The van Wijngaarden Grammars: A Syntax Primer with Decidable Restrictions

Luis M. Augusto*

Editor-in-Chief

Journal of Knowledge Structures & Systems

October 2023

Vol.: 4 Issue: 2 Pages: 1-39

Abstract

Expressiveness and decidability are two core aspects of programming languages that should be thoroughly known by those who use them; this includes knowledge of their metalanguages a.k.a. formal grammars. The van Wijngaarden grammars (WGs) are capable of generating all the languages in the Chomsky hierarchy and beyond; this makes them a relevant tool in the design of (more) expressive programming languages. But this expressiveness comes at a very high cost: The syntax of WGs is extremely complex and the decision problem for the generated languages is generally unsolvable. With this in mind, I provide here a short primer of the syntax of WGs, which includes syntactic restrictions that guarantee decidability for the corresponding generated languages.

Key words: van Wijngaarden Grammars; Chomsky Grammars; Generative Power; Descriptive Power; The Membership Problem; The Parsing Problem

1 Introduction

Programming languages have been around for quite a long time now, but only recently did scholars begin to be concerned with the problem of their knowledge, or how and to what extent programmers understand the languages they create programs with (e.g., Clayton et al., 1998; Soloway & Ehrlich, 1984). This knowledge requires in turn that their metalanguages, a.k.a. formal grammars, be thoroughly understood. Interestingly, the same type of formal grammar can describe (many) different programming languages, as is well known from the case of the context-free grammars (CFGs), by which most contemporary programming languages can be described (e.g., Sebesta,

*✉ luis.ml.augusto@gmail.com

2012). By a slight abuse of terminology, we can in fact confuse *generation* and *description*, and we are thus justified in saying that these programming languages are all context-free languages.

There is, however, a type of formal grammar that, despite being CFG-based, can generate not only context-free languages, but also context-sensitive and recursively enumerable languages, and even beyond, in the sense that it can handle infinite alphabets and infinite sets of production rules. I speak here of the van Wijngaarden grammars (abbr.: WGs).[1] These grammars were originally conceived by A. van Wijngaarden as the metalanguage of the programming language ALGOL 68 (van Wijngaarden et al., 1969; van Wijngaarden et al., 1976), but were soon thereafter put into relation with other programming languages (e.g., Bryant et al., 1988; Edupuganty & Bryant, 1989), as well as other applications, such as in computational linguistics and automated translation (e.g., Chastellier & Colmerauer, 1969). This *generative* power is supported by a unique *descriptive* power that comes at a high cost as far as the syntax is concerned: The syntax of WGs is extremely complex and its knowledge, or adequate understanding, requires an extra effort, especially if one is concerned with decidability issues. In order to facilitate this effort I give here a short primer of the syntax of WGs. Additionally, I provide the main syntactic restrictions that guarantee decidability of the languages generated by WGs while preserving their expressive potentialities. I refer the reader to Augusto (2023) for an elaboration on WGs from the viewpoint of formal grammars, and to Cleaveland & Uzgalis (1977) for a comprehensive discussion from the programming perspective.[2]

2 The Syntax of WGs

A presentation of WG syntax requires three aspects to be handled in tandem: (1) The alphabet and the strings up to the words derived by means of (2) interacting rewriting rules at/in (3) distinct levels and grammars. The terminology and notation is partly that of the Chomsky grammars, of which a summary is given here.[3] In order to distinguish a WG from a Chomsky grammar G the former shall be denoted by $\mathcal{G}$.

2.1 Formal Languages and Chomsky Grammars: Basic Terminology and Notation

A formal language L is a set of strings of symbols specified by a formal grammar G. Chomsky grammars are the paradigmatic formal grammars. 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

[1] Another frequent abbreviation is "W-grammars". WGs are also often confused with the two-level grammars, of which in fact they are only a sub-class.

[2] This article has been submitted to reviewers in May 2023 and reports, when finally received, are expected to motivate edits, as is usually the case in the review process. I write "finally," because I have been cut off from all online communication media by a lamer located in Calle Riojanos, 2, 1ª 4, 28018 Madrid. This lamer, who calls himself privately Zorra Negra, is being assisted in his activities by the rest of his family (car plate: four five double-seven lfg). This prolonged event is also severely disrupting further work of mine on WGs (e.g., Augusto, 2024 [forthcoming]).

[3] See Augusto (2021) for a textbook centered in the Chomsky hierarchy. I consider here the Chomsky hierarchy as having the grammars of Type 0 at the top and the Type-3 grammars at the bottom. This is thus a hierarchical perspective with full expressiveness located at the top.

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 (abbr.: r) 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 through 3 of the Chomsky hierarchy, with 3 being the most restricted grammar type. 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 a 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 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. A derivation of a word $w \in L(G)$ from $S \in V_G$ is thus a finite sequence of steps each of which is the application of some production rule in $\{r_i\}_{i=1}^k \subseteq R_G$:

$$\mathscr{D}_w = S \overset{1}{\underset{r_1}{\Longrightarrow}} \vartheta_1 \overset{2}{\underset{r_i}{\Longrightarrow}} \vartheta_2 \overset{3}{\underset{r_i}{\Longrightarrow}} ... \overset{n}{\underset{r_i}{\Longrightarrow}} w$$

The sequence of the k rules applied in n steps in a derivation $\mathscr{D}_w$ is called a *parse* and is defined as:

$$\mathscr{P}_w = r_{1,1} r_{i,2} r_{i,3} ... r_{i,n}$$

Clearly, every derivation $\mathscr{D}_w$ corresponds to a parse $\mathscr{P}_w$, and reciprocally. Other similar correspondences are found for finite derivation trees $\mathcal{T}_w$ (then also called parse trees) and finite computations by abstract machines $\mathcal{M}_w$ that accept or recognize w. The Kleene star is a useful abstraction when we do not care to count the number of derivation steps, because for practical ends it denotes a bounded sequence (differently from the superscript ω, which denotes an infinite sequence). When applied over an arbitrary set of symbols Σ, the Kleene star denotes the (possibly infinite) set of all strings – including the empty string ϵ – over Σ; this set is denoted by Σ^*.

