# Formal Languages And Compilers *Theory*

Christian Rossi

Academic Year 2023-2024

**Abstract**

The lectures are about those topics:

- Definition of language, theory of formal languages, language operations, regular expressions, regular languages, finite deterministic and non-deterministic automata, BMC and Berry-Sethi algorithms, properties of the families of regular languages, nested lists and regular languages.
- Context-free grammars, context-free languages, syntax trees, grammar ambiguity, grammars of regular languages, properties of the families of context-free languages, main syntactic structures and limitations of the context-free languages.
- Analysis and recognition (parsing) of phrases, parsing algorithms and automata, push down automata, deterministic languages, bottom-up and recursive top-down syntactic analysis, complexity of recognition.
- Syntax-driven translation, direct and inverse translation, syntactic translation schemata, transducer automata, and syntactic analysis and translation. Definition of semantics and semantic properties. Static flow analysis of programs. Semantic translation driven by syntax, semantic functions and attribute grammars, one-pass and multiple-pass computation of the attributes.

The laboratory sessions are about those topics:

- Modellization of the lexicon and the syntax of a simple programming language (C-like).
- Design of a compiler for translation into an intermediate executable machine language (for a register-based processor).
- Use of the automated programming tools Flex and Bison for the construction of syntax-driven lexical and syntactic analyzers and translators.

# Contents

# Chapter 1

# Regular Languages

### 1.0.1 Formal language theory

A *formal language* consists of words whose letters are taken from an alphabet and are well-formed according to a specific set of rules.

**Definition**

An *alphabet* is a finite set of elements called terminal symbols or *characters*. The *cardinality* of an alphabet

$$\Sigma = \{a_1, a_2, \dots, a_k\}$$

is the number of characters that it contains: $|\Sigma| = k$. A *string* or word is a sequence of characters.

**Example :** The alphabet $\Sigma = \{a, b\}$ has a cardinality of two. Some possible languages derived from this alphabet can be:

- $L_1 = \{aa, aaa\}$
- $L_2 = \{aba, aab\}$
- $L_3 = \{ab, ba, aabb, abab, \dots, aaabbb, \dots\}$

**Definition**

Given a language, a string belonging to it is called a *sentence* or *phrase*. The *cardinality* or size of a language is the number of sentence it contains. If the cardinality is finite, the language is called *vocabulary*.

**Example :** Given the language (that is a vocabulary) $L_2 = \{bc, bbc\}$ we have that its cardinality is equal to two.

**Definition**

The number of repetitions of a certain letter in a word is called *number of occurrences*. The *length* of a string is the number of its elements. Two strings are *equal* if and only if:

- They have the same length.
- Their elements, from left to right, coincide.

**Example :** The number of occurrences of $a$ and $c$ in $aab$ is indicated with:

$$|aab|_a = 2$$

$$|aab|_c = 0$$

The length of the string $aab$ is equal to:

$$|aab| = 3$$

### 1.0.2 Operations on strings

**Operation (*Concatenation*)**

Given two strings $x = a_1a_2 \dots a_h$ and $y = b_1b_2 \dots b_k$ the *concatenation* is defined as:

$$x \cdot y = a_1a_2 \dots a_hb_1b_2 \dots b_k$$

Concatenation is non-commutative and associative ($x(yz) = (xy)z$). The length of the result is the sum of the length of the concatenated strings ($|xy| = |x| + |y|$).

**Operation (*Empty string*)**

The *empty string* $\varepsilon$ is the neutral element for concatenation that satisfies the identity:

$$x\varepsilon = \varepsilon x = x$$

It is important to note that $|\varepsilon| = 0$ and that the set that contains this operator is not the empty set.

**Operation (*Substring*)**

Let string $x = xyv$ be written as the concatenation of three, possibly empty, strings $x, y$ and $v$. Then, strings $x, y$ and $v$ are *substrings* of $x$.

Moreover, string $u$ is a prefix of $x$ and $v$ is a suffix of $x$. A non-empty substring is called proper if it does not coincide with string $x$.

