Two-Level Grammars: Some Interesting Properties of van Wijngaarden Grammars

Luis M. AUGUSTO*

Editor-in-Chief

Ω ::= Journal of Formal Languages

January 2023 Volume 1

Abstract

The van Wijngaarden grammars are two-level grammars that present many interesting properties. In the present article I elaborate on six of these properties, to wit, (i) their being constituted by two grammars, (ii) their ability to generate (possibly infinitely many) strict languages and their own metalanguage, (iii) their context-sensitivity, (iv) their high descriptive power, (v) their productivity, or the ability to generate an infinite number of production rules, and (vi) their equivalence with the unrestricted, or Type-0, Chomsky grammars.

Key words: Two-Level Grammar; van Wijngaarden Grammar; Deep Grammar and Surface Grammar; Strict Language and Strict Metalanguage; Descriptive Power; Productivity

1 Introduction

Formal languages are at the heart of a large plethora of computing applications, such as programming, compiling, etc. As is well known, formal languages are generated by formal grammars, which are essentially rewriting systems. *Two-level grammars* are formal grammars that are a combination $\frac{G_2}{G_1} = \frac{\text{Level 2}}{\text{Level 1}}$ of two grammars G_1 and G_2 in two levels, lower and upper, respectively denoted by G^1 and G^2; grammars G_1 and G_2 can be of the same or of different type in the (extended) Chomsky hierarchy. Many central computational applications in both theory and practice have been found for these grammars (e.g., Bryant & Lee, 2002; Bryant et al., 1989; Edupuganty & Bryant, 1989; Kupka, 1980; Małuszyńsky, 1984; Visser, 1997).

*✉ luis.ml.augusto@gmail.com

There are several kinds of two-level grammars, all of which share similar formal properties, parsing problems, and implementation techniques (Koster, 1993). Among these, the *van Wijngaarden grammars* (abbr.: WGs), grammars of two-level type $\frac{\text{CFG}}{\text{CFG}}$, present many interesting properties, of which I explore here six, to wit, (i) their being equipped with a *deep grammar* and a *surface grammar*, (ii) their ability to generate (infinitely) many *strict languages* and their own *strict metalanguage*, (iii) their *context-sensitivity*, (iv) their *high descriptive power*, (v) their *productivity*, or the ability to generate an *infinite number of production rules*, and (vi) their *equivalence* with the *unrestricted*, or *type-0 grammars*. Thus, although these grammars do not belong to the Chomsky hierarchy they capture essential properties of the context-free grammars (CFGs; type 2), the context-sensitive grammars (CSGs; type 1), and the unrestricted grammars (UGs; type 0).[1] The class **WL** of the van Wijngaarden languages properly includes all the languages in the Chomsky hierarchy, so that we have

$$\underbrace{\mathbf{RGL} \subset \mathbf{CFL} \subset \mathbf{CSL} \subset \mathbf{REL}}_{\text{Chomsky Hierarchy}} \subset \mathbf{WL}$$

where the Chomsky classes of regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages are respectively denoted by **RGL**, **CFL**, **CSL**, and **REL**. However, the WGs are *not* members of the Chomsky grammar hierarchy, so that this language-inclusion relation naturally raises our interest, namely with respect to their syntactic properties.[2]

WGs were invented by A. van Wijngaarden (1965; 1969) as the syntax and semantics of the programming language ALGOL 68.[3] Despite their many interesting properties the literature on WGs is not abundant and their study has not been as constant and consistent as that of other formal grammars, namely the CFGs; in effect, it seems to have (almost) ceased around 1995. Their equivalence with the UGs, established early on in Sintzoff (1967), makes them a hard subject to study, and this scarcity and difficulty is what motivates this review article. It is hoped that the reader will be convinced of both the interesting properties of WGs and their various (potential) applications in classical computing, in particular in programming:[4] For instance, Edupuganty & Bryant (1989) listed "readability, implementability, referential transparency, and the capability for data and procedure abstraction" as the primary advantages of WG-based programs over other programming languages, and Grune & Jacobs (2008) wrote with respect to WG-based programming that if it "becomes a

[1] Other abbreviations common in the literature are "W-grammar," "VW-grammar," and even "TLG," because two-level grammars are frequently identified with WGs. Although relatively scarce, the literature on WGs presents many variations to the notation and even to the terminology, making it even more difficult to approach these grammars. Aiming at uniformity, I adopt, with minor adaptations, the original (not intuitive) notation and terminology in van Wijngaarden et al. (1976) and Cleaveland & Uzgalis (1977). These two references provide treatments of WGs from the programming viewpoint; in this review I focus on the strictly formal-grammar perspective. Familiarity with the basic aspects of the theory of formal languages, including the Chomsky hierarchy, is assumed; Augusto (2021) provides a treatment of this subject clearly centered in the Chomsky hierarchy.

[2] De Graaf & Ollongren (1984) show that the two-level type $\frac{\text{CFG}}{\text{RGG}}$, where "RGG" abbreviates "regular grammar" (Type 3), is sufficiently general to generate each member of **WL**.

[3] See Lindsey (1996). I refer the reader interested in programming with ALGOL 68 to Lindsey & van der Meulen (1977). A recent extension and implementation of ALGOL 68 called *Algol 68 Genie* is available at [1].

[4] See Augusto (2021) for the meaning of the expression "classical computing."

full-fledged paradigm, we will no doubt find the style presented here as archaic as we find today machine code of the 1950s." I add that a good grasping of these grammars can help us to understand better the CSGs and the UGs, thus potentially contributing to the design of (more efficient) compiling algorithms for the languages generated by them; in effect, compiling the CSLs is still a difficult and inefficient process, and no compilation algorithms have been devised for the RELs. All these objectives are expected to pay off for the extra effort of learning the verbose terminology and the highly complex syntax of the WGs.

The aim of a review article is rarely that of providing novel results; in accordance with this typology, I largely restrict the discussion to already published contents.[5] However, I introduce novel or adapted concepts and formal definitions (e.g., surface vs. deep grammar/language; strict metalanguage) that in my view contribute to a better systematization of these grammars and foster thus their further development.

2 The Syntax of van Wijngaarden Grammars

As is well known, formal languages have two main components to them, to wit, syntax and semantics, and they can be approached via accepting systems (typically automata) and/or generating systems (typically grammars). The latter approach is adopted here, with only the odd reference to automata, and the focus of this work is on the syntax of the WGs: This review falls on the syntactic aspects of these grammars that account for the language-inclusion relation above. The WGs have relations with many other classes of grammars, but here they will be put into relation only with the class of the Chomsky grammars, i.e. the formal grammars in the Chomsky hierarchy.

Recall that a Chomsky grammar is a 4-tuple $G = (V, T, S, R)$, where $V \cup T = \Sigma \neq \emptyset$ for the finite disjoint sets V and T of variable symbols and terminal symbols, respectively, is called the alphabet, $S \in V$ is the start symbol, and R is a finite set of production rules r_P of the form $\alpha \to \beta$, read "α is rewritten as β," where α is the left-hand side of the rule (abbr.: LHS), $\alpha \in (V \cup T)^+$, $|V(\alpha)| \geq 1$, and $\beta \in (V \cup T)^* = \left((V \cup T)^+ \cup \{\epsilon\}\right)$ is the right-hand side of the rule (RHS), so that the empty symbol "ϵ" is allowed only on the RHS. Restrictions apply to this general, unrestricted, type-0 definition for types 1 - 3 of the Chomsky hierarchy.

By a slight abuse of terminology, G is also called a generative system and the language L generated by G is defined as

$$L(G) = \left\{ w \in T^* | S \overset{*}{\underset{G}{\Longrightarrow}} w \right\}$$

where "$w = t_1 t_2 ... t_k$" denotes a string of terminal symbols (the concatenation of the terminal symbols $t_1, t_2, ..., t_k$) called word, "$\Longrightarrow$" denotes a derivation step such that we have $S \overset{n}{\underset{G}{\Longrightarrow}} \vartheta$ for the sentential form $\vartheta \in (V \cup T)^*$ in n derivation steps, and "*" (Kleene star) denotes the reflexive and transitive closure of the relation $\underset{G}{\Longrightarrow} \subseteq (V \cup T)^+$. A derivation $S \overset{*}{\underset{G}{\Longrightarrow}} w$ in G can be leftmost (denoted by $S \overset{*}{\underset{G}{\Longrightarrow}}_l w$) or

[5] Cleaveland & Uzgalis (1977), in particular, is a major source for this article. "Old" as it is, this book remains an essential source for the study of WGs, even if their notion of the two levels differs largely from that of the standard literature (see below).

rightmost ($S \overset{*}{\underset{G}{\Longrightarrow}}_r w$) depending on whether the leftmost or the rightmost variables, respectively, are rewritten at each step of the derivation. When applied over an arbitrary set of symbols Σ, the Kleene star denotes the set of all strings – including the empty string ϵ – over Σ; this set is denoted by Σ^*.[6] Another, less abstract, way to define a language in set-theoretic terms is as

$$L(G) = \{w \in T^* | w \text{ satisfies } \mathcal{P}\}$$

where $\mathcal{P}$ is some property (e.g., no letter of the Roman alphabet occurs more than once).

A Chomsky grammar $G = (V, T, S, R)$ where for strings $\alpha, \beta \in \Sigma^*$ we have $\alpha \to \beta$ if there exist strings $x, y, u, v \in \Sigma^*$ such that $\alpha = xuy$, $\beta = xvy$, is equivalent to a string rewriting system, or semi-Thue system $\Gamma = (\Sigma^*, R)$ where Σ is (usually) finite and R is a set of binary relations $\to \subseteq \Sigma^* \times \Sigma^*$.

2.1 Syntactic Marks, Notions, and Typographical Characters

To avoid confusion, a formal grammar G that is a WG will be denoted by $\mathcal{G}$. Contrasting with the above simplicity of a Chomsky grammar G, a WG $\mathcal{G}$ is a 7-tuple whose components are not so straightforwardly defined; this requires some preliminary definitions and notation.[7] To begin with, a WG uses sequences of symbols called *syntactic marks*.

Definition 1. The syntactic marks of a WG are of three types:

1. *Small syntactic marks*, written as “a”, “b”, “c”, ..., “y”, “z”, “(”, “)”.
2. *Large syntactic marks*, written as “A”, “B”, “C”, ..., “Y”, “Z”, “0”, “1”, ..., “9”.
3. *Other syntactic marks*, written as “.” (“point”), “,” (“comma”), “:” (“colon”), “;” (“semicolon”), “ ’ ” (“apostrophe”), “-” (“hyphen”), “*” (“asterisk”).

The *empty syntactic mark* is denoted by ϵ. Note that the parentheses – “(” and “)” – are considered small syntactic marks, not being included in the class of other syntactic marks. The digits “0”, “1”, ..., “9” are considered large syntactic marks, but they can also be written with small syntactic marks (“zero”, “one”, ..., “nine”).

Instead of symbols a WG $\mathcal{G}$ uses finite sequences $\overline{\sigma} = \sigma_1\sigma_2...\sigma_n$ of syntactic marks where σ_i may belong to any of the three types above and there may be blank spaces between any σ_i and σ_j, $i < j$ is a chain, i.e. there is no x such that $i < x < j$. Just as in a Chomsky grammar G, a WG $\mathcal{G}$ has a set V of variables (also called protovariables) and a set T of terminals, but these do not exhaust the sets of sequences of syntactic marks of a WG $\mathcal{G}$.[8]

[6] Recall that more generally for some set S of strings we have $S^0 := \{\epsilon\}$, $S^* := \bigcup_{i \in \mathbb{N}, i \geq 0} S^i$, and $S^+ := \bigcup_{i \in \mathbb{N}, i \geq 1} S^i$, where the concatenation $\underbrace{SS...S}_{i \text{ times}} = S^i$ denotes the i-th power of S such that $S^{i+1} = S^i S$.

[7] See Table 5 at the end of Section 3.2 for a quick reference.

[8] Typically, (single) symbols, such as those used in Definition 2, are used only in abstract rules and formal statements. The term “sequence” is preferable to “string,” because commas (denoting concatenation) and blank spaces are allowed in $\overline{\sigma}$. But terminal sequences of symbols are called words.