The Type-0 grammars generate the (class of) recursively enumerable languages (RELs), the Type-1 grammars do so for the context-sensitive languages (CSLs), the Type-2 grammars generate the context-free languages (CFLs) and the Type-3 grammars generate the regular languages (REGs). We have the *extended* Chomsky hierarchy, or class-inclusion relation

$$\mathbf{RGL} \subset \mathbf{CFL} \subset \mathbf{CSL} \subset \mathbf{RCL} \subset \mathbf{REL}$$

where **RCL** denotes the class of recursive languages, for which no specific generating grammar type is known to exist. The prototypical languages in the Chomsky hierarchy

are: $L \in$ **RGL** is $L = \{a^n | n > 0\}$, $L \in$ **CFL** is $L = \{a^n b^n | n > 0\}$, $L \in$ **CSL** is $L = \{a^n b^n c^n | n > 0\}$, $L \in$ **REL** is $L = \{a^{2^n} | n \geq 0\}$.

2.2 The Basic Components of WGs: Alphabet, Sequences, and Words

WGs are not Chomsky grammars, but they share with them many features; in particular, WGs can generate all the languages in the Chomsky hierarchy, reason why the contents of Section 2.1 should be borne in mind throughout this article.

Definition 1. The alphabet Σ of a WG $\mathcal{G}$ is constituted by *syntactic marks*, namely (i) small syntactic marks ("a", "b", "c", ..., "y", "z", "(", ")"), (ii) large syntactic marks ("A", "B", "C", ..., "Y", "Z", "0", "1", ..., "9"), and (iii) other syntactic marks (".", ",", ":", ";", "", "-", "*"). An additional syntactic mark is ϵ, the *empty* syntactic mark.

The syntactic marks build finite sequences $\overline{\sigma} = \sigma_1 \sigma_2 ... \sigma_n$ 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 $i < x < j$.

Definition 2. The finite sequences $\overline{\sigma}$ of syntactic marks of a WG such that $|\overline{\sigma}| \geq 0$ are called *notions* and there are three types thereof, to wit, *protonotions*, *metanotions*, and *hypernotions*.

The different sets of notions and their relation to the syntactic marks are shown in Table 1. Just as in the case of sets of strings in Chomsky grammars, given a set of sequences $S = \{\overline{\sigma}_1, ..., \overline{\sigma}_i, ...\}$ we write S^* for $S \cup \{\epsilon\}$ and we write S^+ for $S - \{\epsilon\}$. V and T denote sets of *protovariables* and *terminals*, respectively. To contrast with the members of V, metanotions in M can also be called metavariables. In fact, metanotions can be either *metavariables* or *metaterminals*.

Table 1: Notions and syntactic marks. (SSM and LSM abbreviate respectively "small syntactic marks" and "large syntactic marks".)

NOTION & SET	SSM	LSM
Protonotion $x \in [P^* = (V \cup T)^*]$: Protovariable $v \in V$ Terminal $t \in T$	✓	
Metanotion $X \in M$		✓
Hypernotion $\langle \mathcal{X} \rangle \in [N^* = (M \cup V)^*]$	✓	✓

Example 3. 'letter a' is a protonotion, namely the protovariable letter a; 'letter a symbol' is a terminal protonotion, namely the typographical character *a*. "ALPHA" is a metanotion, namely the metanotion ALPHA. "letter ALPHA symbol" is a terminal metanotion, or metaterminal. 'reference to INTREAL' is a hypernotion, namely the hypernotion reference to INTREAL.

Blank spaces are added for readability: 'reference-to-INTREAL' and 'reference-toINTREAL' are two equally admissible ways of writing the hypernotion 'reference to INTREAL'. In the following example, and henceforth, the mention symbols ('.' and ".") are omitted, the font sufficing to segregate WG notions from the main text.

Example 4. The word "*age* := *53*" is formed by the terminal protonotions letter a symbol, letter g symbol, letter e symbol, becomes symbol, digit five symbol, and digit three symbol.

From Example 4, it should be understood that terminal protonotions always end in the protonotion 'symbol' and they are replaced by typographical characters in a terminal sequence, i.e. a *word*.[4] Every typographical character, including punctuation and other symbols, is associated with a terminal protonotion in a *representation table* (e.g., Table 2). This allows for the abbreviation of, say, letter a symbol as simply a, where a is a typographical character.[5] This association is actually a well-defined function mapping directly protovariables and indirectly metavariables to terminals:

Definition 5. Let there be given the sets of notions M, V, and T. A *(terminal) representation function* is defined as:

$$\mathsf{symbol} : (M \cup V) \longrightarrow T$$

Remark 6. The representation mapping for protovariables is direct, reason why we can define it as

$$\varsigma : V \longrightarrow T$$

where ς replaces symbol for convenience. It is, however, indirect for metanotions, because it requires a composition $\varsigma \circ \varrho$ where ϱ (also replacing symbol) is the function

$$\varrho : M \longrightarrow V.$$

Thus, ϱ is actually a *substitution*, or *replacement*, *function* mapping metavariables and metaterminals to protovariables and it can instead be seen as a homomorphism h (see below), reason why for practical ends one may ignore this functional definition of ϱ.

