Formal Languages and Compilers
Lecture II: Formal Language Theory

Alessandro Artale

Free University of Bozen-Bolzano
Faculty of Computer Science – POS Building, Room: 2.03
artale@inf.unibz.it
http://www.inf.unibz.it/~artale/

Formal Languages and Compilers — BSc course

2020/21 – Second 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 V (e.g., $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$).
  2. **String.** A string over an alphabet, V, is a sequence (concatenation) of symbols belonging to the alphabet (e.g., "518" is a string over the above V). The empty string is denoted by ϵ.
  3. **Linguistic Universe.** Indicated by V^*, denotes the set of all possible finite strings over V, included ϵ. The set V^+ denotes the set $V^* \setminus \epsilon$.

Formal Language Theory (cont.)

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

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

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

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

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 $\alpha \rightarrow \beta \in P$ 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 (so called, sentences) if and only if:

1. The string is made only of Terminal Symbols;
2. The string is 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):

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

Then:

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

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:

$$
\begin{array}{llll}
r1. & S & \rightarrow & AB \\
r2. & A & \rightarrow & aA \\
r3. & A & \rightarrow & \epsilon \\
r4. & B & \rightarrow & bB \\
r5. & B & \rightarrow & \epsilon
\end{array}
$$

Then:

- $$S \Rightarrow^{r1} AB \Rightarrow^{r2} aAB \Rightarrow^{r2} aaAB \Rightarrow^{r2} aaaAB \Rightarrow^{r3} aaaB \Rightarrow^{r4} aaabB \Rightarrow^{r4} aaabbB \Rightarrow^{r5} aaabb$$
- $L(G) = \{a^m b^n \mid m, n \geq 0\}$

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:

$$
\begin{array}{llcl}
r1. & S & \to & QNQ \\
r2. & QN & \to & QR \\
r3. & RN & \to & NNR \\
r4. & RQ & \to & NNQ \\
r5. & N & \to & a \\
r6. & Q & \to & \epsilon
\end{array}
$$

Then:

- $S \Rightarrow^{r1} QNQ \Rightarrow^{r2} QRQ \Rightarrow^{r4} QNNQ \Rightarrow^{r2} QRNQ \Rightarrow^{r3} QNNRQ \Rightarrow^{r4} QNNNNQ \Rightarrow^{*} aaaa$
- $L(G) = \{a^{(2^n)} \mid n \geq 0\}$

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 $\alpha \in V^* \cdot V_N \cdot V^*$ 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. $\alpha A\gamma \to \alpha\beta\gamma$
with $\alpha, \gamma \in V^*$, $\beta \in V^+$ and $A \in V_N$.
Furthermore, a rule of the following form is allowed:
$S \to \epsilon$
if 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 determines 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, 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 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:

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

- **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 conditions, where $A, B \in V_N$ and $a \in V_T$:

Type 3. $A \to aB$, or $A \to a$
Furthermore, a rule of the following form is allowed:
$S \to \epsilon$
if S does not appear on the right side of any rule.

- The above define the *Right-Regular Grammars*. The following Productions:
  $A \to Ba$, or $A \to a$
  define *Left-Regular Grammars*.
- Right-Regular and Left-Regular Grammars define the same set of Languages.

Type 3, Regular Grammars (cont.)

- 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 $w \in L(\mathrm{G})$.

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 $A \in V_N$;
  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:
  $$A \to A_1, A_2, \ldots, A_k$$
  must be a production in P.

Derivation Trees: An Example

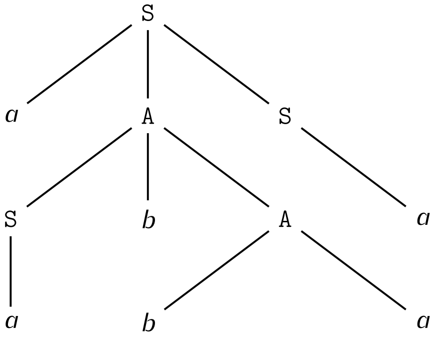
Example. Let G=($\{a, b\}, \{S, A\}, S,$ P), where P is:

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

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

$A \rightarrow SS$

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 *sentence*, called also the *result* of the derivation tree.
- **Theorem.** Let G= (V_T, V_N, S, P) be 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* α.

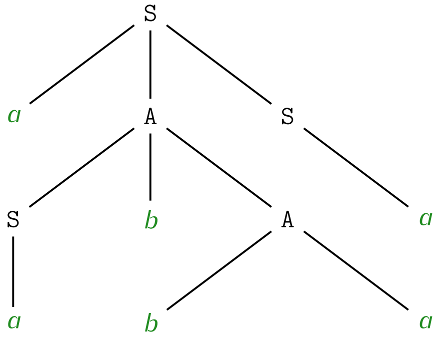
Derivation Trees: An Example (Cont.)

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: $aabbaa$. 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.