

# Compilers

## Lecture II: Formal Language Theory

Alessandro Artale

Faculty of Computer Science – Free University of Bolzano

POS Building – Room: 2.03

`artale@inf.unibz.it`

`http://www.inf.unibz.it/~artale/`

## Summary of Lecture II

- **Grammars.**
- Generating Languages from Grammars.
- Chomsky Classification.
- Derivation Trees

## Formal Language Theory

- The **Formal Language Theory** considers a Language as a mathematical object.
- A Language is just a set of strings. To formally define a Language we need to formally define what are the strings admitted by the Language.
- Formal Notions:
  1. **Alphabet.** A set of symbols, indicated by  $V$  (e.g.,  $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ).
  2. **String.** A string over an alphabet,  $V$ , is a sequence of symbols belonging to the alphabet (e.g., “518” is a string over the above  $V$ ). The empty string is denoted by  $\epsilon$ .
  3. **Linguistic Universe.** Indicated by  $V^*$ , denotes the set of all possible strings over  $V$ , included  $\epsilon$ . The set  $V^+$  denotes the set  $V^* \setminus \epsilon$ .
- Formally characterize a Language means:

**Find a finite representation of all admissible strings.**

## Grammars

- The notion of **Grammar** is related to studies in natural languages.
- Linguists were concerned with:
  1. Defining the valid sentences of a Language;
  2. Providing a structural definition of such valid sentences.
- A Grammar is a formalism to give a finite representation of a Language.
- A Grammar gives a **Generative** perspective: It defines the way in which all admissible strings can be generated.

## Formal Notion of Grammar

- A Grammar,  $G$ , is a tuple:  $G = (V_T, V_N, S, P)$ , such that:
  - $V_T$  is the finite set of *Terminal Symbols*.
  - $V_N$  is the finite set of *Non-Terminal Symbols*.
  - $S$  is the *Scope* of the Language, and  $S \in V_N$ .
  - $P$  is the finite set of *Productions*,  $P = \{\alpha \rightarrow \beta \mid \alpha \in V^+ \wedge \beta \in V^*\}$ .
  - Terminal and Non-Terminal symbols give rise to the alphabet:  
 $V = V_T \cup V_N$ .
  - Terminal and Non-Terminal symbols are disjoint sets:  $V_T \cap V_N = \emptyset$ .

## Summary

- Grammars.
- **Generating Languages from Grammars.**
- Chomsky Classification.
- Derivation Trees.

## Notion of Derivation

- To characterize a Language starting from a Grammar we need to introduce the notion of **Derivation**.
- The notion of Derivation uses Productions to generate a string starting from another string.
- **Direct Derivation (in symbols  $\Rightarrow$ )**.

If  $\alpha \rightarrow \beta \in P$  and  $\gamma, \delta \in V^*$ , then,  $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ .

- **Derivation (in symbols  $\Rightarrow^*$ )**.

If  $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \dots, \alpha_{n-1} \Rightarrow \alpha_n$ , then,  $\alpha_1 \Rightarrow^* \alpha_n$ .

## Generating Languages from Grammars

**Generative Definition of a Language.** We say that a Language  $L$  is *generated* by the *Grammar*  $G$ , in symbols  $L(G)$ , if:

$$L(G) = \{w \in V_T^* \mid S \Rightarrow^* w\}.$$

We say that two Languages are *equivalent* if  $L(G_1) \equiv L(G_2)$ .

The above definition says that a string belongs to a Language if and only if:

1. The string is made only of Terminal Symbols;
2. The string can be Derived from the Scope,  $S$ , of the Language.

## Generating Languages from Grammars: Examples

**Example 1.** Let us consider the following Grammar,  $G = (V_T, V_N, S, P)$ :

- $V_T = \{0, 1\}$ ;
- $V_N = \{S\}$ ;
- $P = \{S \rightarrow 0S1, S \rightarrow 01\}$ ;

Then:

- $S \Rightarrow^* 0^n 1^n$ ;
- $L(G) = \{0^n 1^n \mid n \geq 1\}$ .