2.3 Description Rules, Uniform Replacements, and Grammar Levels

As just seen, a terminal metanotion X symbol is not a terminal notion simpliciter. In order for the (terminal) metanotion X to become the (terminal) protonotion letter x (symbol), i.e. in order for the functions ϱ and ς to be applicable, two rules of type r_M and r_V are required that *describe* the admissible replacement $URR^{r_V j}_{r_M i}(\mathsf{X}, \mathsf{x})$, where URR abbreviates "uniform replacement rule," instructing that the metanotion X in metarule $r_M i$ is replaced uniformly in hyper-rule $r_V j$ by the protonotion x.[6]

[4]The protonotion symbol designates a construct, namely a terminal notion. Protonotions that designate constructs are called *paranotions*.

[5]This abbreviation in the WGs, which is only admissible for terminal protonotions, is typically not used in programming languages based on WGs, namely in ALGOL 68, the programming language entirely defined as a WG. See, e.g., van Wijngaarden et al. (1976) and Cleaveland & Uzgalis (1977).

[6]Importantly, the term "uniformly" does not mean that a $URR(\mathsf{X}, \mathsf{x})$ must be applied to all the hyper-rules in which X occurs; it only means that if X occurs more than once in a single hyper-rule, then in that particular hyper-rule it has to be uniformly replaced by x. As it will be seen, this is a major cause for ambiguity in WGs or even/consequently for undecidability of the languages generated by WGs (WLs, in abbreviation). On the other hand, to force a $URR(\mathsf{X}, \mathsf{x})$ for given X and x to be applied uniformly throughout the set of hyper-rules would severely diminish the expressive capabilities of WGs. This "dilemma" will be discussed at length below.

Table 2: A representation table.

Terminal Notation	**Typographical Character**
colon symbol	:
comma symbol	,
⋮	
becomes symbol	:=
letter a symbol	a
⋮	⋮
letter z symbol	z
digit zero symbol	0
⋮	⋮
digit nine symbol	9

Definition 7. These two rules are schematically defined as follows:[7]

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

$$(r_M) \qquad \underbrace{X_0}_{\text{LHS}} :: \underbrace{X_1; ....; X_n}_{\text{RHS}}.$$

  where $X_0 \in M$ and every *meta-alternative* $X_i \in (M \cup V)^*$ for $1 \leq i \leq n$.

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

$$(r_V) \qquad \underbrace{\langle X_0 \rangle}_{\text{LHS}} : \underbrace{\mathring{X}_1; ....; \mathring{X}_m}_{\text{RHS}}.$$

  where $\langle X_0 \rangle \in N$ and $\mathring{X}_i \in \left[(N \cup T \cup \{,\})^* = (M \cup P \cup \{,\})^*\right]$ for $1 \leq i \leq m$, $\mathring{X}_i = (\gamma \langle \mathcal{X} \rangle \delta)_{i_1} ... (\gamma \langle \mathcal{X} \rangle \delta)_{i_k}$ where $\gamma, \delta \in (T \cup \{,\})^*$ are the *hyper-alternatives*.[8]

Remark 8. The rules above in Def. 7 are extensions of the basic rules

$$X_{\in M} :: Y_{\in (M \cup V)^*}$$

and

$$\langle X_0 \rangle_{\in N^+} : \mathring{X}_{\in (M \cup P \cup \{,\})^*}$$

for metarules and hyper-rules, respectively. The extensions above are actually abbreviations, in the sense that the number of rules of a WG can be significantly reduced, but the basic forms above are recruited for some computations.

Example 9. The terminal metanotion letter ALPHA symbol in Example 3 can be replaced by any terminal protonotion letter x symbol such that x = a, b, ..., z as described in the following two rules:

[7] As usually, LHS and RHS abbreviate "left-hand side" and "right-hand side."

[8] The notation $\langle \cdot \rangle$ is used here only for abstract rules, but the reader should note that it is often used in concrete rules in the WG literature.

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

Let us consider the two rules above as partly constituting the WG $\mathcal{G}_{Alpha1}$ (abbreviated as *Alpha1*). The hyper-rule can be abbreviated as follows:[9]

$$\mathsf{ALPHA} : ALPHA \ ; \ \mathsf{ALPHA}, \ ALPHA.$$

Obviously, in the first meta-alternative we have $\gamma = ALPHA \in T$, $\langle \mathcal{X} \rangle = \epsilon$, and $\delta = \epsilon$; as for the second meta-alternative, we have $\gamma = \epsilon$, $\langle \mathcal{X} \rangle = \mathsf{ALPHA} \in M$, and $\delta = (, ALPHA) \in (T \cup \{, \})$. But note that $ALPHA$ still counts as a metanotion: The metarule r_M defines the possible replacements for the metanotion ALPHA (or $ALPHA$) and the hyper-rule r_V specifies that ALPHA, when replaced by one of the alternative definitions in r_M – say, $URR^{r_V}_{r_M}(\mathsf{ALPHA}, \mathsf{a})$ –, can be replaced either indirectly by the typographical character a (via the representation function ς; first alternative) or iteratively by the protovariable a concatenated (denoted by the comma) with the typographical character a. In sum, whenever typographical symbols or "terminal" metanotions feature in a rule r_V, this is just a convenient abbreviation for x symbol or X symbol, respectively.