Definition 2. The finite sequences $\overline{\sigma}$ of syntactic marks of a WG such that $|\overline{\sigma}| \geq 0$ are called *notions* and their respective sets are defined as follows:

1. A *protonotion* $x \in P$ is a (possibly empty) sequence of small syntactic marks. A non-empty protonotion x ending with 'symbol' is a *terminal* $t \in T$; otherwise, it is a *protovariable* $v \in V$.

2. A *metanotion* $X \in M$ is a non-empty sequence of large syntactic marks. A metanotion ending with 'symbol' is a *terminal metanotion*, or *metaterminal*. The sets of metanotions and metaterminals $M_T \subseteq M$ are respectively defined as
$$M = \{W_1, ..., W_l\}$$
and
$$M_T = \{\text{“X symbol”} | X \in M\}.$$

3. A *hypernotion* $\langle \mathcal{X} \rangle$ is a (possibly empty) sequence of protonotions and/or metanotions. The finite set of non-empty hypernotions N is defined as
$$N = \left\{ \langle X \rangle \, | X \in (M \cup V)^+ \right\}.$$

A metanotion $X \in M$, often denoted by $W_{(i)}$, can be called a *metavariable*, especially to distinguish it from a hypernotion $\langle X \rangle$ that is/has a *protovariable* $v \in V$; in effect, we have $M \cap V = \emptyset$. Protonotions can be either variables or terminals, i.e. $P \supseteq (V \cup T)$; to denote $P \diagdown T$ I shall write P'. From this and the above definitions, it can be inferred that $V \supseteq (P' \cup (N \diagdown M))$. As it will be seen, M_T is not straightforwardly a set of terminals in a full-fledged WG, requiring the application of simultaneous substitutions to the members of M.

Example 3. The string of small syntactic marks 'variable point' is a protonotion, namely the protonotion variable point. 'variable point numeral' is a notion, namely the notion variable point numeral. “INTREAL” is a metanotion, namely the metanotion INTREAL. 'reference to INTREAL' is a hypernotion, namely the hypernotion reference to INTREAL. 'letter a symbol' is a terminal protonotion, namely the typographical character *a*. 'letter ALPHA symbol' is a terminal metanotion, namely the terminal protonotions 'letter a symbol', 'letter b symbol', ..., 'letter z symbol', i.e. the typographical characters *a*, *b*, ..., *z*. The blank spaces and hyphens are added for the sake of readability: 'reference-to-INTREAL' and 'referencetoINTREAL' are two equally admissible ways of writing the hypernotion 'reference to INTREAL'.

The correspondence between the terminal protonotions of a WG and the typographical characters of terminal strings is typically given by a *representation table*. Table 1 gives the most commonly used terminal protonotions in a WG and the corresponding typographical characters.

2.2 Two Description Rules and an *Ad-Hoc* Definition of $L(\mathcal{G})$

As recalled above, a Chomsky grammar G has a single set R of production rules that generate directly the words of a given language $L(G) = \left\{ w \in T^* | S \overset{*}{\underset{G}{\Longrightarrow}} w \right\}$, where the

Table 1: A representation table.

Terminal Notation	**Typographical Character**
colon symbol	:
semicolon symbol	;
comma symbol	,
period symbol	.
becomes symbol	:=
left parenthesis symbol	(
right parenthesis symbol	)
letter a symbol	a
⋮	⋮
letter z symbol	z
digit zero symbol	0
⋮	⋮
digit nine symbol	9

words w are taken in the strictly generative sense as *-step derivations from the start symbol S. As seen, an alternative definition is $L(G) = \{w \in T^* | w \text{ satisfies } \mathcal{P}\}$, but $\mathcal{P}$ is not described in the production rules. This description of $\mathcal{P}$ is a unique feature of WGs: A WG $\mathcal{G}$ has *description rules* in the sense that they are designed first and foremost to *describe* a language $L(\mathcal{G})$, rather than to generate directly the words of $L(\mathcal{G})$– although they can also be used for this end. In the following definitions, it is assumed that there may be more than one RHS for the same LHS, i.e. we may have k rules $\alpha \to \beta_1, ..., \alpha \to \beta_k$, which is abbreviated in a single rule as $\alpha \to \beta_1 \,;\, ... \,;\, \beta_k$ where ";" denotes alternative RHSs and thus replaces the symbol "|" typically used in Chomsky-grammar production rules.

Definition 4. A *metaproduction rule* r_M (abbr.: *metarule*) is a rule of the form

$$X_0 \to X_1; X_2; ...; X_n$$

where $X_0 \in M$, $X_i \in (M \cup V)^*$ for $1 \leq i \leq n$. $R_M \subseteq M \times (M \cup V)^*$ is the finite set of metarules. The *metarule symbol* for "$\to$" is "::" and the semicolon separates the X_i *meta-alternatives*.

Let W denote an arbitrary metanotion. In practical terms, a metarule describes what (alternative) substitutions of $X_i \in \left[(M \cup V)^* = (M \cup P')^*\right]$, $1 \leq i \leq n$, are allowed for $X_0 \in \left(M \supseteq \{W_i\}_{i=1}^{l}\right)$.

Example 5. The following is a metarule:

- INTREAL :: SIZETY integral; SIZETY real.

This metarule specifies that the metanotion "INTREAL" can be substituted by the metanotion "SIZETY" concatenated with either the protonotion 'integral' or the protonotion 'real'.

From this example, it can be easily seen that a metarule consists of the following ordered items: A non-empty sequence W of large syntactic marks on the LHS, two colons (the metaproduction symbol) separating the LHS and the RHS, a (possibly empty) sequence of X_i meta-alternatives constituted by protonotions and/or metanotions separated by semicolons, and a period.[9] Schematically, we have:

$$(r_M) \qquad \underbrace{\mathsf{METANOTION}}_{\text{LHS}} :: \underbrace{X_1; ....; X_n.}_{\text{RHS}}$$

where the RHS actually corresponds to the *definition* of the metanotion on the LHS.

Definition 6. A *hyperproduction rule* r_V (abbr.: *hyper-rule*) is a rule of the form

$$\langle X_0 \rangle \to \overset{\circ}{X}_1; \overset{\circ}{X}_2; ...; \overset{\circ}{X}_m$$

where $\langle X_0 \rangle \in N$ and $\overset{\circ}{X}_i \in \left[(N \cup T \cup \{,\})^* = (M \cup P \cup \{,\})^* \right]$ for $1 \leq i \leq m$, $\overset{\circ}{X}_i = (\gamma \langle \mathcal{X} \rangle \delta)_{i_1} ... (\gamma \langle \mathcal{X} \rangle \delta)_{i_k}$ where $\gamma, \delta \in (T \cup \{,\})^*$. The finite set of hyper-rules is the set $R_V \subseteq N \times (N \cup T)^*$. The *hyper-rule symbol* for "$\to$" is ":" and the semicolon separates the $\overset{\circ}{X}_i$ *hyper-alternatives*.[10]

In practical terms, a hyper-rule describes what (alternative) terminal substitutions on the RHS (i.e. the $\overset{\circ}{X}_i$, $1 \leq i \leq m$) for the LHS (i.e. $\langle X_0 \rangle \in N$) are allowed. Terminal substitutions are either (meta)terminals or protonotions.

Example 7. The following hyper-rule specifies that the hyper-notion 'NOTION sequence' can be substituted either by the metanotion "NOTION" or by the sequence of metanotions and protonotions 'NOTION, NOTION sequence', where the comma denotes concatenation:

- NOTION sequence : NOTION; NOTION, NOTION sequence.

From this example it can be easily seen that the constituents of a hyper-rule are, in order, a non-empty hypernotion on the LHS, a colon (the hyper-rule symbol), and a RHS with a (possibly empty) sequence of hyper-alternatives constituted by (terminal) protonotions and/or (terminal) metanotions (the $\overset{\circ}{X}_i$ possibly containing one or more commas to denote concatenation) separated by semicolons, and a period. Schematically, we have:

$$(r_V) \qquad \underbrace{\mathsf{HyperNotion}}_{\text{LHS}} : \underbrace{\overset{\circ}{X}_1; ....; \overset{\circ}{X}_m.}_{\text{RHS}}$$

Note how distinguishing marks (quotes, apostrophes, or hyphens) are wholly removed from the rules above; they are used only in metanotions and hypernotions that stand for themselves (see Example 3), but we can relax this convention whenever there is

[9] An empty RHS is typically the case only for the metarule for the metanotion "EMPTY", to wit, EMPTY :: .

[10] In the WGs given below, I shall dispense with this formal notation, but the reader should be aware that it is, or variations thereof are, very frequent in the literature.

Table 2: Components of metarules and hyper-rules of a WG.

Rule	**LHS**	**RHS**
$r_M \in R_M$	Metanotion $W \in M$	Meta-alternatives $X_i \in (M \cup V)^*$
$r_V \in R_V$	Hypernotion $\langle X_0 \rangle \in N$	Hyper-alternatives $\overset{\circ}{X}_i \in (N \cup T \cup \{,\})^*$

no risk of ambiguity, the different font and the size of the syntactic marks sufficing for the identification of the kind of notion.[11]

Table 2 summarizes the above. The definitions of a WG $\mathcal{G}$ and of the language $L(\mathcal{G})$ generated by $\mathcal{G}$ can now be given.

Definition 8. A *van Wijngaarden grammar* is a 7-tuple $\mathcal{G} = (M, V, N, T, R_M, R_V, S)$ where the finite sets M, V, N, T, R_M, R_V are as above, $S = \langle s \rangle \in N$ such that $s \in V^+$ is the *start notion*, and *concatenation* is the sole string operation.

This allows for a preliminary *ad-hoc* definition of a language generated by a WG:

Definition 9. Let $\mathcal{G}$ be a WG and let $\overset{*}{\underset{\mathcal{G}}{\Longrightarrow}}$ be the reflexive and transitive closure of the relation $\underset{\mathcal{G}}{\Longrightarrow} \subseteq (N \cup T)^+ = H$. We have the derivation step $\alpha \underset{\mathcal{G}}{\Longrightarrow} \beta$ ($\alpha \underset{\mathcal{G}}{\Longrightarrow}_l \beta$ or $\alpha \underset{\mathcal{G}}{\Longrightarrow}_r \beta$ for a leftmost or rightmost derivation step, respectively) if there is a hyper-rule $r_V j = \left\langle X_{0_j} \right\rangle \rightarrow \overset{\circ}{X}_{1_j}; \overset{\circ}{X}_{2_j}; ...; \overset{\circ}{X}_{m_j}$ and strings $\gamma, \delta \in (N \cup T)^*$ such that $\underbrace{\gamma, \left\langle X_{0_j} \right\rangle, \delta}_{\alpha} \rightarrow \underbrace{\gamma, \overset{\circ}{X}_{i_j}, \delta}_{\beta}$, where the commas denote concatenation. Then, the *hyper-language* generated by $\mathcal{G}$ is defined as:

$$L_H(\mathcal{G}) = \left\{ w \in T^* \subseteq P | \langle s \rangle \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}} w \right\}$$

Leftmost and rightmost derivations, denoted by $\alpha \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}}_l \beta$ and $\alpha \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}}_r \beta$, respectively, can be specified whenever *all* the derivation steps are of the form $\alpha \underset{\mathcal{G}}{\Longrightarrow}_l \beta$ or $\alpha \underset{\mathcal{G}}{\Longrightarrow}_r \beta$.

Example 10. Beginning with a very simple language will prove useful for pedagogical reasons. I shall be working with the WG conveniently called $\mathcal{G}_{Alpha1}$, abbreviated as *Alpha1* whenever there is no risk of confusion. The following rules generate $L_H(Alpha1) = \{t^n | n \geq 1, t = a, b, ..., \text{ or } z\}$:[12]

[11] Strictly, one should write, for instance, 'x' $\in P$ (Def. 2.1) and "W" $\in M$ (Def. 2.2) to denote the facts that x is an arbitrary sequence of small syntactic marks and W is an arbitrary sequence of large syntactic marks. However, this would significantly increase the complexity of the notation and I therefore relax this practice when denoting arbitrary sequences of either small or large syntactic marks.

[12] Abbreviations such as "a; b; c; ...; y; z" should be used with care in programming with WGs, i.e. they should be either altogether avoided or defined in the program.