**Operation (*Reflection*)**

The *reflection* of a string $x = a_1 a_2 \dots a_h$ is:

$$x^R = a_h a_{h-1} \dots a_1$$

The following identities are immediate:

$$(x^R)^R = x \quad (xy)^R = y^R x^R \quad \varepsilon^R = \varepsilon$$

**Operation (*Repetition*)**

The *repetition* is the $m$-th power $x^m$ of a string $x$ is the concatenation of $x$ with himself $m-1$ times. The formal definition o the following:

$$x^m = x^{m-1}x \ \ form \geq 1 \quad x^0 = \varepsilon$$

Repetition and reflection take precedence over concatenation.

### 1.0.3 Operations on languages

Operations are typically defined on a language by extending the string operation to all its phrases.

**Operation (*Reflection*)**

The *reflection* $L^R$ of a language $L$ is the finite set of strings that are the reflection of a sentence of $L$:

$$L^R = \{x | \exists y \left( y \in L \land x = y^R \right) \}$$

**Operation (*Prefix*)**

The set of *prefixes* of a language $L$ is defined as:

$$Prefixes(L) = \{y | y \neq \varepsilon \land \exists x \exists z \left( x \in L \land x = yx \land z \neq \varepsilon \right) \}$$

A language is prefix-free if none of the proper prefixes of its sentences is in the language.

**Operation (*Concatenation*)**

Given languages $L^{'}$ and $L^{''}$ we have that *concatenation* is defined as:

$$L^{'}L^{''} = \{xy | x \in L^{'} \land y \in L^{''} \}$$

**Operation (*Repetition*)**

The *repetition* is redefined as:

$$L^m = L^{m-1}L \; for\; m \geq 1 \qquad L^0 = \{\varepsilon\}$$

The identity now became:

$$\varnothing^0 = \{\varepsilon\} \qquad L.\varnothing = \varnothing.L = \varnothing \qquad L.\{\varepsilon\} = \{\varepsilon\}.L = L$$

The power operator allows one to define concisely the language of strings whose length is not greater than a given integer $K$.

**Operation (*Set operations*)**

Since a language is a set, the classical set operation of union ($\cup$), intersection ($\cap$), difference ($\setminus$), inclusion ($\subseteq$), strict inclusion ($\subset$), and equality ($=$).

**Operation (*Universal language*)**

The *universal language* is defined as the set of all the strings, over an alphabet $\Sigma$, of any length including zero:

$$L_{universal} = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

**Operation (*Complement*)**

The *complement* of a language $L$ over an alphabet $\Sigma$, denoted by $\neg L$, is the set difference:

$$\neg L = L_{universal} - L$$

That is, the set of the strings over the alphabet $\Sigma$ that are not in $L$. Note that:

$$L_{universal} = \neg\varnothing$$

The complement of a finite language is always infinite. The complement of an infinite one is not necessarily finite.

Given a set A and a relation $R \subseteq A \times A$, $(a_1, a_2) \in R$ is also denoted as $a_1 R a_2$. $R^*$ is a relation defined by:

- $xR^*x \; \forall x \in A$ (reflexive property).
- $x_1 R x_2 \wedge x_2 R x_3 \wedge \dots x_{n-1} R x_n \implies x_1 R^* x_n$ (transitive property).

**Example:** Given $R = \{(a,b), (b,c)\}$, the transitive closure will be:

$$R^* = \{(a,a), (b,b), (c,c), (a,b), (b,c), (a,c)\}$$

Given a set A and a relation $R \subseteq A \times A$, $(a_1, a_2) \in R$ is also denoted as $a_1 R a_2$. $R^+$ is a relation defined by: $x_1 R x_2 \wedge x_2 R x_3 \wedge \ldots x_{n-1} R x_n \implies x_1 R^* x_n$ (transitive property).

**Example:** Given $R = \{(a, b), (b, c)\}$, the transitive closure will be:

$$R^+ = \{(a, b), (b, c), (a, c)\}$$

**Operation (*Star operator*)**

The *star operator* (also called Kleene star) is the reflexive transitive closure under the concatenation operation. It is defined as the union of all the powers of the base language:

$$L^* = \bigcup_{h=0\ldots\infty} L^h = L^0 \cup L^1 \cup L^2 \cup \cdots = \varepsilon \cup L^1 \cup L^2 \cup \ldots$$

**Example:** Given the language $L = \{ab, ba\}$ we have that the star operation gives the following language:

$$L^* = \{\varepsilon, ab, ba, abab, abba, baab, baba, \ldots\}$$

It is possible to see that $L$ is finite and $L^*$ is infinite.