The metarule and the hyper-rule above actually belong to two distinct *levels* of the given WG such that we have

$$\frac{\text{Level 2}}{\text{Level 1}} = \frac{\mathcal{G}^2}{\mathcal{G}^1} = \frac{\text{Hyper-level}}{\text{Meta-level}} = \frac{R_V}{R_M}$$

where R_M and R_V are the sets of metarules and hyper-rules, respectively.[10] This justifies the membership of WGs in the class of *two-level grammars*, other members of which are, for instance, the attribute grammars (see Augusto, 2024 [forthcoming]).

Definition 10. We have now the definition of a WG $\mathcal{G}$ as the 7-tuple

$$\mathcal{G} = \left(\underbrace{M, V, R_M}_{\mathcal{G}^1}, \underbrace{N, T, S, R_V}_{\mathcal{G}^2} \right)$$

where $S = \{\langle s \rangle\}$, $S \subseteq N$.

Def. 10 allows us to abbreviate the 7-tuple above as the pair $\mathcal{G} = (\mathcal{G}^1, \mathcal{G}^2)$. For practical reasons, we often consider $\mathcal{G}^1 = (M, V, R_M, W)$, where $W \in M$. Also with practicality in mind, every rule should be sequentially numbered as $r_M 1, r_M 2, ..., r_M n$ and $r_V 1, r_V 2, ..., r_V m$; this done, we write URR^j_i where i, j abbreviate respectively $r_M i$, $r_V j$ (additionally, super-/subscripts of the form $m, ..., k$ can be used to reuse a single URR and super-/subscripts of the form $m - k$ denote transitivity).

[9] Often found in the WG literature in the form "⟨ALPHA⟩ → ALPHA; ⟨ALPHA⟩ ALPHA".

[10] The subscript V denotes that this set is constituted by rules for variables, i.e. meta- and protovariables alike.

2.4 Homomorphisms, Grammars, and (Meta-)Languages

Definition 11. The set of all the URRs of a WG $\mathcal{G}$ is defined as

$$URR_{R_M}^{R_V}(\mathcal{G}) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} URR_i^j = \mathcal{U}(\mathcal{G}).$$

Recall that $N = M \cup V$; thus, $\mathcal{G}^1$ and $\mathcal{G}^2$ share sets M and V, it being the case that we have $M \cap V = \emptyset$. Hence, $\mathcal{G}^1$ and $\mathcal{G}^2$ *interact* by means of the metarules and hyper-rules sharing these sets. Taken strictly mathematically, this interaction is based on a homomorphism $h \in Hom$ such that we have:

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

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

Obviously, we have $\widehat{\mathcal{G}} = \mathcal{U}(\mathcal{G})$. Recall now that a language L generated by a formal grammar $G = (V, T, S, R)$ is defined as

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

where S is the start symbol, "*" (Kleene star) denotes the reflexive and transitive closure of the relation $\underset{G}{\Longrightarrow} \subseteq (V \cup T)^+$ called (one-step) derivation, and R is the set of production rules.

Definition 12. In the case of a WG $\mathcal{G}$, we have the definition

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

where $\langle s \rangle$ is the (abstract) start hypernotion.

In both cases, $w = t_1 ... t_k$ is the word factorization into $k > 0$ typographical characters; when $k = 0$, we have either $L(\mathcal{G}) = \{\epsilon\}$ (as ϵ is not a typographical character) or $L(\mathcal{G}) = \emptyset$.

Example 13. The WG $\mathcal{G}_{Alpha1}$, abbreviated as *Alpha1*, is given by the following numbered rules:

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

These rules describe the generation of $L(Alpha1) = \{t^n | n \geq 1, t = a, b, ..., \text{ or } z\}$. For instance, by applying $URR_1^{1,2}(\mathsf{ALPHA}, \mathsf{a})$, the (rightmost) derivation of the word *aaa* is as follows:

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

Figure 1 shows the parse tree for this word.

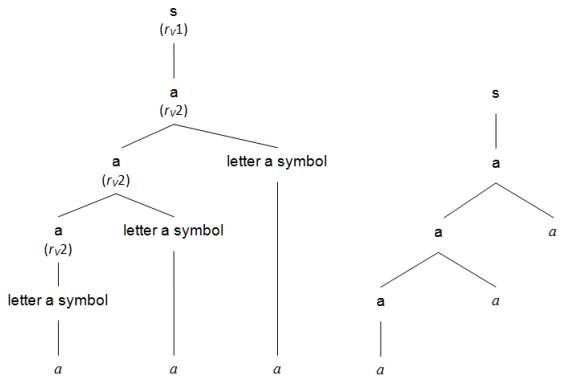


Figure 1: Parse tree for the word *aaa* generated by the WG *Alpha1*. The complete parse tree is on the left and its simplified version is on the right.

Definition 14. The definition of $L(\mathcal{G})$ given above is more correctly reformulated as

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

where $\mathcal{G}^D$ denotes *deep grammar*.