$(r_M 1)$	ALPHA	::	a; b; c; ...; y; z.
$(r_V 1)$	s	:	ALPHA.
$(r_V 2)$	ALPHA	:	letter ALPHA symbol;
			ALPHA, letter ALPHA symbol.

The derivation of the word $aaa \in L_H\,(Alpha1)$ is as follows:

$$\mathsf{s} \underset{r_V 1}{\Longrightarrow} \mathsf{a} \underset{r_V 2}{\Longrightarrow} \mathsf{a},\ \overbrace{a}^{\text{letter a symbol}} \underset{r_V 2}{\Longrightarrow} \mathsf{a},\ \overbrace{a}^{\text{letter a symbol}}\ a \underset{r_V 2}{\Longrightarrow} \overbrace{a}^{\text{letter a symbol}}\ aa$$

This is obviously a rightmost derivation, denoted by $\mathsf{s} \Longrightarrow_r aaa$; in order to obtain a leftmost derivation, i.e. $\mathsf{s} \Longrightarrow_l aaa$, the two alternatives of $r_V 2$ must change positions in the RHS. Figure 1 shows the parse tree for the word *aaa* generated by *Alpha1*. In this, as in all parse trees of words generated by WGs, the terminals are given in a representation table (see Table 1), reason why terminals can be represented directly in the leaves of the parse tree (Fig. 1, right).[13]

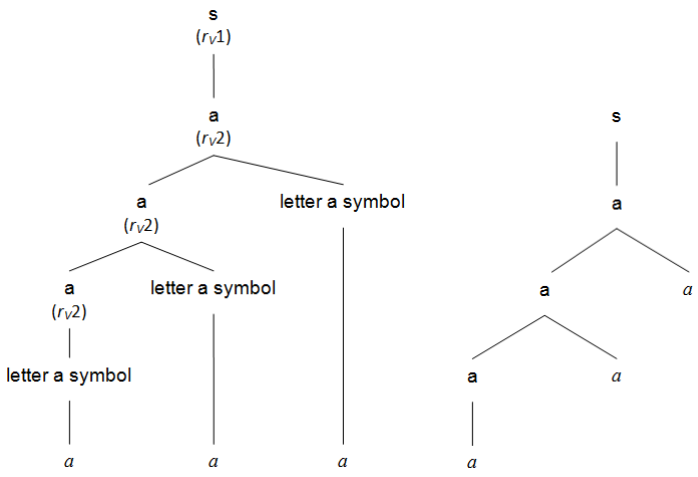

Figure 1: Parse tree for the word *aaa* generated by the WG *Alpha1*. The complete parse tree is on the left and its simplified version is on the right.

In the very first step of the derivation above, we have $\mathsf{s} \underset{r_V 1}{\Longrightarrow} \mathsf{a}$ by applying $r_M 1$, i.e. the metanotion ALPHA was replaced by the very first meta-alternative in the RHS of this rule. This replacement constitutes one of the core aspects of a WG and it will be adequately formalized below.

[13] I apply this simplification in the parse trees below.

3 Six Interesting Properties

3.1 Two-Level and Two-Grammar Grammars

A WG can be generally characterized as a finite specification of a possibly infinite set of context-free production rules (e.g., Baker, 1972).[14] The levels of the finite specification and the possibly infinite set of production rules are thus both distinguishable and interactive, but this is not what is meant when we say that a WG is a two-level grammar: Firstly, a WG is constituted by two CFGs that, when combined as $\frac{\text{CFG}}{\text{CFG}} = \frac{\text{Level 2}}{\text{Level 1}}$, result in a CFG, a CSG, or a UG; these two CFGs correspond then to two levels of the WGs in the sense that in one of the levels (the first) the non-terminal elements of the other level (the second) are defined. These two levels, denoted by $\mathcal{G}^1$ and $\mathcal{G}^2$, correspond respectively to the two kinds of rules of a WG discussed above, to wit, the metarules in the set R_M and the hyper-rules in the set R_V, and their associated sets. They can thus be seen as constituting two subgrammars, to wit, the *metagrammar* and the *hypergrammar*, respectively, so that in a WG we have the level distinction $\frac{\text{Level 2}}{\text{Level 1}} = \frac{\mathcal{G}_2}{\mathcal{G}_1} = \frac{\text{Hyper-level}}{\text{Meta-level}}$. As seen in Example 10, this combination generates what I called above the hyper-language of a WG, denoted by $L_H(\mathcal{G})$. This coinage was purely *ad hoc* and will soon be replaced by another term. Together, and to borrow a well known distinction in linguistics, these two kinds of rules constitute the *deep* level or grammar upon which the *surface* level or grammar is based.[15] I shall denote these two levels or grammars by $\mathcal{G}^D$ and $\mathcal{G}^S$, respectively. In order to avoid confusion I shall reserve the term "level" for the above distinction and speak of grammars when meaning this latter distinction. This latter *two-grammar distinction* is accounted for by the fact that there is a single parse tree in $\mathcal{G}^D$ for the (possibly infinitely many) different words of a WG generated by the production rules in $\mathcal{G}^S$; the parse tree in $\mathcal{G}^D$ is thus seen as the deep structure that accounts for the (possibly infinitely many) terminal strings of a WG taken as surface structures.

We have the two-level/grammar structure of a WG decomposable as shown in Table 3.[16] I next elaborate on the contents of this table. Let us retake Definition 8; this corresponds in fact to grammar $\mathcal{G}^D$, so that we have $\mathcal{G}^D = (M, V, N, T, R_M, R_V, S)$. The two levels $\mathcal{G}^1$ and $\mathcal{G}^2$ can be segregated as follows:

Definition 11. Let $\mathcal{G}^D$ be the deep grammar of a WG $\mathcal{G}$. Then:

1. The metagrammar of $\mathcal{G}$ is the 3-tuple $\mathcal{G}^1 = (M, V, R_M)$.

[14] Recall that in a CFG for a rule $\alpha \to \beta$ we have $\alpha \in V$, $|V(\alpha)| = 1$, and $\beta \in (V \cup T)^*$.

[15] The borrowing is more directly from Chomsky (1964), in which deep and surface *structures* are distinguished as two levels in which a string can be analyzed in a generative grammar, and more indirectly from Hockett (1958), which distinguishes deep and surface *grammars*. The former borrowing was already found in de Chastellier & Colmerauer (1969).

[16] There is no uniformity in the literature as to what the two levels are, but it is obviously certain that there are two levels. For instance, Cleaveland & Uzgalis (1977) consider a lower or first level constituted by the hyperproduction and the metaproduction rules together and an upper or second level constituted by the production rules; roughly the distinction here proposed between deep grammar and surface grammar, respectively. Koster (1974) also considers these two levels, but appears to invert the hierarchy, seeing the production rules as the first level of a WG. Greibach (1974) reserves the two-level distinction to the metarules (level 1) and the hyper-rules (level 2), a practice adopted by other authors. Considering the distinction of two levels and two grammars as proposed above largely captures the usual distinctions.

Table 3: Grammars and levels of a WG $\mathcal{G}$ and corresponding rules.

Grammar	Subgrammar	Level	Rules
Deep ($\mathcal{G}^D$)	Metagrammar	$\mathcal{G}^1$	Metarules
	Hypergrammar	$\mathcal{G}^2$	Hyper-rules
Surface ($\mathcal{G}^S$)	Strict Grammars $\mathcal{G}_{x_i'}$		Production rules

2. The hypergrammar of $\mathcal{G}$ is the 4-tuple $\mathcal{G}^2 = (N, T, R_V, S) = ((M \cup V), T, R_V, S)$.

For practical reasons, one often considers $\mathcal{G}^1 = (M, V, R_M, W)$, where $W \in M$. This definition shows that the sets common to both $\mathcal{G}^1$ and $\mathcal{G}^2$ are M and V. In Example 10, we have "ALPHA" as both a metanotion in R_M and a metaterminal in R_V (since we have 'letter ALPHA symbol', i.e. $(\{\text{ALPHA}\} \subseteq M) \cup \{\text{symbol}\}$). However, the terminal symbols of the language $L\left(\mathcal{G}^D_{Alpha1}\right) = \{t^n | n \geq 1, t = a, b, ..., \text{ or } z\}$ are a, b, ..., z, whose corresponding protonotions $x \in V$ are precisely the meta-alternatives a, b, ..., z given for ALPHA, so that there is an interaction between R_M and R_V implicating the sets M and V.[17] This interaction is in fact a replacement defined as follows.

Definition 12. Let $\mathcal{G}$ be a WG. Then, the set of all the homomorphisms $h \in Hom$ ("choice functions") assigning to each metavariable $W \in M$ an element of the language determined by it and the metarules

$$Hom(M, V, R_M) = Hom\left(\mathcal{G}^1\right) = \widehat{\mathcal{G}} =$$

$$= \left\{h : M \longrightarrow \bigcup \left\{L\left(\mathcal{G}^1\right) | W \in M\right\} | \forall W \in M, h(W) \in L\left(\mathcal{G}^1\right)\right\}$$

is called the *universal assignment to metavariables* and the homomorphism $h(W)$ is referred to as *uniform replacement*. The rule that applies this uniform replacement is called *uniform replacement rule* (URR).

Example 13. Consider the following set of (abstract) metarules of a given WG $\mathcal{G}$:

$$R_M = \left\{ \begin{array}{c} R \to S\,;\, \epsilon \\ S \to r\,;\, rS \\ X \to a;\, b\,;\, c \end{array} \right\}$$

Then, we have $Hom(\{R, S, X\}, \{a, b, c, r\}, R_M) = \widehat{\mathcal{G}}$ if and only if we have $h(R) \in \{r^n | n \geq 0\}$, $h(S) \in \{r^n | n > 0\}$, and $h(X) \in \{a, b, c\}$.

$Hom\left(\mathcal{G}^1\right) = \widehat{\mathcal{G}}$ can thus be seen as a metagrammar with respect to $\mathcal{G}^2$ in the sense that it specifies the allowed uniform replacements $h(W)$ for every $W \in M$ in the hyper-rules of $\mathcal{G}$. In practice, URR is applied only when there are two or more occurrences of the same metanotion in a hyper-rule, in which case each occurrence of that metanotion is replaced with the same (terminal) meta-alternative; for a metanotion that occurs only once in a hyper-rule, it suffices to replace it with a (terminal) meta-alternative. I shall denote the application of URR to a metanotion W in a

[17] The above *ad-hoc* language $L_H(\mathcal{G})$ is replaced by $L\left(\mathcal{G}^D\right)$.

Table 4: Rules, notions, and grammar levels in a WG.

Rules	**Non-Terminals**	**Terminals**	**Grammar and Level**
$r_M \in R_M$	Metanotions	Protonotions	$\mathcal{G}^{D.1}$
$r_V \in R_V$	Hypernotions	Hypernotions	$\mathcal{G}^{D.2}$
$r_S \in R_S$	Protonotions (Strict Notions)	Protonotions ending in 'symbol'	$\mathcal{G}^S$

hyper-rule $r_V i$ where W is replaced by the protonotion x as defined in the metarule $r_M j$ by $URR^{r_V i}_{r_M j}(W, x)$ (abbreviated as $URR^i_j(W, x)$); if transitivity of replacement is the case, I write $URR^{r_V i}_{r_M j-k}(W, x)$, where $r_M j$ and $r_M k$, $j < l \leq k$, are the metarules in the transitive replacement. Once one has applied URR (or made a simple replacement for a single-occurrence metanotion) to all the metanotions of a hyper-rule, one or more production rules have been derived. The following definition formalizes this derivation.