Every string of the star language $L^*$ can be chopped into substrings in $L$. The star language $L^*$ can be equal to the base language $L$. If we take $\Sigma$ as the base language, then $\Sigma^*$ contains all the strings built on that alphabet (it is the universal language of alphabet $\Sigma$). We often say that $L$ is a language on alphabet $\Sigma$ by writing $L \subseteq \Sigma$.

| **Property** | **Meaning** |
|---|---|
| $L \subseteq L^*$ | Monotonicity |
| if $x \in L^* \wedge y \in L^*$ then $xy \in L^*$ | Closure by concatenation |
| $(L^*)^* = L^*$ | Idempotence |
| $(L^*)^R = (L^R)^*$ | Commutativity with reversal |

Furthermore, if $L^*$ is finite we have $\varnothing^* = \{\varepsilon\}$ and that $\{\varepsilon\}^* = \{\varepsilon\}$.

**Operation (*Cross operator*)**

The *cross operator* is the transitive closure under the concatenation operation. It is defined as the union of all the powers of the base language except the first power $L^0$:

$$L^+ = \bigcup_{h=1\dots\infty} L^h = L^1 \cup L^2 \cup \dots$$

**Example :** Given the language $L = \{ab, ba\}$ we have that the star operation gives the following language:

$$L^* = \{ab, ba, abab, abba, baab, baba, \dots\}$$

**Operation (*Language quotient*)**

The *quotient operator* shortens the phrases of $L_1$ by cutting off a suffix that belongs to $L_2$:

$$L = L_1/L_2 = \{y | \exists x \in L_1 \exists y \in L_2 (x = yz)\}$$

**Example :** Given the languages $L_1 = \{a^{2n}b^{2n} | n > 0\}$ and $L_2 = \{b^{2n+1} | n \geq 0\}$ the quotient language is:

$$L = L_1/L_2 = \{aab, aaaab, aaaabbb\}$$

### 1.0.4 Regular expressions and languages

The family of regular languages is our simplest formal language family. It can be defined in three ways: algebraically, by means of generative grammars, and by means of recognizer automata.

**Definition**

A *regular expression* is a string $r$ containing the terminal characters of the alphabet $\Sigma$ and the following meta-symbols: union ($\cup$), concatenation (.), star (*), empty string ($\varepsilon$), and parenthesis in accordance with the following rules:

| | |
|---|---|
| $r = \varepsilon$ | Empty string |
| $r = a$ | Unitary language |
| $r = s \cup t$ | Union of expressions |
| $r = (st)$ | Concatenation of expressions |
| $r = s^*$ | Iteration of an expression |

where the symbols $s$ and $t$ are regular sub-expression.

For expressivity, the metasymbol cross is allowed. The operators precedence is: star, concatenation, and union.

**Definition**

A *regular language* is a language denoted by a regular expression.

The *family of regular languages* (REG) is the collection of all regular languages.

The *family of finite languages* (FIN) is the collection of all languages having a finite cardinality

We have that every finite language is regular because it is the union of a finite number of strings each one being the concatenation of a finite number of alphabet symbols. The family of regular languages also includes languages having infinite cardinality (hence $FIN \subset REG$)

The union and repetition operators correspond to possible choices. One obtains a sub-expression by making a choice that identifies a sub-language. Given a regular expression one can derive another one by replacing any outermost sub-expression with another that is a choice of it.

**Definition**

We say that a regular expression $e^{'}$ *derives* a regular expression $e^{''}$, written $e^{'} \implies e^{''}$, if the two regular expressions can be factorized as

$$e^{'} = \alpha\beta\gamma \quad e^{''} = \alpha\delta\gamma$$

where $\delta$ is a choice of $\beta$.

The derivation relation can be applied repeatedly, yielding relation $\implies^{n}$ ($n$ steps), $\implies^{*}$ ($n \geq 0$ steps), and $\implies^{+}$ ($n > 0$ steps).

**Definition**

Two regular expressions are *equivalent* if they define the same language.

A regular expression is *ambiguous* if the language of the numbered version $f^{'}$ includes two distinct strings $x$ and $y$ that coincide when numbers are erased.

# Chapter 2

# Grammars

## 2.1 Context-free generative grammars

Regular expressions are very practical for describing lists but fall short of the capacity needed to define other frequently occurring constructs. For defining other useful languages, regular or not, we move to the formal model of generative grammars. A generative grammar or syntax is a set of multiple rules that can be repeatedly applied in order to generate all and only the valid strings.