## Generating Languages from Grammars: Examples

**Example 2.** Let us consider the following Grammar,  $G = (V_T, V_N, S, P)$ :

- $V_T = \{a, b\}$ ;
- $V_N = \{S, A, B\}$ ;
- $S = S$ .

With Productions in P:

1.  $S \rightarrow AB$
2.  $A \rightarrow aA$
3.  $A \rightarrow \epsilon$
4.  $B \rightarrow bB$
5.  $B \rightarrow \epsilon$

Then:

- $S \Rightarrow^1 AB \Rightarrow^2 aAB \Rightarrow^2 aaAB \Rightarrow^4$
- $aaaAB \Rightarrow^3 aaaB \Rightarrow^4 aaabB \Rightarrow^4$
- $aaabbB \Rightarrow^5 aaabb$
- $L(G) = \{a^m b^n \mid m, n \geq 0\}$

## Generating Languages from Grammars: Examples

**Example 3.** Let us consider the following Grammar with more than one symbol on the left side of Productions,  $G = (V_T, V_N, S, P)$ :

- $V_T = \{a\}$ ;
- $V_N = \{S, N, Q, R\}$ ;
- $S = S$ .

With Productions in P:

1.  $S \rightarrow QNQ$
2.  $QN \rightarrow QR$
3.  $RN \rightarrow NNR$
4.  $RQ \rightarrow NNQ$
5.  $N \rightarrow a$
6.  $Q \rightarrow \epsilon$

Then:

- $S \Rightarrow^1 QNQ \Rightarrow^2 QRQ \Rightarrow^4 QNNQ \Rightarrow^2 QRNQ \Rightarrow^3 QNNRQ \Rightarrow^4 QNNNNQ \Rightarrow^* aaaaa$
- $L(G) = \{a^{(2^n)} \mid n \geq 0\}$

## Summary

- Grammars.
- Generating Languages from Grammars.
- **Chomsky Classification.**
- Derivation Trees.

## Chomsky Classification

- The concept of **Grammar Classification** was introduced by Noam Chomsky in the 1950s as a way to describe the structural complexity of particular sentences of natural language.
- Languages are classified w.r.t. the Grammar that generates them: Different constraints on Productions define different classes of Grammars/Languages.

## Type 0 Grammars

The most general Grammars are the so called **Type 0** Grammars. They are formal Grammars,  $G = (V_T, V_N, S, P)$ , such that all productions in  $P$  respect the following condition:

**Type 0.**  $\alpha \rightarrow \beta$

with  $\alpha \in V^+$  and  $\beta \in V^*$ .

The Grammar of **Example 3** is a Type 0 Grammar.

## Type 1, Context-Sensitive Grammars

**Context-Sensitive Grammars**, also called **Type 1 Grammars**, are formal Grammars,  $G = (V_T, V_N, S, P)$ , such that all productions in  $P$  respect the following condition:

**Type 1.**  $\alpha A \gamma \rightarrow \alpha \beta \gamma$

with  $\alpha, \gamma \in V^*$  and  $\beta \in V^+$  and  $A \in V_N$ . Furthermore, a rule of the form:

$S \rightarrow \epsilon$

is allowed if  $S$  does not appear on the right side of any rule.

- The meaning of “Context-Sensitive” is explained by the  $\alpha$  and  $\gamma$  that form then context of  $A$  and determine whether  $A$  can be replaced with  $\beta$  or not.

## Type 2, Context-Free Grammars

**Context-Free Grammars**, also called **Type 2 Grammars**, are formal Grammars,  $G = (V_T, V_N, S, P)$ , such that all productions in  $P$  respect the following condition:

**Type 2.**  $A \rightarrow \beta$

with  $A \in V_N$  and  $\beta \in V^*$ .