Definition 14. Let $\mathcal{G}$ be a WG and $r_V j = \left(\langle X_{0_j} \rangle \to \overset{\circ}{X}_{1_j}; \overset{\circ}{X}_{2_j}; ...; \overset{\circ}{X}_{m_j}\right)$ a hyper-rule with the $n \geq 1$ metanotions $W_1, W_2, ..., W_n$. The set R_S of *strict production rules* of $r_V j$ is defined as

$$R_S(r_V j) =$$

$$\bigcup \left\{ h(\langle X_{0_j} \rangle) \to h\left(\overset{\circ}{X}_{1_j}\right); h\left(\overset{\circ}{X}_{2_j}\right); ...; h\left(\overset{\circ}{X}_{m_j}\right) | h\left(X_{i_j}\right) = URR^{r_V j}_{R_M}(W_l, x) \right\}$$

where $h(v) = v$, $v \in V \cup \{,\}$, $h(\langle X_0 \rangle) = x' \neq \epsilon$ is a *strict notion*, and $h(W_l) \in L((M, V, R_M, W_l))$. Equally, one says that $x' = h\left(X_{i_j}\right) \in \text{RHS}(r_S k)$, $i = 1, ..., m$, $k = 1, ..., s$, $X_i \notin T$, is a strict notion if there is a production rule $r_S j$ such that $h\left(X_{i_j}\right) \in \text{LHS}(r_S j)$, $j = 1, ..., s$, and the set of (strict) notions is defined as:

$$N_S = \bigcup \left\{ h(X) = x' \in \text{LHS}(r_S j)^s_{j=1} | X \in N, h(X) \in P \right\}$$

For some X_i, $i \neq 0$, if we have the paranotion '$h(X_i)$ symbol' $= x$, then x is called a *terminal (strict) notion.*

Intuitively, a strict production rule r_S is derived from a hyper-rule r_V if one can turn a copy of the latter into a copy of the former by means of a consistent substitution in such a way that the hypernotions in r_V are turned into protonotions in r_S. These replaced hypernotions, called strict notions, contain protonotions that are replacing metanotions in r_V for which there is a terminal metarule r_M, and the derived production rule *describes* precisely this replacement by means of a terminal production. Thus, and more formally, we have $R_S = \bigcup R_S(r_V i)^k_{i=1}$, i.e. the set of strict production rules of a WG $\mathcal{G}$ is the union of all the derivations via $\widehat{\mathcal{G}}$ from the set $R_V = \{r_V i\}^k_{i=1}$. Table 4 gives a summary of all the above in terms of notions.

A definition of a surface grammar $\mathcal{G}^S$ can now be given:

Definition 15. The *surface grammar* of a WG $\mathcal{G}$ is the 4-tuple $\mathcal{G}^S = (N_S, T, s, R_S)$ where T is the set of terminal protonotions of $\mathcal{G}$, $s \in N_S$ is the *start strict notion*

derived from $\langle s \rangle \in R_V$, and the remaining elements are defined as

$$N_S = \bigcup_i \left\{ x_i^{'} \right\}$$

and

$$R_S = \bigcup R_{S_{x_i'}}$$

where $R_{S_{x_i^{'}}}$ is the subset of strict production rules for $x_i^{'}$. We have it that the 4-tuple $\mathcal{G}_{x_i'} = \left(N_{S_{x_i'}}, T', s, R_{S_{x_i'}} \right)$ is the *strict (sub-)grammar* for the i-th strict notion $x_i^{'}$.[18]

We thus can say, by a slight abuse of terminology, that the surface grammar $\mathcal{G}^S$ of a WG $\mathcal{G}$ is the union of all its strict (sub-)grammars.

Example 16. In $\mathcal{G}^2_{Alpha1}$, which can be abbreviated as $Alpha1^2$, the meta-alternatives of the metanotion ALPHA are a, b, ..., z; applying URR to the first rule in R_V for each of the meta-alternatives, i.e. URR_1^1 (ALPHA,x) for $x = \mathsf{a}, \mathsf{b}, ..., \mathsf{z}$, we have the following twenty-six production rules:

$$R_{S_{Alpha1}}(r_V 1) = \left\{ \begin{array}{c} \mathsf{s} : \mathsf{a}. \\ \mathsf{s} : \mathsf{b}. \\ \vdots \\ \mathsf{s} : \mathsf{z}. \end{array} \right\}$$

Or, equivalently, the single production rule with twenty-six alternatives:

$$R_{S_{Alpha1}}(r_V 1) = \{ \mathsf{s} : \mathsf{a}; \mathsf{b}; ...; \mathsf{z}. \}$$

It is easy to see that this single production rule corresponds exactly to the single rule in $R_{M_{Alpha1}}$, i.e. in $Alpha1^1$. Applying now URR on the metanotion ALPHA on the second rule in R_V for the protonotion 'a' we obtain:

$$URR_1^2(\mathsf{ALPHA}, \mathsf{a}) = \mathsf{a} : \mathsf{letter\ a\ symbol}; \mathsf{a}, \mathsf{letter\ a\ symbol}.$$

By means of the application of URR for all the a, b, ..., z, twenty-six production rules with two alternatives of the form

$$x' : \underbrace{\mathsf{letter}\ x\ \mathsf{symbol}}_{\mathrm{A}_1}; \underbrace{x', \mathsf{letter}\ x\ \mathsf{symbol}}_{\mathrm{A}_2}.$$

can be derived from the second rule in R_V:

$$R_{S_{Alpha1}}(r_V 2) = \left\{ \begin{array}{c} \mathsf{a} : \mathsf{letter\ a\ symbol}; \mathsf{a}, \mathsf{letter\ a\ symbol}. \\ \mathsf{b} : \mathsf{letter\ b\ symbol}; \mathsf{b}, \mathsf{letter\ b\ symbol}. \\ \vdots \\ \mathsf{z} : \mathsf{letter\ z\ symbol}; \mathsf{z}, \mathsf{letter\ z\ symbol}. \end{array} \right\}$$

[18] Henceforth, I abbreviate "strict production rule" as "production rule"; the context disambiguates whether "production rule" refers to a rule in a Chomsky grammar or a strict production rule in a WG, denoted respectively by r_P and r_S. Additionally, I abbreviate "$x_i^{'}$" as "x'".

We thus have $|R_{S_{Alpha1}}| = 52$, i.e. the grammar $Alpha1^S$ has fifty-two production rules. If $R_{S_{Alpha1}}(r_V 1)$ is abbreviated as above, then we end up with twenty-seven production rules in $Alpha1^S$. But $Alpha1^S$ corresponds in fact to the single abstract strict grammar $\mathcal{G}_{S_{x'}} = \left(\{x'\}, \{x\}, s, R_{S_{x'}}\right)$ where the set of strict production rules is

$$R_{S_{x'}} = \left\{ \begin{array}{l} \mathsf{s} : x'. \\ x' : \text{ letter } x \text{ symbol}; \\ \qquad x', \text{ letter } x \text{ symbol}. \end{array} \right\}$$

from which twenty-six disjoint patterns of words (for a, b, ..., z) can be derived.

3.2 Languages, Strict Languages, and Strict Metalanguages

Derivations in a WG $\mathcal{G}$ and the language generated by $\mathcal{G}$ can now – and replacing Def. 9 – be defined as follows:

Definition 17. Let $\mathcal{G}$ be a WG and let $\overset{*}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}}$ be the reflexive and transitive closure of the relation $\underset{\widehat{\mathcal{G}}}{\Longrightarrow} \subseteq P^+$. We have the derivation step $\alpha \underset{\widehat{\mathcal{G}}}{\Longrightarrow} \beta$ if there is a production rule $r_S = x' \in (N_S)^+ \to x \in P^*$ and strings $\gamma, \delta \in P^*$ such that $\underbrace{\gamma, x', \delta}_{\alpha} \to \underbrace{\gamma, x, \delta}_{\beta}$, where the commas denote concatenation.[19] Then, the language generated by $\mathcal{G}$ is defined as

$$L(\mathcal{G}) = \left\{ w \in T^* | \mathsf{s} \overset{*}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}} w \right\} = L\left(\mathcal{G}^S\right)$$

where $\mathsf{s} \in N_S$ is the start strict notion.

Because R_S is a description of the uniform replacements admissible in the metagrammar $\widehat{\mathcal{G}}$ that are implemented in the production rules it can be considered as generating a *metalanguage* $L\left(\widehat{\mathcal{G}}\right)$ of $L(\mathcal{G})$ such that

$$L\left(\widehat{\mathcal{G}}\right) = L\left(L\left(\mathcal{G}^D\right)\right) = L\left(\mathcal{G}^S\right).$$

Example 18. Let us retake the grammar $\mathcal{G}_{Alpha1}$. The $URR_1^2(\mathsf{ALPHA}, \mathsf{a})$ applied to the entire set R_V gives a strict subgrammar, namely the set of strict productions R_{S_a} that generates the surface language $L_\mathsf{a}\left(Alpha1^S\right) = \{a^n | n > 0\}$:

$$URR(\mathsf{ALPHA}, \mathsf{a}) = R_{S_\mathsf{a}} = \left\{ \begin{array}{l} \mathsf{s} \ : \ \mathsf{a}. \\ \mathsf{a} \ : \ \text{letter a symbol}; \\ \qquad \mathsf{a}, \text{ letter a symbol}. \end{array} \right\}$$

[19] Just as in the case of Def. 9, leftmost and rightmost derivations, denoted by $\alpha \overset{*}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}}_l \beta$ and $\alpha \overset{*}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}}_r \beta$, respectively, can be specified whenever all the derivation steps are of the form $\alpha \underset{\widehat{\mathcal{G}}}{\Longrightarrow}_l \beta$ or $\alpha \underset{\widehat{\mathcal{G}}}{\Longrightarrow}_r \beta$.

The terminal string a^n is generated by the subgrammar $R_{S_{\mathsf{a}}}$ of $\mathcal{G}_{Alpha1}$ in the following rightmost derivation $\mathsf{s} \overset{n+1}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}}_r a^n$:

$$\mathsf{s} \Longrightarrow \mathsf{a} \underset{\mathrm{A}_2}{\Longrightarrow}_r \mathsf{a}, \overbrace{a}^{\text{letter a symbol}} \underset{\mathrm{A}_2}{\Longrightarrow}_r$$

$$\mathsf{a}, \overbrace{a}^{\text{letter a symbol}} a \underset{\mathrm{A}_2}{\Longrightarrow}_r \ldots \underset{\mathrm{A}_2}{\Longrightarrow}_r \mathsf{a}, \overbrace{aaa\ldots a}^{n-1} \underset{\mathrm{A}_1}{\Longrightarrow}_r a^n$$

where in the last step of the derivation the first alternative (A_1) of the production rule for 'a' is applied. We could easily obtain a leftmost derivation by formulating the second alternative A_2 as 'letter a symbol, a'. The word a^n is infinite, denotable by a^ω, if the first alternative A_1 of this production rule is never applied.

It is easy to see that $R_{S_{\mathsf{a}}}$ corresponds to the CFG $G_{Alpha1(a)} = (\{A\}, \{a\}, S, R)$ where

$$R = \left\{ \begin{array}{c} S \to A \,|\, SA \\ A \to a \,|\, \epsilon \end{array} \right\}$$

but whereas 26 separate CFGs are required to produce strings of the form t^n, it suffices to have a single strict grammar $\mathcal{G}_{x'}$ in the case of a WG $\mathcal{G}$. Thus, the surface language $L\left(\mathcal{G}^S\right)$ can be more than a single (possibly infinite) set of terminal strings; it can actually be a (possibly infinite) set of languages, namely the union of all the sets of words that can be generated for the different strict notions in R_S. This, in turn, requires a union of subgrammars, as seen above.

Definition 19. Let $x' \in N_S$ be a strict notion in a production rule $r_{S_{x'}}$ and such that $\mathcal{G}_{x'}$ is the strict grammar for x'.[20] A string of terminal symbols derived from x', denoted by $w_{x'}$, such that we have $x' \overset{*}{\underset{\mathcal{G}^S}{\Longrightarrow}} w_{x'}$, is called a *strict word*. The set of all the strict words that can be generated by $r_{S_{x'}}$ is called the *strict language of* x', and it is defined as:

$$L_{x'}\left(\mathcal{G}^S\right) = \left\{ w_{x'} \in T^* | x' \overset{*}{\Longrightarrow} w_{x'} \right\}$$

The *strict metalanguage of a WG* $\mathcal{G}$ is then defined as

$$L_{N_S}(\mathcal{G}) = \bigcup L_{x'} \text{ such that } \bigcap L_{x'} = \emptyset.$$

$L_{x'}\left(\mathcal{G}^S\right)$ and $L_{N_S}(\mathcal{G})$ are abbreviated as $L_{x'}$ and $L_{\widehat{\mathcal{G}}}$, respectively. The latter abbreviation denotes the fact that $L\left(\mathcal{G}^S\right)$ and $L_{N_S}(\mathcal{G})$ coincide for a WG $\mathcal{G}$. I shall abbreviate "strict word" as "word" whenever there is no risk of confusion.