In the case at hand we have $Alpha1^D$ given in Example 13 above. This coinage is owed to the fact that any word $w = t_1 ... t_k$ generated by *Alpha1* has a unique parse tree for arbitrary k. But strictly taken $\langle s \rangle \overset{*}{\underset{\widehat{\mathcal{G}}}{\Longrightarrow}} w$ is *not* a derivation proper: It is rather the *description* of the *-step derivation from $\langle s \rangle$ to w based on $\widehat{\mathcal{G}}$. In effect, $\widehat{\mathcal{G}}$ describes the derivation, or generation, of any specific word $|w| = k$ for $k = 0, 1, 2, ...$ in the sense that a $URR_i^j(\mathsf{X}, \mathsf{x})$ applied simultaneously to all the hypernotions in $r_V j$ containing the metanotion X derives a *(strict) production rule* $r_S l \in R_S$ such that the protonotion x is in $r_S l$; when x occupies the LHS of a production rule it is called a *strict notion* and is abstractly denoted by x'. The set of strict notions N_S such that $|N_S| = n$ is thus defined as:

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

We have now the set of production rules

$$R_S = \bigcup R_S(r_V i)_{i=1}^k = \bigcup_{j=1}^{\infty} r_S j$$

where $R_S(r_V i)$ denotes the derivation of a rule in R_S from the i-th rule in R_V.

Definition 15. We define a production rule of a WG $\mathcal{G}$ as a rule of the form

$$(r_S) \qquad \underbrace{\mathsf{x}'}_{\mathrm{LHS}} : \underbrace{\dot{\mathsf{x}}_1; ...; \dot{\mathsf{x}}_k}_{\mathrm{RHS}}.$$

where each *strict alternative* $\dot{\mathsf{x}}_i = x_{i_1}^{(\prime)}, ..., x_{i_k}^{(\prime)} \in (P \cup \{,\})^*$.

Example 16. Using the elaboration above, the following set $R_{S_{x'}}$ of abstract production rules for any given strict notion x' and where x abbreviates letter x symbol is derived from $Alpha1^D$:

$$R_{S_{x'}}(Alpha1^D) = \left\{ \begin{array}{l} \mathsf{s} : \mathsf{x}'. \\ \mathsf{x}' : x; \mathsf{x}', x. \end{array} \right\}$$

Specifying x' by means of $\mathcal{U}(Alpha1)$ and removing commas for the sake of simplification gives the sets of concrete production rules

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

and

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

In this Example, $R_{S_{Alpha1}}(r_V 1)$ can be abbreviated as $\{\mathsf{s} : \mathsf{a}; \mathsf{b}; ...; \mathsf{z}.\}$, so that we have 27 (instead of 52) rules by $|R_{S_{Alpha1}}(r_V 1) \cup R_{S_{Alpha1}}(r_V 2)|$. This union is the surface grammar for *Alpha1*, denoted by $Alpha1^S$; this surface grammar is, in turn, the union of all the strict subgrammars for each of the strict notions $\mathsf{x}' = \mathsf{a}, \mathsf{b}, ..., \mathsf{z}$. Formally:

Definition 17. 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 $x_i^{'}$.

Table 3 summarizes the elaboration above. The language $L(\mathcal{G})$ can now be defined:

Definition 18. 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 $r_S = x'_{\in (N_S)^+} \rightarrow x_{\in P^*}$ and strings $\gamma, \delta \in P^*$ such that $\underbrace{\gamma, x', \delta}_{\alpha} \rightarrow \underbrace{\gamma, x, \delta}_{\beta}$, where the commas denote concatenation. Then, the language generated by $\mathcal{G}$ is defined as

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

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

Grammar & Level	**Rule**	**LHS**	**RHS**
$\mathcal{G}^{D.1}$	$r_M \in R_M$	Metanotion $W \in M$	Meta-alternatives $X_i \in (M \cup V)^*$
$\mathcal{G}^{D.2}$	$r_V \in R_V$	Hypernotion $\langle X_0 \rangle \in N$	Hyper-alternatives $\mathring{X}_i \in (N \cup T \cup \{,\})^*$
$\mathcal{G}^S$	$r_S \in R_S$	Strict Notion $x' \in N_S$	Strict Alternatives $\dot{x}_i \in (P \cup \{,\})^*$

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

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 $\alpha \underset{\widehat{\mathcal{G}}}{\Longrightarrow}_r \beta$. 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 $L\left(\mathcal{G}\right)$ such that we have:

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

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'.[11] 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:

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

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

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

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

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

We thus have the strict languages:

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

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

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

$$\vdots$$

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

For instance, the $URR_1^2(\mathsf{ALPHA}, \mathsf{a})$ applied to the entire set R_V of *Alpha1* gives a strict subgrammar, namely the set of strict productions R_{S_a} that generates the surface strict language $L_{\mathsf{a}}\left(Alpha1^S\right) = \{a^n | n > 0\}$:

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

Because a is a typographical character to be retrieved in a representation table (cf. Table 1), the language $L_{\mathsf{a}} = \{a^n | n > 0\}$ can also be called a *representation language* for $Alpha1^S$.[12] The strict metalanguage of *Alpha1* is thus the union of all the representation languages for $Alpha1^S$:

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

3 The Generative Power of WGs: Handling Infinity

3.1 Infinite Productions

One of the main advantages of the syntax elaborated on above is that it gives WGs the ability to handle infinite sets of production rules, an ability that is missing in the Chomsky grammars; as seen, this allows for the generation of not only infinite languages but also an infinite number of strict languages. In effect, if the set of productions of a WG $\mathcal{G}$ is finite, then the language $L(\mathcal{G})$ just is a Chomsky language.

