F
a
a