Example 20. The grammar *Alpha1* generates twenty-six strict languages, one for each of the letters of the Roman alphabet. In effect, we have:

$$L\left(\mathcal{G}^S_{Alpha1}\right) = \bigcup_{x'=a,b,\ldots}^{z} L_{x'}$$

[20] Note that the strict notion x' can coincide with the strict start notion s.

We thus have the strict languages:

$$L_{\mathsf{a}} = \{a^n | n > 0\}$$

$$L_{\mathsf{b}} = \{b^n | n > 0\}$$

$$\vdots$$

$$L_{\mathsf{z}} = \{z^n | n > 0\}$$

The strict metalanguage of *Alpha1* is thus:

$$L_{\widehat{Alpha1}} = L_{\mathsf{a}} \cup L_{\mathsf{b}} \cup ... \cup L_{\mathsf{z}} = L\left(Alpha1^S\right) = L_{N_S}\left(Alpha1^S\right)$$

$$= \left\{ \begin{array}{c} L_{\mathsf{a}} = \{a^n | n > 0\} \\ L_{\mathsf{b}} = \{b^n | n > 0\} \\ \vdots \\ L_{\mathsf{z}} = \{z^n | n > 0\} \end{array} \right\}$$

This completes the elaboration on all the basic components of a WG, all of which are given with examples on Table 5.

3.3 Infinite Productivity

WGs exhibit the interesting property that $\mathcal{G}^D$, which contains a finite number of rules, can derive an infinite number of production rules in $\mathcal{G}^S$. This contrasts with, and thus segregates them from, the CFGs. These latter grammars can generate (languages of) infinite words $w \in \Sigma^\omega$ (or $w \in (\Sigma^* \cup \Sigma^\omega)$),[21] known in the literature as ω-words and where the superscript ω denotes the operation over some set $S \subseteq \Sigma^*$ defined as

$$S^\omega := \{t_1 t_2 ... t_i ... | i \in \mathbb{N}, t_i \in S - \{\epsilon\}\}$$

but the number of rules in their production sets is per force finite. A language of ω-words is called an ω-language, but it is not a necessary condition that a language be an ω-language to be an infinite language. I give the example of a complete WG for the CFG generating the CFL $L = \{a^n b^n | n > 0\}$. This CFL can be generated by a simple CFG: The grammar $G_{Alpha2} = (V, T, S, R)$ with $V = \{S\}$, $T = \{a, b\}$, and $R = \{r_P 1, r_P 2\}$, where $r_P 1 = S \to aSb$ and $r_P 2 = S \to ab$. By applying $n - 1$ times rule $r_P 1$ and then $r_P 2$ we obtain the derivation

$$\underbrace{S \Longrightarrow aSb \Longrightarrow a^2Sb^2 \Longrightarrow ... \Longrightarrow a^{n-1}Sb^{n-1}}_{(n-1)r_P 1} \underbrace{\Longrightarrow a^n b^n}_{r_P 2}.$$

While n can be arbitrarily large, the terminal string $a^n b^n$ is per force finite, namely by the application of $r_P 2$ in the final step of the derivation. However, the language $L\left(G_{Alpha2}\right)$ need not be finite, as long as no upper bound for n is specified, in which case we may have $L\left(G_{Alpha2}\right) = \left\{ab, a^2b^2, ..., a^i b^i, ...\right\}$. This infinite *generativity*

[21] Clearly, $w \in \Sigma^\omega$ is a special case of $w \in (\Sigma^* \cup \Sigma^\omega)$.

Table 5: Components of a WG with examples from $\mathcal{G}_{Alpha1}$.

Term	**Symbol**	**Set**	**Description**	**Example**
Protonotion	x	P^*	Sequence $\lvert\overline{\sigma}\rvert \geq 0$ of small syntactic marks	a; letter a symbol
Terminal (Symbol)	t	$T \subset P$	Protonotion $\lvert\overline{\sigma}\rvert \geq 1$ ending in 'symbol'	letter a symbol
(Proto-)Variable	v	$V \subseteq P$	Non-terminal protonotion $\lvert\overline{\sigma}\rvert \geq 1$	s; a
Strict Notion	x'	$N_S \subset P'$	Protonotion $\lvert\overline{\sigma}\rvert \geq 1$ in the LHS of a production rule	s; a
Metanotion	W	M	Sequence $\lvert\overline{\sigma}\rvert \geq 1$ of large syntactic marks	ALPHA
Hypernotion	$\langle\mathcal{X}\rangle$	$N \subseteq (M \cup P)^*$	Sequence $\lvert\overline{\sigma}\rvert \geq 0$ of large and/or small syntactic marks	ALPHA; letter ALPHA symbol
Word	w	$L\left(\mathcal{G}^D\right)$	A (pattern of a) finite sequence of typographical characters $t_1t_2...t_k$	$t^n \lvert n > 0, t \in \{a, b, ..., z\}$; aaa
Strict Word	$w_{x'}$	$L\left(\mathcal{G}^S\right)$	A word derived from a strict notion x'	aaa
Metarule	r_M	R_M	Definition of a metanotion	ALPHA :: a; b; ...; z.
Hyper-rule	r_V	R_V	A production pattern	s : ALPHA.
Strict Production rule	r_S	R_S	Word-generation rule	a : letter a symbol.
Deep Language	$L_{\mathcal{G}}$	$L\left(\mathcal{G}^D\right)$	Set of word patterns	$L_{Alpha1} = \{t^n \lvert n > 0, n \in \{a, b, ..., z\}\}$
Strict Language	L_{N_S}	$L_{x'}\left(\mathcal{G}^S\right)$	Set of words for x'	$L_{\mathsf{a}}\left(Alpha1^S\right) = \{a^n \lvert n > 0\}$
Surface Language, Strict Metalanguage	$L_{\widehat{\mathcal{G}}}$	$L\left(\mathcal{G}^S\right)$	The union $\bigcup_{i=1}^{k} L_{x'_i}\left(\mathcal{G}^S\right)$ of the strict languages	$L_{\widehat{Alpha1}} = \left\{ \begin{array}{c} L_{\mathsf{a}} = \{a^n \lvert n > 0\} \\ L_{\mathsf{b}} = \{b^n \lvert n > 0\} \\ \vdots \\ L_{\mathsf{z}} = \{z^n \lvert n > 0\} \end{array} \right\}$

needs thus to be distinguished from the infinite *productivity* of the WGs, whose production set can actually have an infinite number of rules, namely one rule for each of the infinitely many words that can be generated by the grammar. The ω-languages are theoretically a messy matter (e.g., Staiger, 1997), so a simpler means to allow for the generation of an infinite number of terminal strings is welcome in the field of formal languages for practical ends. A WG $\mathcal{G}$ allows precisely for this by the derivation from $\mathcal{G}^D$ of an infinite number of production rules in $\mathcal{G}^S$ such that we have

$$R_S = \bigcup_{i=1}^{\omega} r_{S_w} i$$

where w denotes a terminal string such that $i < j$ if $|w|_{r_S i} < |w|_{r_S j}$, $|w|_{r_S i}$ denotes the length (i.e. number of typographical characters) of the word w generated by the production rule $r_S i$.

Example 21. The following rules constitute the WG $\mathcal{G}_{Alpha2}$ (abbreviated as *Alpha2*) that generates the CFL $L(Alpha2) = \{a^n b^n | n > 0\}$:

$(r_M 1)$	N	::	n; Nn.
$(r_V 1)$	s	:	Na, Nb.
$(r_V 2)$	nNa	:	letter a symbol, Na.
$(r_V 3)$	nNb	:	letter b symbol, Nb.
$(r_V 4)$	na	:	letter a symbol.
$(r_V 5)$	nb	:	letter b symbol.

Rule $r_M 1$ allows for the generation of a string of n 'n' as long as desired by the application of the second meta-alternative $n-1$ times. Suppose we want $n = 5$; applying four times the second meta-alternative of this rule will generate the sequence 'Nnnnn'. We can see the first meta-alternative of this rule as a $URR_1^1(\mathsf{N}, \mathsf{n})$, so that by applying this URR to 'Nnnnn' we obtain the desired sequence 'nnnnn'. By proceeding in this way, we obtain the following infinite set of production rules for the strict start notion from $r_V 1$:

$$R_{S_{Alpha2}}(r_V 1) = \left\{ \begin{array}{c} \mathsf{s} : \mathsf{na}, \mathsf{nb}. \\ \mathsf{s} : \mathsf{nna}, \mathsf{nnb}. \\ \mathsf{s} : \mathsf{nnna}, \mathsf{nnnb}. \\ \mathsf{s} : \mathsf{nnnna}, \mathsf{nnnnb}. \\ \vdots \end{array} \right\}$$

Once the required number of n is obtained, say, five (s : nnnnna, nnnnnb.), one has to start "eating up" the n to the left of a and b and replace it by these letters. By applying $URR_1^{2,3}(\mathsf{N}, \mathsf{n})$ repeatedly on hyper-rules $r_V 2$ and $r_V 3$ we obtain the production rules

$$\underbrace{\mathsf{nnnnn}}_{\ast}\mathsf{a} : \mathsf{letter\ a\ symbol}, \underbrace{\mathsf{nnnn}}_{\ast}\mathsf{a}.$$

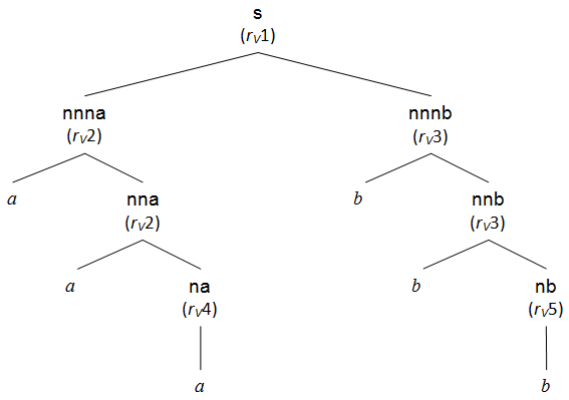

Figure 2: Parse tree for the word *aaabbb* generated by the WG *Alpha2*.

and

$$\underbrace{\mathsf{nnnnn}}_{\divideontimes}\mathsf{b} \ : \ \text{letter b symbol}, \underbrace{\mathsf{nnnn}}_{\divideontimes}\mathsf{b}.$$

where $\divideontimes$ stands for $URR\left(\mathsf{N}, \mathsf{n}^{n-1}\right)$. By decreasingly iterating on n^{n-1} until $\mathsf{n}^0 = \epsilon$ – and thus "eating up" one n every time a terminal a is generated – we obtain the sequence $\mathsf{na}aaaa,\mathsf{nb}bbbb$.[22] The conclusion of the derivation of the word *aaaaabbbbb* requires the application of the production rules directly derived from the hyper-rules r_V4 and r_V5. We have the following set of production rules for $\mathcal{G}_{Alpha2}$ where $\left[\mathsf{n}^{n-1}\right]^\sharp = \left[\mathsf{n}^{n-1}\right]_{n-1}^{0}$ denotes decreasing iteration:

(r_S1)	s	:	$\mathsf{n}^{n-1}\mathsf{a}$, $\mathsf{n}^{n-1}\mathsf{b}$.
(r_S2)	$\mathsf{n}\left[\mathsf{n}^{n-1}\right]^\sharp\mathsf{a}$	:	letter a symbol, $\left[\mathsf{n}^{n-1}\right]^\sharp\mathsf{a}$.
(r_S3)	$\mathsf{n}\left[\mathsf{n}^{n-1}\right]^\sharp\mathsf{b}$	:	letter b symbol, $\left[\mathsf{n}^{n-1}\right]^\sharp\mathsf{b}$.
(r_S4)	na	:	letter a symbol.
(r_S5)	nb	:	letter b symbol.

Because $n \in \mathbb{N}$, $R_{S_{Alpha2}}$ is in fact an infinite set of production rules. Figure 2 shows the parse tree for the word *aaabbb* generated by this grammar. Note in this parse tree that the nodes other than leaves are labeled with strict notions of the form $\mathsf{n}^n\mathsf{a}$, but the rules that are indicated are the hyper-rules. This allows for the construction of a parse tree from a finite number of rules; otherwise, the construction of the parse tree can be impractical or even intractable. However, as in the case at hand, a production tree – instead of a parse tree – can be constructed from the production rules whenever they are abstracted in a finite way, but this is mostly for illustrative ends; given a WG, a parser always constructs a parse tree from the hyper-rules in R_V.