Example 21. The language $L(Alpha2) = \left\{t^{2^n} | n \geq 0\right\}$ can be generated by the following WG $\mathcal{G}^D_{Alpha2}$:

(r_M1)	ALPHA	::	a; b; c; ...; y; z.
(r_M2)	N	::	ALPHA.
(r_V1)	s	:	N.
(r_V2)	N	:	NN; N symbol.

From R_V a finite set of strict production rules can be derived as follows:

(r_S1)	s	:	x.
(r_S2)	x	:	xx ; x symbol.

$Alpha2^S$ is equivalent to a CFG and thus $L\left(Alpha2^S\right)$ is a CFL. In effect, the production rules of $Alpha2^S$ mimic the (ambiguous) CFG $G = (\{S, A\}, \{a\}, S, R)$ where R is as follows:

[12] Baker (1972) extends this coinage to match grammar types with representation languages; e.g., $L = \{a^n | n > 0\}$ is a representation language for both type-3 Chomsky grammars and WGs that correspond to this Chomsky type.

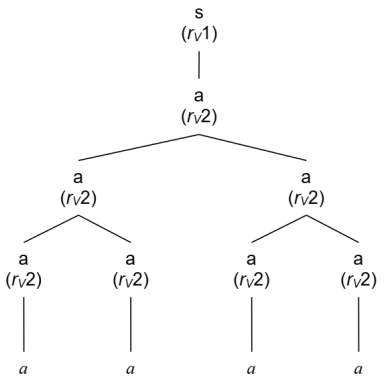


Figure 2: Parse tree for the word $a^4 \in L\left(Alpha2^S\right)$.

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

Figure 2 shows the parse tree for the word $a^4 \in L\left(Alpha2^S\right)$.

Recall from Section 2.1 above that the prototypical Chomsky languages are based on exponentiation over symbols of the alphabet. The WGs turn this feature into the ability to generate infinite production rules. The unique ability of deriving an infinite set of production rules at the surface level from a finite deep grammar is particularly well concretized in metarules of the forms $\mathsf{N} :: \mathsf{a}^n$, $\mathsf{N} :: \mathsf{Na}^n$, etc. or with notions that denote different exponents (e.g., $\mathsf{I} :: \mathsf{a}^i$, $\mathsf{J} :: \mathsf{a}^j$). With these metanotions any kind of exponentiation can in principle be implemented by a WG with an infinite set of production rules. Some simple examples follow; see, e.g., Greibach (1974) and Mateescu & Salomaa (1997) for further examples.

Example 22. The following WG $\mathcal{G}^D_{Alpha2n}$ describes the generation of the language $L\left(Alpha2n\right) = \left\{t^{2n} | n \geq 1, t = a, b, ..., \text{ or } z\right\}$.

(r_M1)	ALPHA	::	a; b; c; ...; y; z.
(r_M2)	N	::	ALPHA^n.
(r_V1)	s	:	N.
(r_V2)	N	:	N symbol, N symbol.

The simplifying strategy here relies on, say for the protonotion a, $URR^{1,2}_{1-2} = (\mathsf{N}, \mathsf{a}^n)$. The use of the terminal metanotion N symbol on the RHS of r_V2 guarantees that the iteration over N both is finite and produces an exponent of $2n$ for a. The strict grammar $R_{S_{Alpha2n}}$ for $\mathsf{x} =$ a, b, ..., z is constituted by the following production rules:

(r_S1)	s	:	x^n.
(r_S2)	x^n	:	$x^n x^n$.

$R_{S_{Alpha2n}}$ is obviously an infinite set, as $n \in \mathbb{N}$. However, the hyper-grammar allows for the construction of a finite parse tree for arbitrary n. Figure 3 (left) shows the parse tree of the string $a^8 \in L(Alpha2n)$. Note that if the RHS of r_V2 is only N symbol, we have a simplified version of *Alpha1*; this simplification can actually be optimized by making it so that there is a single production rule in $R_{S_{Alpha1}}$, to wit, $\mathsf{s} : x^n$ by designing r_V1 as $\mathsf{s} :$ N symbol. Adapted to *Alpha2n*, this simplification gives the single production rule $\mathsf{s} : x^{2n}$. This simplification requires that r_V2 be redesigned as $\mathsf{s} :$ N symbol, N symbol. Figure 3 (right) shows the parse tree of $a^8 \in L(Alpha2n)$ for this simplification, but care must be taken not to turn this and the above simplification into oversimplifications that might significantly diminish the expressive power of WGs. The frugal trees in Figure 3 might actually not be very useful in practical terms, say, for a parser. Because exponentiation in this language is based on the recursive function $f(n) = 2n$ it can be rather impractical to decide membership of a word to these oversimplified WGs.

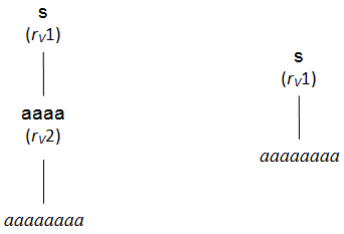


Figure 3: Parse trees of the string $a^8 \in L(Alpha2n)$.