- The term “Context-Free” comes from the fact that the non-terminal  $A$  can always be replaced by  $\beta$ , in no matter what context it occurs.
- **Context-Free Grammars are important because they are powerful enough to describe the syntax of programming languages; in fact, almost all programming languages are defined via Context-Free Grammars.**

## Type 2, Context-Free Grammars (Cont.)

- Context-Free Grammars are simple enough to allow the construction of efficient parsing algorithms which for a given string determine whether and how it can be generated from the Grammar.
- The Syntactical Analysis of a Compiler is based on implementing Parses based on Context-Free Grammars.
- The Grammar of **Example 1** is a Context-Free Grammar. The Grammar describing assignment is a Context-Free Grammar:

$$\langle assignment \rangle \rightarrow ID \text{ “ := ” } \langle expr \rangle$$

$$\langle expr \rangle \rightarrow ID \mid NUM \mid \langle expr \rangle \langle op \rangle \langle expr \rangle \mid ( \langle expr \rangle )$$

$$\langle op \rangle \rightarrow + \mid - \mid * \mid /$$

- **Exercise.** What is the alphabet  $V$  of the above Grammar?

## Type 3, Regular Grammars

**Regular Grammars**, also called **Type 3 Grammars**, are formal Grammars,  $G = (V_T, V_N, S, P)$ , such that all productions in  $P$  respect the following condition:

**Type 3.**  $A \rightarrow aB$ , or  $A \rightarrow a$

with  $A, B \in V_N$  and  $a \in V_T$ . Furthermore, a rule of the form:

$S \rightarrow \epsilon$

is allowed if  $S$  does not appear on the right side of any rule.

- The above define the *Right-Regular Grammars*. The following Productions:

$A \rightarrow Ba$ , or  $A \rightarrow a$

define *Left-Regular Grammars*.

- Right-Regular and Left-Regular Grammars define the same set of Languages.
- **Regular Grammars are commonly used to define the lexical structure of programming languages.**
- **Exercise.** Even if the Grammar of **Example 2** is a Context-Free Grammar the generated Language can be expressed by an equivalent Regular Grammar.

## Summing Up

- Grammar/Language Types form a hierarchy of languages, also called the *Chomsky Hierarchy*.
- Every Regular Language is Context-Free, every Context-Free Language is Context-Sensitive and every Context-Sensitive Language is a Type 0 Language.
- These are all proper inclusions, meaning that there exist Type 0 Languages which are not Context-Sensitive, Context-Sensitive Languages which are not Context-Free and Context-Free Languages which are not Regular.
- **Theorem.** Let  $G$  be a Context-Sensitive-Grammar then  $G$  is *recursive*: There is an algorithm such that for any string  $w$  determines whether  $w \in L(G)$ .

## Summary

- Grammars.
- Generating Languages from Grammars.
- Chomsky Classification.
- **Derivation Trees.**

## Derivation Trees for Context-Free Grammars

- **Derivation Trees**, called also **Parse Trees**, are a visual method of describing any derivation in a context-free grammar.
- Let  $G = (V_T, V_N, S, P)$  be a CFG. A tree is a *derivation tree* for  $G$  if:
  1. Every node has a *label*, which is a symbol of  $V$ ;
  2. The label of the root is  $S$ ;
  3. If a node,  $n$ , labeled with  $A$  has at least one descendant, then  $A$  must be in  $V_N$ ;
  4. If nodes  $n_1, n_2, \dots, n_k$  are direct descendants of node  $n$ , with labels  $A_1, A_2, \dots, A_k$ , respectively, then:
$$A \rightarrow A_1, A_2, \dots, A_k$$
must be a production in  $P$ .