**Definition**

A *context-free grammar* $G$ is defined by four entities:

1. $V$ non-terminal alphabet, is the set of non-terminal symbols.
2. $\Sigma$ terminal alphabet, is the set of the symbols of which phrases or sentences are made.
3. $P$ is the set of rules or productions.
4. $S \in V$ is the specific non-terminal, called the axiom ($S$), from which derivations start.

A rule of set $P$ is an order pair $X \to \alpha$, with $X \in V$ and $\alpha \in (V \cup \Sigma)^*$. Two or more rules:

$$X \to \alpha_1 \quad X \to \alpha_2 \quad \dots \quad X \to \alpha_n$$

with the same left part $X$ can be concisely groped in:

$$X \to \alpha_1 | \alpha_2 | \dots | \alpha_n$$

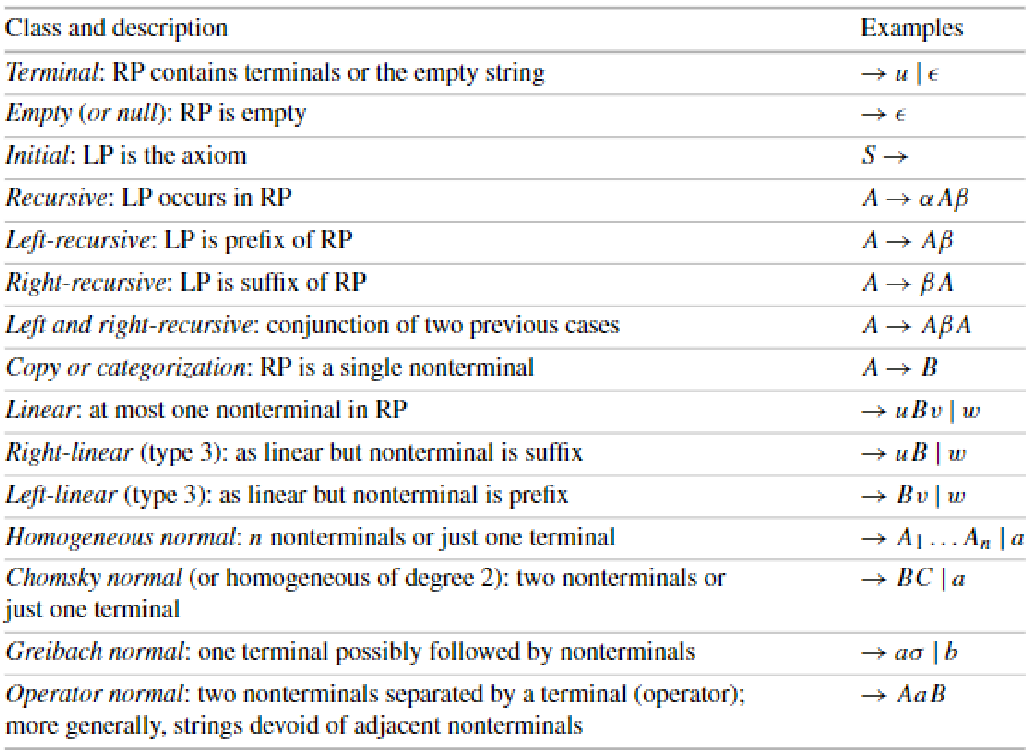
We say that the strings $\alpha_1, \alpha_2, \dots, \alpha_n$ are the alternative of $X$.

### 2.1.1 Conventional grammar representation

In professional practice, different styles are used to represent terminals and non-terminals. We usually adopt these conventions:

- Lowercase Latin letters $\{a, b, \dots\}$ for terminal characters.
- Uppercase Latin letters $\{A, B, \dots\}$ for non-terminal symbols.
- Lowercase Latin letters $\{r, s, \dots, z\}$ for strings over the alphabet $\Sigma$.
- Lowercase Greek letters $\{r, s, \dots, z\}$ for both terminals and non.
- $\sigma$ only for non-terminals.

