Formal Languages and 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/

2014/2015 – First Semester

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 finite, non-empty set of symbols, indicated by $\mathbf{V}$ (e.g., $\mathbf{V} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$).
  2. **String.** A string over an alphabet, $\mathbf{V}$, is a sequence (concatenation) of symbols belonging to the alphabet (e.g., "518" is a string over the above $\mathbf{V}$). The empty string is denoted by ϵ.
  3. **Linguistic Universe.** Indicated by $\mathbf{V}^*$, denotes the set of all possible finite strings over $\mathbf{V}$, included ϵ. The set $\mathbf{V}^+$ denotes the set $\mathbf{V}^* \setminus \epsilon$.

Formal Language Theory (cont.)

- **Language** L over V is any subset of V^*: $\mathsf{L} \subseteq \mathsf{V}^*$.
  Note: L may be infinite.

- Examples.

$$\begin{cases} \mathsf{V} & = \{a, b, \ldots, z\} \\ \mathsf{L} & = \{\text{all English words}\} \end{cases}$$

$$\begin{cases} \mathsf{V} & = \{0, 1\} \\ \mathsf{L} & = \{\epsilon, 01, 0011, 000111, \ldots\} \end{cases}$$

- 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 that gives a finite representation of a Language.
- A Grammar gives a Generative perspective: It defines the set of rules by which all admissible strings can be generated.

Formal Notion of Grammar

- Introduced by the linguist Noam Chomsky in the 1950s.
- A Grammar, **G**, is a tuple: $\mathbf{G} = (\mathbf{V_T}, \mathbf{V_N}, \mathbf{S}, \mathbf{P})$, such that:
  - $\mathbf{V_T}$ is the finite set of *Terminal Symbols*.
  - $\mathbf{V_N}$ is the finite set of *Non-Terminal Symbols*.
  - Terminal and Non-Terminal symbols give rise to the alphabet: $\mathbf{V} = \mathbf{V_T} \cup \mathbf{V_N}$.
  - Terminal and Non-Terminal symbols are disjoint sets: $\mathbf{V_T} \cap \mathbf{V_N} = \emptyset$.
  - $\mathbf{S} \in \mathbf{V_N}$ is the *Scope* of the Language.
  - $\mathbf{P}$ is the finite set of *Productions*: $\mathbf{P} = \{\alpha \rightarrow \beta \mid \alpha \in \mathbf{V}^* \cdot \mathbf{V_N} \cdot \mathbf{V}^*, \textit{ and } \beta \in \mathbf{V}^*\}$.

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 $\alpha \rightarrow \beta \in \mathbf{P}$ and $\gamma, \delta \in \mathbf{V}^*$, then, $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$.
- **Derivation (in symbols $\Rightarrow^*$).**
  If $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$, 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* $\mathbf{G}$, in symbols $\mathsf{L}(\mathsf{G})$, if:

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

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

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 is Derived from the Scope, $\mathbf{S}$, of the Language;
3. Strings belonging to a language L are called sentences.

Generating Languages from Grammars: Examples

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

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

Then:

- $S \Rightarrow^* 0^n1^n$;
- $\mathsf{L(G)} = \{0^n1^n \mid n \geq 1\}$.

Generating Languages from Grammars: Examples

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

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

With Productions in $\mathbf{P}$:

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

Then:

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

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**):

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

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:

- $S \Rightarrow^1 QNQ \Rightarrow^2 QRQ \Rightarrow^4 QNNQ \Rightarrow^2 QNNNQ \Rightarrow^3 QNNNRQ \Rightarrow^4 QNNNNNQ \Rightarrow^* aaaa$
- $\mathbf{L(G)} = \{a^{(2^n)} \mid n \geq 0\}$

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, $\mathbf{G}=(\mathbf{V_T},\mathbf{V_N},\mathbf{S},\mathbf{P})$, such that all productions in $\mathbf{P}$ respect the following condition:

Type 0. $\alpha \rightarrow \beta$

with $\alpha \in \mathbf{V}^* \cdot \mathbf{V_N} \cdot \mathbf{V}^*$ and $\beta \in \mathbf{V}^*$.

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, $\mathbf{G}$= ($\mathbf{V_T}$,$\mathbf{V_N}$,$\mathbf{S}$,$\mathbf{P}$), such that all productions in $\mathbf{P}$ respect the following condition:

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

with $\alpha, \gamma \in \mathbf{V}^*$, $\beta \in \mathbf{V}^+$ and $A \in \mathbf{V_N}$. Furthermore, a rule of the form:
$\mathbf{S} \rightarrow \epsilon$
is allowed if $\mathbf{S}$ does not appear on the right side of any rule.

- The meaning of "Context-Sensitive" is explained by the α and γ that form then context of A and determine whether A can be replaced with β or not.