One way to simplify without oversimplifying is to have URRs of the abstract form $URR\left(W, t^{f(n)}\right)$ for $n \in \mathbb{N}$. This exponentiation strategy is particularly useful in the case of recursive functions.

Example 23. The language $L(Alpha2)$ of Example 21 is based on the recursive function

$$f(n) = 2^n = \begin{cases} 1 & \text{for } n = 0 \\ 2\left(f(n-1)\right) & \text{for } n \geq 1 \end{cases}.$$

By changing r_V2 into $\mathsf{N} :$ N symbol, N; N symbol., we can design the set of strict production rules to be constituted by the two rules (r_S1) $\mathsf{s} :$ x. and (r_S2) $\mathsf{x}^{f(n)} : x^{f(n-1)}\mathsf{x}^{f(n-1)} ; x^{f(n)}$. Let us call this WG *Alpha2Imp*, where "Imp" abbreviates "improved." Obviously, rule r_S2 accounts for an infinite number of productions. For the strict language $L_{\mathsf{a}} = \left\{a^{2^n} | n \geq 0\right\}$ for arbitrary n, we have the derivation

$$\mathscr{D}^L_{a^{2^n}} = \mathsf{s} \underset{r_S1}{\Longrightarrow} \mathsf{a} \underset{r_S2}{\Longrightarrow} a\mathsf{a} \underset{r_S2}{\Longrightarrow} aa\mathsf{aa} \underset{r_S2}{\Longrightarrow} aaaa\mathsf{aaaa} \underset{r_S2}{\Longrightarrow} aaaaaaaa\mathsf{aaaaaaaa} \underset{r_S2}{\Longrightarrow} \ldots$$

$$\underset{r_S2}{\Longrightarrow} a^{f(n-1)}\mathsf{a}^{f(n-1)} \underset{r_S2}{\Longrightarrow} a^{f(n-1)}a^{f(n-1)} = a^{f(n)}$$

where to terminate the word we apply the second strict alternative of rule r_S2. (Note also that this is a leftmost derivation.)

Another efficient strategy adopted for WGs to increase (possibly) infinitely their generative power while keeping a limited number of rules 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

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

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 is more often than not assumed implicitly in the hyper-rules and if k is unbounded so that we have metarules of the form r^{ω}_{M}, then the descriptive power of a WG is virtually infinite.[13]

Example 24. Consider the language

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

where $T = \{a, b, ..., z\}$. ("*NoRep*" abbreviates "*Alpha No Repetition.*") 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. The WG $\mathcal{G}_{NoRep}$ requires only the following rules in the deep grammar:

$(r_{\mathsf{M}}1)$	ALPHA	::	a; b; ...; z.
$(r_{\mathsf{M}}2)$	LETTER	::	letter ALPHA.
$(r_{\mathsf{M}}3)$	TAG	::	LETTER; TAG LETTER.
$(r_{\mathsf{M}}4)$	EMPTY	::	.
$(r_{\mathsf{M}}5)$	NOTION	::	ALPHA; NOTION ALPHA.
$(r_{\mathsf{M}}6)$	NOTETY	::	EMPTY; NOTETY ALPHA.

[13] An additional positive effect of this strategy is the elimination of ambiguity, as it fixes a $URR(W_i, v)$ for all the hyper-rules in a hyper-grammar.

$(r\forall 1)$	s	:	TAG unique.
$(r\forall 2)$	LETTER TAG unique	:	LETTER symbol, TAG unique, where LETTER is not in TAG.
$(r\forall 3)$	LETTER unique	:	LETTER symbol.
$(r\forall 4)$	where LETTER1 is not in LETTER2 TAG	:	where LETTER1 isnt LETTER2, where LETTER1 is not in TAG.
$(r\forall 5)$	where LETTER1 is not in LETTER2	:	where LETTER1 isnt LETTER2.
$(r\forall 6)$	where letter ALPHA1 isnt letter ALPHA2	:	where ALPHA1 precedes ALPHA2 in abcde...wxyz; where ALPHA2 precedes ALPHA1 in abcde...wxyz.
$(r\forall 7)$	where ALPHA1 precedes ALPHA2 in NOTETY1 ALPHA1 NOTETY2 ALPHA2 NOTETY3	:	EMPTY.

For simplicity, we consider a three-letter word generated by this WG, but it should be obvious that words of arbitrary length $|w| \leq 26$ – there being exactly 26! words $|w| = 26$ – can be parsed with it. The strict language in consideration is thus:

$$L_{\text{fun}}\left(NoRep^{S}\right) = \{fun\}$$

Figure 4 shows the parse tree for this strict language. In this example (from Cleaveland & Uzgalis, 1977), use is made of protonotions beginning with 'where'.

Definition 25. Protonotions beginning with 'where', 'unless', 'is', etc., are called *predicates*.

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.

3.2 Infinite Alphabets

WGs can not only handle infinite sets of production rules (as just seen), but also infinite alphabets, what makes it that **WL**, the class of languages generated by WGs, properly contains the languages of the Chomsky hierarchy.