The classification of grammar rule forms is the following.



| Class and description | Examples |
|---|---|
| *Terminal*: RP contains terminals or the empty string | $\to u \mid \epsilon$ |
| *Empty (or null)*: RP is empty | $\to \epsilon$ |
| *Initial*: LP is the axiom | $S \to$ |
| *Recursive*: LP occurs in RP | $A \to \alpha A \beta$ |
| *Left-recursive*: LP is prefix of RP | $A \to A\beta$ |
| *Right-recursive*: LP is suffix of RP | $A \to \beta A$ |
| *Left and right-recursive*: conjunction of two previous cases | $A \to A\beta A$ |
| *Copy or categorization*: RP is a single nonterminal | $A \to B$ |
| *Linear*: at most one nonterminal in RP | $\to uBv \mid w$ |
| *Right-linear* (type 3): as linear but nonterminal is suffix | $\to uB \mid w$ |
| *Left-linear* (type 3): as linear but nonterminal is prefix | $\to Bv \mid w$ |
| *Homogeneous normal*: $n$ nonterminals or just one terminal | $\to A_1 \dots A_n \mid a$ |
| *Chomsky normal* (or homogeneous of degree 2): two nonterminals or just one terminal | $\to BC \mid a$ |
| *Greibach normal*: one terminal possibly followed by nonterminals | $\to a\sigma \mid b$ |
| *Operator normal*: two nonterminals separated by a terminal (operator); more generally, strings devoid of adjacent nonterminals | $\to AaB$ |

### 2.1.2 Derivation and Language Generation

We reconsider and formalize the notion of string derivation. Let $\beta = \delta A \eta$ be a string containing a non-terminal, where $\delta$ and $\eta$ are any, possibly empty strings. Let $A \to \alpha$ be a rule of $G$ and let $\gamma = \delta \alpha \eta$ be the string obtained replacing in $\beta$ non-terminal $A$with the right part $\alpha$. The relation between such two strings is called derivation. We say that $\beta$ derives $\gamma$ for grammar $G$, written:

$$\beta \implies \gamma$$

$A \to \alpha$ is applied in such derivation and string $\alpha$ reduced to non-terminal $A$. The possible closures are: power ( $\implies {}^n$), reflexive ( $\implies {}^*$), and transitive ( $\implies {}^+$).

**Definition**

If $A \implies {}^*\alpha$ we have that $\alpha \in (V \cup \Sigma)$ is called *string form* generated by $G$.

If $S \implies {}^*\alpha$ we have that $\alpha$ is called *sentential* or phrase form.

If $A \implies {}^*s$ we have that $s \in \Sigma^*$ is called *phrase* or sentence.

Language is *context-free* if a context-free grammar exists that generates it.

Two grammars $G$ and $G'$ are *equivalent* if they generate the same language.

### 2.1.3 Erroneous grammars and useless rules

When writing a grammar attention should be paid that all non-terminals are defined and that each one effectively contributes to the production of some sentence. In fact, some rules may turn out to be unproductive.

**Definition**

A grammar $G$ is called *clean* (or reduced) under the following conditions:

1. Every non-terminal $A$ is reachable from the axiom.

2. Every non-terminal $A$ is well-defined.

It is often straightforward to check by inspection whether a grammar is clean. The following algorithm formalizes the checks. The algorithm operates in two phases, first pinpointing the undefined non-terminals, then the unreachable ones. Lastly the rules containing non-terminals of either type can be canceled. The phases are:

1. Compute the set $DEF \subseteq V$ of well-defined non-terminals. The set $DEF$ is initialized with the non-terminals of terminal rules, those having a terminal string as right part:

$$DEF := \{A | (A \to u) \in P, with u \in \Sigma^*\}$$

Then the next transformation is applied until convergence is reached:

$$DEF := DEF \cup B|(B \to D_1 D_2 \dots D_n) \in P$$

where every $D_i$ is a terminal or a non-terminal symbol present in $DEF$. At each iteration two outcomes are possible:

- A new non non-terminal is found having as right part a string of symbols that are well-defined non-terminals or terminals.
- The termination condition is reached

The non-terminals belonging to the complement set $V - DEF$ are undefined and should be eliminated.