Type 2, Context-Free Grammars

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

Type 2. $A \rightarrow \beta$

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

- The term "Context-Free" comes from the fact that the non-terminal A can always be replaced by β, 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:

$$
\begin{aligned}
< assignment > &\rightarrow \textsf{ID} \text{ “} = \text{”} < expr > \\
< expr > &\rightarrow \textsf{ID} \mid \textsf{NUM} \mid < expr >< op >< expr > \mid (< expr >) \\
< op > &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

- **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. $A \longrightarrow aB$, or $A \longrightarrow a$

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

$\mathbf{S} \longrightarrow \epsilon$

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

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

  $A \longrightarrow Ba$, or $A \longrightarrow a$

  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 $w \in L(\mathsf{G})$.

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 $\mathbf{G}$= ($\mathbf{V_T}$,$\mathbf{V_N}$,$\mathbf{S}$,$\mathbf{P}$) be a CFG. A tree is a *derivation tree* for $\mathbf{G}$ if:
  1. Every node has a *label*, which is a symbol of $\mathbf{V}$;
  2. The label of the root is $\mathbf{S}$;
  3. If a node, n, labeled with A has at least one descendant, then A must be in $\mathbf{V_N}$;
  4. If nodes $n_1, n_2, \ldots, n_k$ are direct descendants of node n, with labels $A_1, A_2, \ldots, A_k$, respectively, then:

     $$A \rightarrow A_1, A_2, \ldots, A_k$$

     must be a production in $\mathbf{P}$.

Derivation Trees: An Example

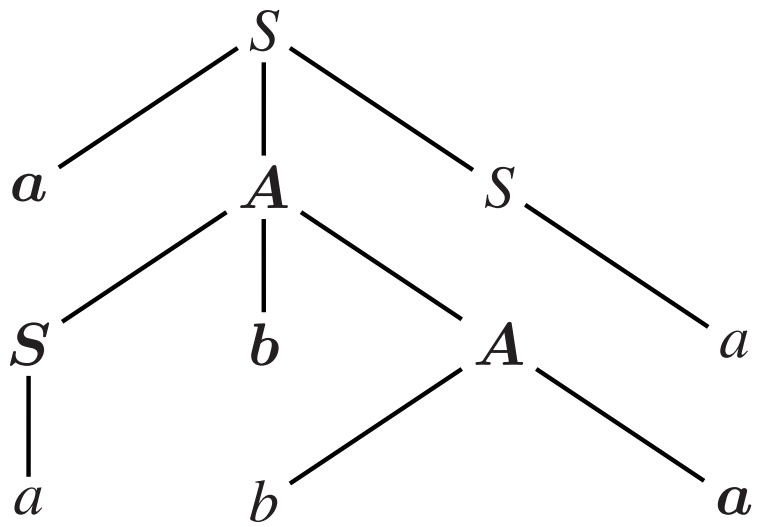
Example. Let **G**=($\{a, b\}$, $\{S, A\}$, S, **P**), where **P** is:

$S \longrightarrow aAS \quad S \longrightarrow a$

$A \longrightarrow SbA \quad A \longrightarrow ba$

$A \longrightarrow SS$

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 *sentence*, called also the *result* of the derivation tree.
- **Theorem.** Let $\mathbf{G}$= ($\mathbf{V_T}$,$\mathbf{V_N}$,$\mathbf{S}$,$\mathbf{P}$) a context-free grammar, then, for $\alpha \neq \epsilon$, $\mathbf{S} \Rightarrow^* \alpha$ if and only if there is a derivation tree in grammar $\mathbf{G}$ with *result* α.

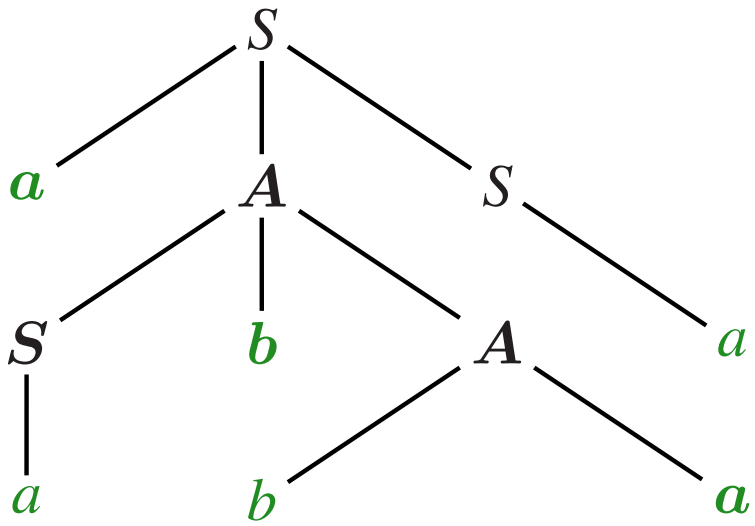
Derivation Trees: An Example (Cont.)

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

$S \longrightarrow aAS \quad S \longrightarrow a$

$A \longrightarrow SbA \quad A \longrightarrow ba$

$A \longrightarrow SS$



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

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

Summary of Lecture II

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