[22] By a "decreasing iteration" it is meant that for the i-th iteration over n, denoted by n_i, we have $n_i < n_j$ for $j < i$. This corresponds to a decreasing (for) loop in programming jargon.

The above strategy of creating replacements of the form $URR(\mathsf{N}, \mathsf{n}^n)$ allows for the efficient generation of WGs with infinite sets of production rules for many languages with exponentiation.

Example 22. The following rules – note the metanotion EMPTY – generate the language $L(\mathcal{G}) = \left\{a^n b^{2n} | n > 0\right\}$:

$(r_M 1)$	N	::	n^n.
$(r_M 2)$	EMPTY	::	.
$(r_V 1)$	s	:	Na, NNb.
$(r_V 2)$	Na	:	letter a symbol.
$(r_V 3)$	NNb	:	letter b symbol, letter b symbol.
$(r_V 4)$	a	:	EMPTY.
$(r_V 5)$	b	:	EMPTY.

Let $[\mathsf{n}^n]^{\sharp\sharp}$ denote the repetition of a decreasing iteration step over n twice; then, the following production rules can be derived:

$(r_S 1)$	s	:	n^na, n^{2n}b.
$(r_S 2)$	$[\mathsf{n}^n]^{\sharp}$a	:	letter a symbol.
$(r_S 3)$	$\left[\mathsf{n}^{2n}\right]^{\sharp\sharp}$b	:	letter b symbol, letter b symbol.
$(r_S 4)$	a	:	.
$(r_S 5)$	b	:	.

As a matter of fact, with this strategy, which can be even more efficient by means of metarules of the forms $\mathsf{N} :: \mathsf{a}^n$, $\mathsf{N} :: \mathsf{Na}^n$, etc. or with notions that denote different exponents (e.g., $\mathsf{I} :: \mathsf{a}^i$, $\mathsf{J} :: \mathsf{a}^j$), any kind of exponentiation can in principle be implemented by a WG with an infinite set of production rules (e.g., Greibach, 1974; Mateescu & Salomaa, 1997). Examples of such languages are:

$$L(\mathcal{G}) = \left\{a^{n^2} | n > 1\right\}$$

$$L(\mathcal{G}) = \{a^n | n \geq 4,\ n \text{ is not prime}\}$$

$$L(\mathcal{G}) = \{a^n | n \text{ is in the Fibonacci sequence}\}$$

$$L(\mathcal{G}) = \left\{a^n b^m c^k | n, m > 0, k = \gcd(n, m)\right\}$$

3.4 Context Sensitivity

It is easy to see that the WG *Alpha1* is constituted by two CFGs whose combination as $\frac{\text{CFG}}{\text{CFG}}$ results in a CFG – if we consider the set of production rules as being finite for any given n. Interestingly enough, this same combination can result in a WG with context sensitivity. Recall that in a CSG $G = (V, T, S, R)$ each production rule $r_P i \in R$, $i = 1, 2, ..., k$, has the form $\gamma A \delta \to \gamma X \delta$, where $A \in V$, $\gamma, \delta \in (V \cup T)^*$, and $X \in (V \cup T)^+$. The informal explanation for the label "context-sensitive" is that the production $\gamma A \delta \to \gamma X \delta$ allows the rewriting of A by X only in the *context* "$\gamma \ldots \delta$".

In terms of the Chomsky hierarchy we have **CFL** $\subsetneq$ **CSL**: There are CSGs that are not CFGs. The language $L(G) = \{a^n b^n c^n | n > 0\}$ is not a CFL, and thus it cannot be generated by a CFG or type-2 grammar; in effect, this language is a CSL generated by a CSG or type-1 grammar.

Example 23. The CSG $G_{Alpha3} = (\{S, X\}, \{a, b, c\}, S, R)$, where P is the following set of production rules

$$R_{Alpha3} = \left\{ \begin{array}{c} S \to abc \,|\, aSX \\ bXc \to bbcc \\ cX \to Xc \end{array} \right\},$$

generates the CSL $L(G_{Alpha3}) = \{a^n b^n c^n | n > 0\}$. The reader can work out the parse of any word $a^n b^n c^n$ to verify this.

The strict class containment **CFL** $\subset$ **CSL** means that we cannot obtain the CSL $L(G_{Alpha3}) = \{a^n b^n c^n | n > 0\}$ from the CFL $L(G_{Alpha2}) = \{a^n b^n | n > 0\}$ by making changes to the CFG G_{Alpha2}, but actually have to construct a wholly new grammar. A WG allows us precisely to extend a given CFG into a CSG. The following example shows how to extend the basic WG *Alpha2* generating $L(\mathcal{G}_{Alpha2}) = \{a^n b^n | n > 0\}$ so that it generates the CSL $L(\mathcal{G}_{Alpha3}) = \{a^n b^n c^n | n > 0\}$.

Example 24. Consider the WG $\mathcal{G}_{Alpha2}$ of Example 21. In order for it to generate the CSL $L(\mathcal{G}_{Alpha3}) = \{a^n b^n c^n | n > 0\}$ we require first of all a second metarule: ABC :: a; b; c. Given this metarule, a simple addition in $r_V 1$ for the additional terminal c and the inclusion of the metanotion ABC in the remaining hyper-rules give us the following WG $\mathcal{G}_{Alpha3}$:

$(r_M 1)$	N	::	n; Nn.
$(r_M 2)$	ABC	::	a; b; c.
$(r_V 1)$	s	:	Na, Nb, Nc.
$(r_V 2)$	nN ABC	:	letter ABC symbol, N ABC.
$(r_V 3)$	n ABC	:	letter ABC symbol.

By applying URR_2^3(ABC,x) for x= a, b, c we derive the following additional hyper-rules:

$(r_V 4)$ na : letter a symbol.

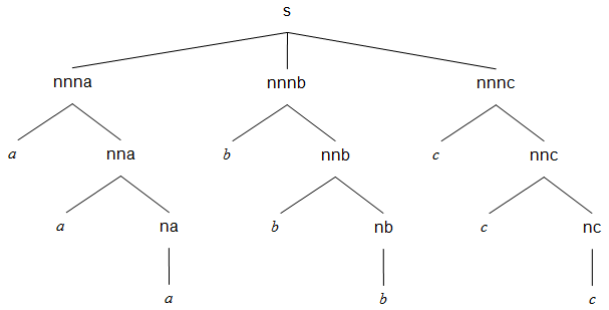

Figure 3: Production tree of the word *aaabbbccc* generated by the WG *Alpha3*.

$$(r_V 5) \qquad \mathsf{nb} : \text{letter b symbol}.$$

$$(r_V 6) \qquad \mathsf{nc} : \text{letter c symbol}.$$

These three hyper-rules are responsible for the termination of a word. Adding them to the grammar would be superfluous, though, because by applying $URR_2^2\,(\mathsf{ABC},x)$ for $x{=}\ \mathsf{a},\mathsf{b},\mathsf{c}$ we obtain the production rules

$$\mathsf{n} \qquad \underbrace{\mathsf{nnnn}}_{\left[\ URR\left(\mathsf{N},\mathsf{n}^{n-1}\right)\ \right]=\divideontimes} \qquad \mathsf{a} : \text{letter a symbol}, \underbrace{\mathsf{nnnn}}_{\divideontimes}\mathsf{a}.$$

$$\underbrace{\mathsf{nnnnn}}_{\divideontimes}\mathsf{b} : \text{letter b symbol}, \underbrace{\mathsf{nnnn}}_{\divideontimes}\mathsf{b}.$$

and

$$\underbrace{\mathsf{nnnnn}}_{\divideontimes}\mathsf{c} : \text{letter c symbol}, \underbrace{\mathsf{nnnn}}_{\divideontimes}\mathsf{c}.$$

By applying $URR_1^1\,(\mathsf{N},\mathsf{n})$ as above, we obtain the following infinite set of initial production rules for *Alpha3*:

$$R_S\,(s) = \left\{ \begin{array}{c} \mathsf{s} : \mathsf{na},\mathsf{nb},\mathsf{nc}. \\ \mathsf{s} : \mathsf{nna},\mathsf{nnb},\mathsf{nnc}. \\ \mathsf{s} : \mathsf{nnna},\mathsf{nnnb},\mathsf{nnnc}. \\ \mathsf{s} : \mathsf{nnnna},\mathsf{nnnnb},\mathsf{nnnnc}. \\ \vdots \end{array} \right\}$$

The reader can now easily derive the whole set of production rules for this WG. Figure 3 shows the parse tree for the word *aaabbbccc* generated by *Alpha3*. It is easy to see that only rules $r_V 1$ through $r_V 3$ are used in it.

Recall now that in a CSG every production is of a form $\alpha \to \beta$ such that $|\beta| \geq |\alpha|$, i.e. no production is length-decreasing, reason why in $\gamma A\delta \to \gamma X\delta$ we have $X \in (V \cup T)^+$. In the terminology of WGs, this property can be defined as follows:

Definition 25. A WG is said to be *lossless* if for each hyper-rule $\langle X_0 \rangle \to \mathring{X}_1; ...; \mathring{X}_m$ we have

$$| \langle X_0 \rangle | \leq |\mathring{X}_i|_{i=1}^{m}$$

and for each $W \in \langle X_0 \rangle$ we have $| \langle X_0 \rangle |_W \leq |\mathring{X}_i|_W$, where $| \langle X_0 \rangle |_W$ ($|\mathring{X}_i|_W$) denotes the length of the hypernotion X_0 (respectively of the hypernotion(s) (in) $\mathring{X}_i = (\gamma \langle \mathcal{X} \rangle \delta)_{i_1} ... (\gamma \langle \mathcal{X} \rangle \delta)_{i_k}$ where $\gamma, \delta \in (T \cup \{, \})^*$) in which W occurs.

The reader can easily verify that *Alpha3* is a lossless WG. Baker (1970) proved that the family of languages generated by lossless WGs is precisely the family of CSLs. The proof requires the construction of a special Turing machine: Any rewriting system such that each derivation step is non-decreasing and there is a multitape Turing machine that can imitate each step $w_1 \Longrightarrow w_2$ by using no more than $|w_2|$ tape cells yields a CSL. (Compare this with the CSLs, which are accepted by linear-bounded Turing machines.[23])

3.5 High Descriptive Power

Another property of interest of the WGs is their high descriptive power, or their capability of describing highly complex languages by employing a rather small – in any case finite – number of rules. Consider the following language:

$$L_{AlphaNoRep} = \{t_1 t_2 ... t_n = w \in T^* | \forall i, j = 1, 2, ..., n : t_i \neq t_j \text{ whenever } i \neq j\}$$

where $T = \{a, b, ..., z\}$ and w is an alphabetic word in which no letter of the Roman alphabet occurs more than once. (*AlphaNoRep* abbreviates *Alphabetic Words with No Repeated Letters.*) The upper bound for the number of CFG rules required to generate this language is $26 + (26)(25) + (26)(25)(24) + ... + 26!$, an impossibly large number. This impracticality is easily overcome by a WG equipped with the two techniques I next elaborate on.

An efficient strategy adopted for WGs to increase (possibly infinitely) their descriptive power while keeping a finite set of rules in the deep grammar is by numbering a given metanotion (e.g., ALPHA1, ALPHA2, ..., ALPHAk); the objective is that of allowing for the application of two or more different URRs on the same metanotion (the *root metanotion*), allowing thus for distinct definitions thereof in a single hyper-rule. This actually corresponds to a variation of the form of metarules that can be formulated as

$$(r_M^k) \qquad \underbrace{\mathsf{METANOTION}_1, ..., \mathsf{METANOTION}_k}_{\text{LHS}} :: \underbrace{X_1; ....; X_n.}_{\text{RHS}}$$

where the subscripts $1 < i \leq k$ index different copies of the same root metanotion and no two distinct copies on the LHS can be replaced simultaneously by the same X_j, $1 \leq j \leq n$, on the RHS. This variation, which is strictly conceived a condensation of k metarules of the form $Wi \to W$ where the Wi are the *specific metanotions* (cf. Meertens, 1969), is more often than not assumed implicitly in the hyper-rules and if

[23] See also Baker (1972) for a proof of the above result recruiting these Turing machines.

k is unbounded so that we have metarules of the form r_M^{ω}, then the descriptive power of a WG is virtually infinite.[24]

Another technique to keep the set of deep-grammar rules of a WG to a minimum while allowing for high descriptive power is the use of *predicates*, i.e. protonotions beginning with 'where', 'unless', 'is', etc. When certain conditions are not met, predicates typically yield so-called *blind alleys*, i.e. protonotions for which no production rule can be derived; otherwise, they yield empty terminal productions. Predicates are thus particularly well-suited to enforce specific restrictions on the syntax of a WG (e.g., the unique occurrence of terminal symbols in a word).