## Derivation Trees: An Example

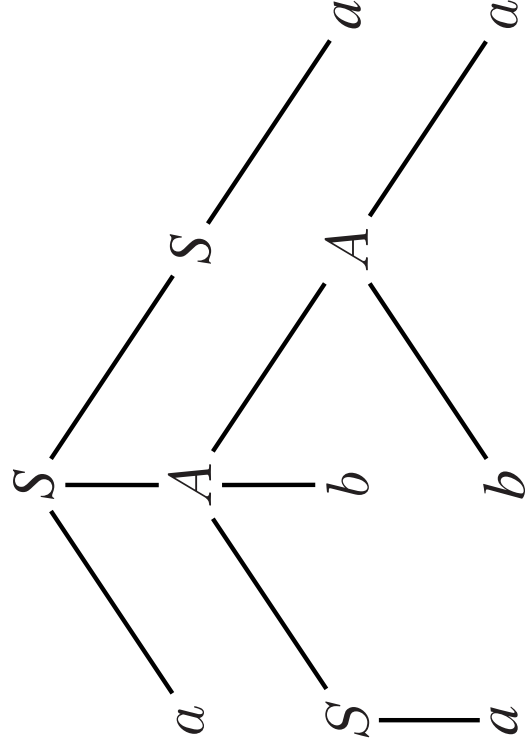
**Example.** Let  $G = (\{a, b\}, \{S, A\}, S, P)$ , where  $P$  is:

$$S \rightarrow aAS \quad S \rightarrow a$$

$$A \rightarrow SbA \quad A \rightarrow ba$$

$$A \rightarrow SS$$

The following is an example of a derivation tree:



## Derivation Trees (Cont.)

- Derivation Trees are visual representation of Grammar's derivations.
- We indicate as *Leaves* nodes in derivation trees without descendants.
- If we read the leaves from left to right we have a *sentential form*, called also the *result* of the derivation tree.
- **Theorem.** Let  $G = (V_T, V_N, S, P)$  a context-free grammar, then, for  $\alpha \neq \epsilon$ ,  $S \Rightarrow^* \alpha$  if and only if there is a derivation tree in grammar  $G$  with *result*  $\alpha$ .

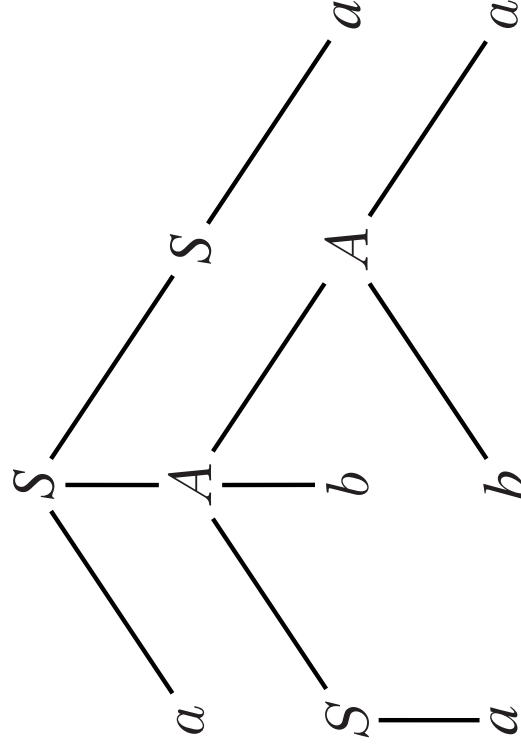
## Derivation Trees: An Example (Cont.)

**Example.** Let  $G = (\{a, b\}, \{S, A\}, S, P)$ , where  $P$  is:

$$S \rightarrow aAS \quad S \rightarrow a$$

$$A \rightarrow SbA \quad A \rightarrow ba$$

$$A \rightarrow SS$$



The *result* of the derivation tree is:  $aabbbaa$ . Now,  $S \Rightarrow^* aabbbaa$  by:

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbbaa.$$

## Summary of Lecture II

- Grammars.
- Generating Languages from Grammars.
- Chomsky Classification.
- Derivation Trees.