2. A non-terminal is reachable from the axiom, if, and only if, there exists a path in the following graph, which represents a relation between non-terminals, called product:

$$A \to^{produce} B$$

saying that $A$ produces $B$ if, and only if, there exists a rule $A \to \alpha B \beta$, where $A, B$ are non-terminals and $\alpha, \beta$ are any strings. Clearly $C$ is reachable from $S$ if, and only if, in this graph there exists an oriented path from $S$ to $C$. The unreachable non-terminals are the complement with respect to $V$. They should be eliminated because they do not contribute to the generation of any sentence.

Quite often the following requirement is added to the above clearness conditions: $G$ should not permit circular deviations $A \implies {}^{+}A$. This is done to avoid ambiguity. We observe that a grammar, although clean, may still contain redundant rules.

### 2.1.4 Recursion and language infinity

An essential property of most technical languages is to be infinite. We study how this property follows from the form of grammar rules. In order to generate an unbound number of strings, the grammar must be able to derive strings of unbound length. To this end, recursive rules are necessary, as next argued. An $n \geq 1$ steps derivation $A \implies {}^{n}xAy$ is called recursive (immediately recursive if $n = 1$); similarly non-terminal $A$ is called recursive. If $x$ is empty, the recursion is termed left.

Let $G$ be a grammar clean and avoid of circular deviations. The language $L(G)$ is infinite if, and only if, $G$ has a recursive derivation.

### 2.1.5 Syntax trees and canonical derivations

**Definition**

A *tree* is an oriented and ordered graph not containing a circuit, such that every pair of nodes is connected by exactly one oriented path.

An *arc* $\langle N_1, N_2 \rangle$ define the $\langle$father,son$\rangle$ relation, customarily visualized from top to bottom as in genealogical trees. The sides of a node are ordered from left to right.

The *degree* of a node is the number of its siblings.

A *tree* contains one node without father, termed root.

Consider an internal node $N$: the subtree with root $N$ is the tree having $N$ as root and containing all descendants of $N$. Nodes without sibling are termed leaves or *terminal nodes*.

The sequence of all leaves, read from left to right, is the *frontier* of the tree.

A *syntax tree* has as root the axiom and as frontier a sentence.

A syntax tree of a sentence $x$ can also be encoded in a text, by enclosing each subtree between brackets. Brackets are subscribed with the non-terminal symbol. The representation can be simplified by dropping the non-terminal labels, thus obtaining a skeleton tree. A further simplification of the skeleton tree consists in shortening non bifurcating paths, resulting in the condensed skeleton tree.

### 2.1.6 Left and right derivations

We can have right (expands at each step the rightmost non-terminal) and left derivation (expands at each step the leftmost non-terminal). However, for a fixed syntax tree of a sentence, there exist a unique right derivation, and a unique left derivation matching that tree. Right and left derivation are useful to define parsing algorithms.

### 2.1.7 Parenthesis languages

Many artificial languages include parenthesized or nested structures, made by matching pairs of opening/closing marks. Any such occurrence may contain other matching pairs. The marks are abstract elements that have different concrete representations indistinct settings.

**Definition**

When a marked construct may contain another construct of the same kind, it is called *self-nested*.

Self-nesting is potentially unbounded in artificial languages, whereas in natural languages its use is moderate, because it causes difficulty of comprehension by breaking the flow of discourse. Abstracting from concrete representation and content, this paradigm is known as a Dyck language. The terminal alphabet contains one or more pairs of opening/closing marks. Dyck sentences are characterized by the following cancelation rule that checks parentheses are well nested: given a string, repeatedly substitute the empty string for a pair of adjacent matching parentheses:

$$[\,] \Longrightarrow \varepsilon \quad (\,) \Longrightarrow \varepsilon$$

Thus obtaining another string. Repeat until the transformation no longer applies; the original string is correct if, and only if, the last string is empty.

**Definition**

Let $G = (V, \Sigma, P, S)$ be a grammar with an alphabet $\Sigma$ not containing parentheses. The *parenthesized grammar* $G_p$ has alphabet $\Sigma \cup \{'(', ')'\}$ and rules:

$$A \to (\alpha) \text{ where } A \to (\alpha) \text{ is a rule of } G$$

The grammar is distinctly parenthesized if every rule has form:

$$A \to ({}_A\alpha)_A \quad B \to ({}_B\alpha)_B$$

where $({}_A$ and $)_A$ are parentheses subscripted with the non-terminal name.

Clearly each sentence produced by such grammars exhibits parenthesized structure. A notable effect of the presence of parentheses is to allow a simpler checking of string correctness.