Example 26. For graphical convenience, I further abbreviate *AlphaNoRep* as *NoRep*. Below, I give the rules of the WG $\mathcal{G}_{NoRep}$ that generates the language *NoRep*.[25] For simplicity, I consider a three-letter word generated by this WG, but it should be obvious that words of arbitrary length $|w| \leq 26$ can be parsed with it. The strict language in consideration is $L_{\mathsf{fun}}\left(NoRep^S\right) = \{fun\}$.

(r_M1)	ALPHA	::	a; b; ...; z.
(r_M2)	LETTER	::	letter ALPHA.
(r_M3)	TAG	::	LETTER; TAG LETTER.
(r_M4)	EMPTY	::	.
(r_M5)	NOTION	::	ALPHA; NOTION ALPHA.
(r_M6)	NOTETY	::	EMPTY; NOTETY ALPHA.

(r_V1)	s	:	TAG unique.
(r_V2)	LETTER TAG unique	:	LETTER symbol, TAG unique, where LETTER is not in TAG.
(r_V3)	LETTER unique	:	LETTER symbol.
(r_V4)	where LETTER1 is not in LETTER2 TAG	:	where LETTER1 isnt LETTER2, where LETTER1 is not in TAG.
(r_V5)	where LETTER1 is not in LETTER2	:	where LETTER1 isnt LETTER2.
(r_V6)	where letter ALPHA1 isnt letter ALPHA2	:	where ALPHA1 precedes ALPHA2 in abcde...wxyz; where ALPHA2 precedes ALPHA1 in abcde...wxyz.
(r_V7)	where ALPHA1 precedes ALPHA2 in NOTETY1 ALPHA1 NOTETY2 ALPHA2 NOTETY3	:	EMPTY.

[24] Consider the root metanotion 'MODE' such that the LHS of a metarule r_M^3 is MODE1, MODE2, MODE3. Then, this entails (implicitly) the rules for the specific metanotions MODE1 :: MODE , ..., MODE3 :: MODE.

[25] This example is from Cleaveland & Uzgalis (1977). Because of the large size of this grammar I give the metarules and the hyper-rules in separate tables. For graphical convenience the sequence of the 26 letters of the Roman alphabet is abbreviated as abcde...wxyz in the RHS of rule r_V6.

I next describe in detail the derivation of the production rules to generate the word "*fun*". (The reader is invited to try their hand at constructing the parse tree for the word "*funny*", in order to verify that this word cannot be parsed by $\mathcal{G}_{NoRep}$, i.e. we have $L_{\mathsf{funny}}\left(NoRep^{S}\right) = \emptyset$.)

- By transitivity of replacement over the metarules r_M1 through r_M3 we obtain $URR^{1}_{1-3}\left(\mathsf{TAG}, \mathsf{letter}\, x\right)$, where x is any of the 26 letters of the Roman alphabet. This URR applied on r_V1 gives the specific start production rule for this word:

$$(r_S1) \qquad \mathsf{s} : \mathsf{letter\ f\ letter\ u\ letter\ n\ unique}.$$

- From r_V2 and from r_S1 we now derive the production rule:

$$(r_S2) \qquad \mathsf{letter\ f\ letter\ u\ letter\ n\ unique} :$$

$$\underbrace{\mathsf{letter\ f\ symbol}}_{\mathsf{A}}, \underbrace{\mathsf{letter\ u\ letter\ n\ unique}}_{\mathsf{B}},$$

$$\underbrace{\mathsf{where\ letter\ f\ is\ not\ in\ letter\ u\ letter\ n}}_{\mathsf{C}}.$$

- Applying the strict alternative A in r_S2 generates directly the terminal symbol "*f*". Strict alternative B gives us recursively

$$(r_S2') \qquad \mathsf{letter\ u\ letter\ n\ unique} :$$

$$\underbrace{\mathsf{letter\ u\ symbol}}_{\mathsf{A'}}, \underbrace{\mathsf{letter\ n\ unique}}_{\mathsf{B'}},$$

$$\underbrace{\mathsf{where\ letter\ u\ is\ not\ in\ letter\ n}}_{\mathsf{C'}}.$$

  and C (a predicate) gives from r_V4:

$$(r_S4) \qquad \mathsf{where\ letter\ f\ is\ not\ in\ letter\ u\ letter\ n} :$$

$$\underbrace{\mathsf{where\ letter\ f\ isnt\ letter\ u}}_{\mathsf{D}}, \underbrace{\mathsf{where\ letter\ f\ is\ not\ in\ letter\ n}}_{\mathsf{E}}.$$

- A′ of r_S2' gives us the terminal symbol "*u*" and B′ gives recursively

$$(r_S3) \qquad \mathsf{letter\ n\ unique} : \mathsf{letter\ n\ symbol}.$$

  which gives directly the terminal symbol "*n*". By now the leftmost three leaves of the parse tree contain precisely the word "*fun*"; the remaining nodes to the right of this word will test for repeated symbols, and the tree will be complete only if all the corresponding leaves are empty; otherwise, there will be blind alleys and the tree will be incomplete. The rules with predicates allow for this testing.

- C′ in r_S2' produces the following production rule, which corresponds to r_V5:

$$(r_S5) \qquad \mathsf{where\ letter\ u\ is\ not\ in\ letter\ n} :$$

$$\mathsf{where\ letter\ u\ isnt\ letter\ n}.$$

- From the RHS of r_S5, and corresponding to r_V6, we obtain the production rule:

$$(r_S6) \qquad \text{where letter u isnt letter n :}$$

$$\text{where n precedes u in abc...xyz.}$$

- From the RHS of this production rule and from r_V7 we obtain:

$$(r_S7) \qquad \text{where n precedes u in abc...xyz : .}$$

The production process now retakes rightwards from r_S4, but the above detailed explanation suffices to show how to construct the parse tree of the word "*fun*" (see Fig. 4). Note that the leaves are empty upon application of hyper-rule r_V7. (The reader can try to build the parse tree for the illegal word "*funny*" to check the descriptive power of the hyper-rules $r_V4 - r_V7$.)

3.6 Equivalence with Type-0 Grammars

A Chomsky grammar $G = (V, T, S, R)$ of type 0, or a UG, was defined above in Subsection 2.1. Just like CSGs, UGs allow for productions in which a variable may depend on the context; for instance, the production rule $Bc \to bc$ allows the substitution of the variable B by the terminal b only if B is immediately followed by c on the LHS. In particular, given a UG G, the assumption $S \underset{G}{\overset{*}{\Longrightarrow}} xAy \underset{G}{\overset{*}{\Longrightarrow}} v$ no longer implies that $v = xuy$ for some terminal string v; this makes of a UG a semi-Thue system. A language $L \subseteq T^*$ is a recursively enumerable language (REL) if and only if there is a UG G such that $L = L(G)$.

Example 27. The language $L(G_{Alpha3}) = \{a^n b^n c^n | n > 0\}$, where G_{Alpha3} is a grammar of type 0, is a REL. This is how $L(Alpha3)$ can be generated (contrast with Example 23): We begin by creating a variable L denoting "left end of a string." We then require production rules that (i) allow us to obtain strings of the form $L(ABC)^n$, and (ii) allow for an alphabetical (re)arrangement of the variables A, B, and C. These productions are as follows:

$$(i) \qquad R_1 = \{S \to SABC \mid LABC\}$$

$$(ii) \qquad R_2 = \{BA \to AB; CB \to BC; CA \to AC\}$$

Now, of course, we need productions that (iii) allow us to replace variables that are already in alphabetic order by terminals:

$$(iii) \qquad R_3 = \{LA \to a; aA \to aa; aB \to ab; bB \to bb; bC \to bc; cC \to cc\}$$

The set of type-0 production rules for $L(Alpha3)$ is $R = \bigcup_{i=1}^{3} R_i$. (The reader can try to work out the derivation of any word $a^n b^n c^n$.) Because the LHS of production rules in a UG typically has more than one symbol and moreover the sentential forms in a derivation can vary greatly in length, at times increasing and decreasing, we cannot

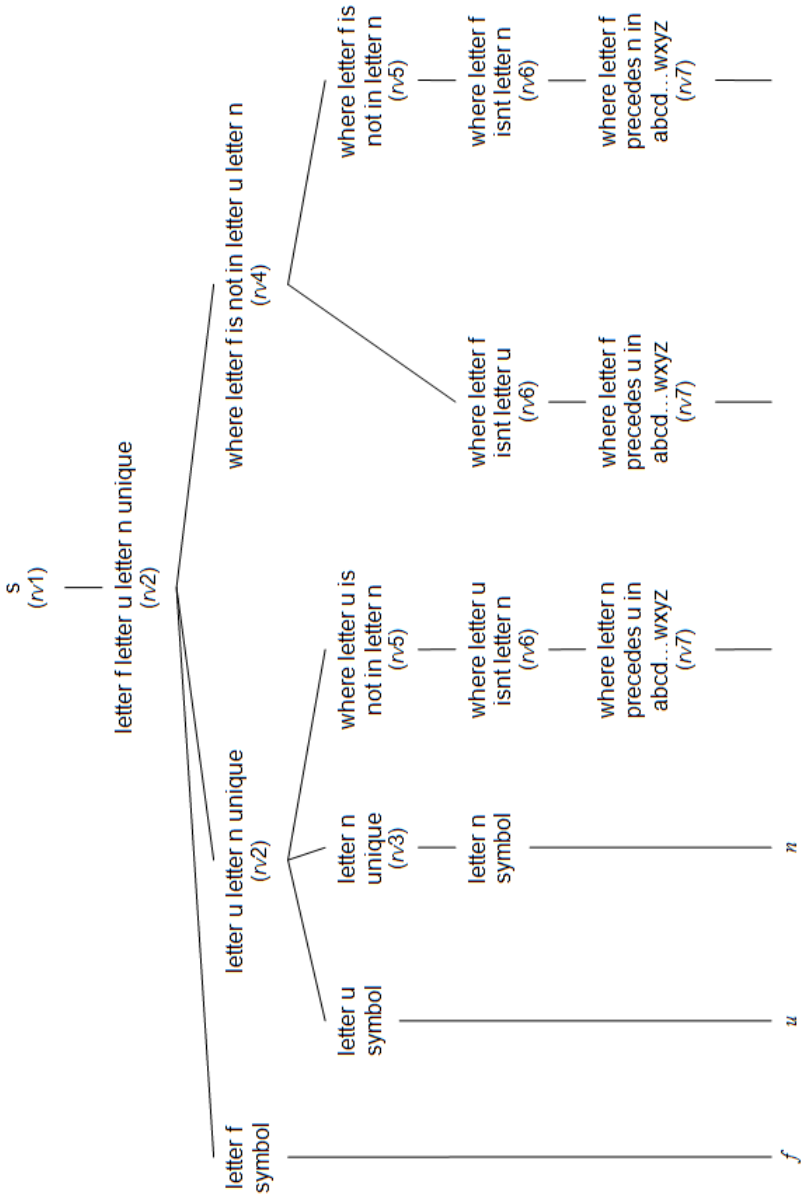

Figure 4: Parse tree of the word "*fun*" generated by the WG $\mathcal{G}_{NoRep}$.

construct parse trees for words of RELs.[26] In effect, the parsing problem for these languages is known to be unsolvable. To put it briefly, the generative capability of UGs is too strong – literally: unrestricted – for most current computational applications of formal grammars.

WGs allow us not only to explore this generative power in a more "restricted" way, but also to extend it. WGs have been shown to be equivalent to UGs in the sense that **REL** $\subseteq$ **WL** (Sintzoff, 1967) and **WL** $\supseteq$ **REL** (Janssen, 1975), so that adding a third level does not increase their generative power;[27] actually, a single metanotion suffices to generate every REL, as shown by A. van Wijngaarden himself (van Wijngaarden, 1974).

But in fact WGs are even more expressive than UGs, in the sense that we have the strict inclusion **REL** $\subset$ **WL** (cf. Grune, 1993); in particular, a UG cannot generate the language consisting of an unbounded number k of different terminal characters each occurring the same number n of times in succession, i.e. the language *AlphaK*:[28]

$$L\left(\mathcal{G}_{AlphaK}\right) = \{t_1^n ... t_k^n | n \geq 0, k > 0, \forall i, j = 1, ..., k : t_i \neq t_j \text{ if } i \neq j\} = Q$$

Of course, the word $a^3b^3c^3$ belongs to the language *AlphaK*, but so do, say, $z^5f^5q^5t^5$ or $p^{125}q^{125}a^{125}$, or also $a_{\overline{5}}^6 a_{\overline{3}}^6 a_{\overline{1}}^6 a_{\overline{159}}^6 a_{\overline{20}}^6$ if we choose – as we must, because k is an unbounded number – to have an infinite alphabet $\Sigma_\omega = \{a_{\overline{1}}, a_{\overline{2}}, ...\} = \{a_{\overline{i}}\}_{i=1}^{\omega}$, where the subscript notation "$\overline{i}$" is used to distinguish these subscripts from those in $t_1^n ... t_k^n$. It must be remarked that Q is not a well-defined set from the viewpoint of a UG, because the t_i are not specified as being members of a given set; in particular, this set cannot be the Roman alphabet, as this set is clearly bounded. On the contrary, from the viewpoint of a WG the set Q is well defined, because each t_i in a terminal string $w = t_1^n t_2^n ... t_k^n$ is to be replaced by some typographical character in a representation table (cf. Table 1 for an example). This entails also a difference with respect to Q as a recursively enumerable set. The reader should recall that a set $A \subseteq \mathbb{N}$ is said to be recursively enumerable if there is an effective algorithmic procedure (e.g., a Turing machine) for enumerating all its elements; more specifically, a set $A \subseteq \mathbb{N}$ is a recursively enumerable set if there is a computable function f such that $A = \{f(1), f(2), ...\} = range(f)$. While Q is a non-recursively enumerable set if taken in the perspective of a UG, since a UG cannot have an unbounded set of terminals, it is a recursively enumerable set from the viewpoint of a WG, because in this every terminal character t_j in $t_1 ... t_k$, $1 \leq j \leq k$, is in the range of a total function mapping elements of $\mathbb{N}$ to names of terminal symbols.

Example 28. The following WG $\mathcal{G}_{AlphaK}$ describes the generation of the language *AlphaK*:[29]

[26] But a directed-graph representation, called derivation graph, is possible (see Augusto, 2021).

[27] Some restrictions and/or specifications (normal forms) apply for this result (e.g., Greibach, 1974; Turakainen, 1978); see van Leeuwen (1977) for a short review. See also Deussen (1975). Note, however, that there is a class of restricted WGs for which the parsing problem is solvable without loss of descriptive power; see Moritz (1989).

[28] The notation "t_i^n" abbreviates "$(t_i)^n$".

[29] Source: Grune & Jacobs (2008); with changes.

(r_M1)	N	::	n N; EMPTY.
(r_M2)	K	::	i; i K.
(r_M3)	EMPTY	::	.
(r_V1)	s	:	N i tail.
(r_V2)	N K tail	:	N K, N K i tail; EMPTY.
(r_V3)	N n K	:	K symbol, N K.
(r_V4)	K	:	EMPTY.

The strategy in this WG is to make every t_j, $1 \leq j \leq k$, in Q correspond to the protonotion σ_j = 'i^j symbol', in order to provide a finite mapping of symbol names to terminal symbols. This is done by means of a composition $p = g \circ f$ of the function $f : \{j\} \longrightarrow \{\mathsf{i}\}^j$ where $j \in \mathbb{N}$ is the subscript of some σ in the sequence $\sigma_1 ... \sigma_k$, and the function $g : \{\mathsf{i}\}^j \longrightarrow T$ where $T \in \mathcal{G}$. For instance, for the word $a^3b^3c^3 \in L\left(AlphaK^S\right)$ corresponding to $t_1^3t_2^3t_3^3$ we make the following mapping:

$$p(\mathsf{i}) = \mathsf{a},\ p\,(\mathsf{ii}) = \mathsf{b},\ p\,(\mathsf{iii}) = \mathsf{c}$$

We thus have i symbol = a symbol, ii symbol = b symbol, and iii symbol = c symbol and we apply the right side of the identities in the production rules. For the word "*fun*" the mapping is i symbol = f symbol, ii symbol = u symbol, and iii symbol = n symbol. Note that it is impossible to have the word "*funny*", because the terminal symbol n is bound to t_3 in $funny = t_1t_2t_3t_4t_5$. With this simple strategy we get a mapping that not only is finite but also hinders the repetition of any terminal symbol, since each terminal is so to say bound to a unique position in the sequence $t_1 ... t_k$. With this, the hyper-rules are easily explained: r_V1 describes the production of a sequence of Ki symbols each n times, r_V2 does so for a sequence of K symbols each n times concatenated with Ki symbol n times, etc.

Acknowledgments

I thank the reviewer for thoroughly reading the submitted manuscript and suggesting ways to improve it. My thanks also to W. S. Saba, a member of the AEB, for handling the blind-review process.

References

Augusto, L. M. (2021). *Languages, machines, and classical computation.* 3rd ed. London: College Publications.

Baker, J. L. (1970). *Some formal properties of the syntax of* ALGOL 68. Ph.D. Dissertation. University of Washington.

Baker, J. L. (1972). Grammars with structured vocabulary: A model for the ALGOL-68 definition. *Information and Control*, *20*, 351-395.

Bryant, B. R. & Lee, B.-S. (2002). Two-level grammar as an object-oriented requirements specification language. In R. H. Sprague, Jr. (ed.), *Proceedings of the 35th Annual Hawaii International Conference on System Sciences* (pp. 3627-3636). Los Alamos, CA: IEEE Computer Society.

Bryant, B. R., Edupuganty, B., Sundararaghavan, K. R., & Takaoka, T. (1988). Two-level grammar: Data flow English for functional and logic programming. In *Proceedings of the 1988 ACM 16th Annual Conference on Computer Science* (pp. 469-474). New York, NY: ACM.

Chomsky, N. (1964). *Current issues in linguistic theory.* The Hague & Paris: Muton.

Cleaveland, J. C. & Uzgalis, R. C. (1977). *Grammars for programming languages.* New York & Oxford: North Holland.

de Chastellier, G. & Colmerauer, A. (1969). W-grammar. In *Proceedings of the 24th National Conference ACM '69* (pp. 511-518). New York, NY: ACM.

De Graaf, J. & Ollongren, A. (1984). On two-level grammars. *International Journal of Computer Mathematics*, *15*(1-4), 269-288.

Deussen, P. (1975). A decidability criterion for van Wijngaarden grammars. *Acta Informatica*, *5*, 353–375.

Edupuganty, B. & Bryant, B. R. (1989). Two-level grammar as a functional programming language. *The Computer Journal*, *32*(1), 36-44.

Greibach, S. (1974). Some restrictions on W-grammars. *International Journal of Computer and Information Science*, *3*, 289-327.

Grune, D. (1993). Two-level grammars are more expressive than type 0 grammars. Or are they? *SIGPLAN Notices*, *28*(8), 43-45.

Grune, D. & Jacobs, C. J. H. (2008). *Parsing techniques: A practical guide.* 2nd ed. New York, NY: Springer.

Hockett, C. F. (1958). *A course in modern linguistics.* New York: Macmillan.

Janssen, T. M. V. (1975). An arithmetization of van Wijngaarden grammar. Report ZW 44/75. Amsterdam: Mathematical Centre.

Koster, C. H. A. (1974). Two-level grammars. In G. Goos & J. Hartmanis (eds.), *Compiler construction: An advanced course* (pp. 146-156). Lecture Notes in Computer Science 21. Berlin: Springer.

Koster, C. H. A. (1993). Informatics and syntax. In J. Darski & Z. Vetulani (eds.), *Akten des 26. Linguistischen Kolloquiums, Poznań 1992.* (pp. 43-54). Tübingen: Max Niemeyer Verlag.

Kupka, I. (1980). van Wijngaarden grammars as a special information processing model. In P. Dembińsky (ed.), *Mathematical foundations of computer science, MFCS 1980* (pp. 387-401). Berlin & Heidelberg: Springer.

Lindsey, C. H. (1996). A history of ALGOL 68. In T. J. Bergin & R. G. Gibson (eds.), *History of programming languages. Vol. II* (pp. 27-96). New York: ACM Press.

Lindsey, C. H. & van der Meulen, S. G. (1977). *Informal introduction to* ALGOL 68. 2nd ed. Amsterdam, etc.: North-Holland.

Małuszyńsky, J. (1984). Towards a programming language based on the notion of two-level grammar. *Theoretical Computer Science*, *28*, 13-43.

Mateescu, A. & Salomaa, A. (1997). Aspects of classical language theory. In In G. Rozenberg & A. Salomaa (eds.), *Handbook of formal languages. Vol. 1: Word, language, grammar* (pp. 175-251). Berlin & Heidelberg: Springer.

Meertens, L. (1969). On the generation of ALGOL 68 programs involving infinite modes. *Algol Bulletin*, *30*, 90-92.

Moritz, M. (1989). *Description and analysis of static semantics by fixed point equations*. Ph.D. thesis. Nijmegen: Bloembergen Santee bv [printer].

Sintzoff, M. (1967). Existence of a Van Wijngaarden system for every recursively enumerable set. *Annales de la Société Scientifique de Bruxelles*, *81*, 115-118.

Staiger, L. (1997). ω-languages. In G. Rozenberg & A. Salomaa (eds.), *Handbook of formal languages. Vol. 3: Beyond words* (pp. 339-387). Berlin & Heidelberg: Springer.

Turakainen, P. (1978). On characterization of recursively enumerable languages in terms of linear languages and VW-grammars. *Indagationes Mathematicae*, *40*, 145-153.

van Leeuwen, J. (1977). Recursively enumerable languages and van Wijngaarden grammars. *Indagationes Mathematicae*, *39*, 29-39.

van Wijngaarden, A. (1965). Orthogonal design and description of a formal language. Report MR 76. Amsterdam: Stichting Mathematisch Centrum.

van Wijngaarden, A. (ed.) (1969). *Report on the algorithmic language ALGOL 68*. Berlin, etc.: Springer. (Reprint of article in *Numerische Mathematik*, *14*, 79-218.)

van Wijngaarden, A. (1974). The generative power of two-level grammars. In J. Loeckx (ed.), *Automata, languages and programming. ICALP 1974* (pp. 9-16). Lecture Notes in Computer Science 14. Berlin: Springer.

van Wijngaarden, A., Mailloux, B. J., Peck, J. E. L., Koster, C. H. A., Sintzoff, M., Lindsey, C. H., Meertens, L. G. L. T., & Fisker, R. G. (eds.) (1976). *Revised report on the algorithmic language ALGOL 68*. Berlin, etc.: Springer.

Visser, E. (1997). *Polymorphic syntax definition*. Report P9701 of the Programming Research Group of the University of Amsterdam.

Online Resources

[1] https://jmvdveer.home.xs4all.nl/en.index.html

Cite this article as:

Augusto, L. M. (2023). Two-level grammars: Some interesting properties of van Wijngaarden grammars. *Omega - Journal of Formal Languages*, *1*, 3-34.

EDITORIAL INFORMATION

Editor-in-chief: Luis M. Augusto

Reviewer:[a] Anonymous

Received: Nov. 1, 2022; **Accepted**: Dec. 18, 2022

Edited on April 18th, 2023: the sentence "the family of languages generated by WGs is precisely the family of lossless CSLs" was corrected as "the family of languages generated by lossless WGs is precisely the family of CSLs."

[a]Blind review