Example 26. The following WG $\mathcal{G}_{AlphaK}$ describes the generation of the language *AlphaK* defined as:[14]

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

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

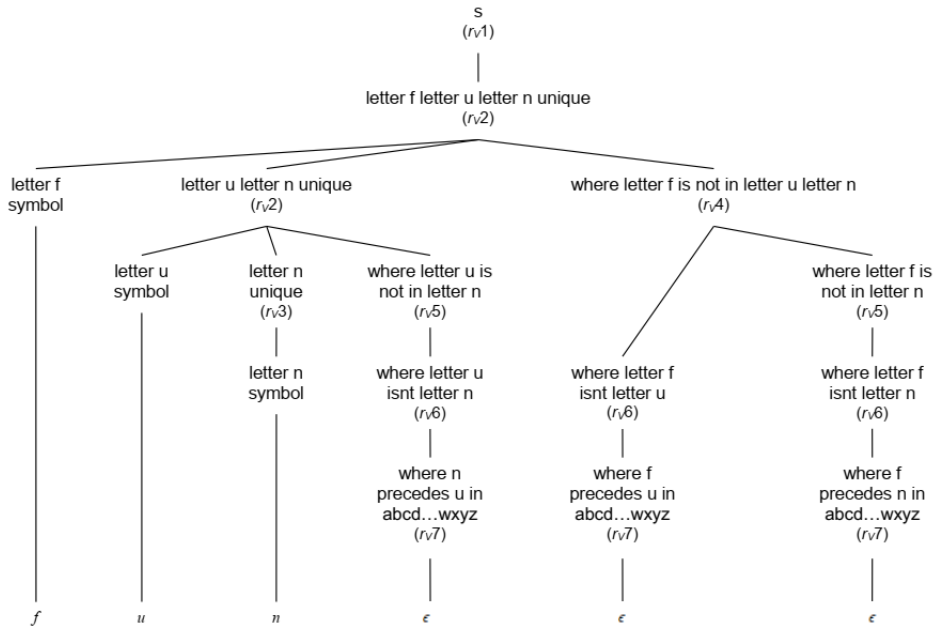


Figure 4: Parse tree $\mathcal{T}_{fun}$ of the string $fun \in L_{\mathsf{fun}}\left(NoRep^{S}\right)$.

(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.

This grammar guarantees that a given terminal symbol occurs only (n times) in a unique j-th position in $w = t_1...t_k$, $j \in \mathbb{N}$, by "binding" it by means of a mapping $f : \{j\} \longrightarrow \{\mathsf{i}\}^j$ from the set of protonotions in the sequence $\sigma_1...\sigma_k$ into the set of natural numbers such that we have $URR_2^{2,3}(\mathsf{K}_j, \mathsf{i}) = \mathsf{i}^j$; by means of a composition $p = g \circ f$ where $g : \{\mathsf{i}\}^j \longrightarrow T \subseteq P'$ we obtain the set of typographical symbols of a given word. The short word "*ffuunn*" is used here as an example. (The reader is welcome to create an infinite alphabet – in any case, some $|\Sigma| > 26$ – and parse words as long as their leisure might allow.) For this word, we have $URR_2^{2,3}(\mathsf{f}_1, \mathsf{i}) = \mathsf{i}$, $URR_2^{2,3}(\mathsf{u}_2, \mathsf{i}) = \mathsf{ii}$, and $URR_2^{2,3}(\mathsf{n}_3, \mathsf{i}) = \mathsf{iii}$, such that we obtain $p(\mathsf{i}) = \mathsf{f\,symbol}$; $p(\mathsf{ii}) = \mathsf{u\,symbol}$, and $p(\mathsf{iii}) = \mathsf{n\,symbol}$. Figure 5 shows the parse tree for this word of grammar *AlphaK*. Note in this parse tree that rule r_V4 is omitted; this rule assures us that if it is not applied after n occurrences of a given terminal protonotion, then an error has been detected and the word cannot be parsed.

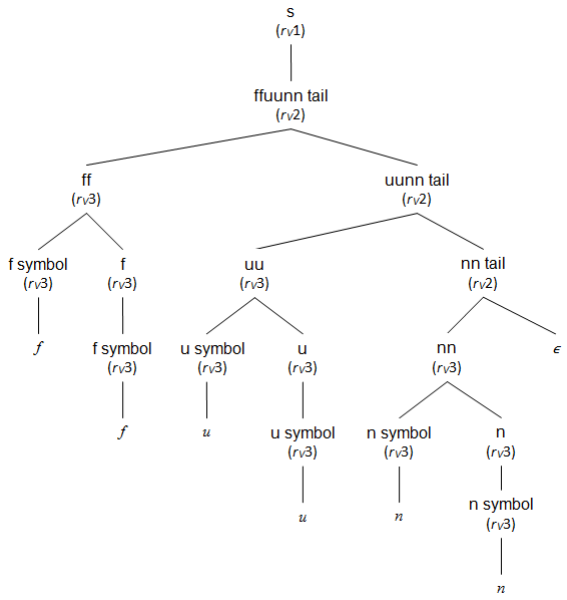


Figure 5: Parse tree of the word "*ffuunn*" generated by $\mathcal{G}_{AlphaK}$.

4 The Undecidability of WLs

4.1 Turing Machines Deciding and Accepting Languages

Example 21 above shows a central feature of WGs, to wit, the ability to generate all languages in the Chomsky hierarchy; this is an advantageous ability especially as we go down in the hierarchy, because we have more and more decidable questions for languages generated by WGs. In effect, when generated by a Chomsky grammar G, the language $L(G) = \left\{t^{2^n} | n \geq 0\right\}$ is *not* a CFL, because a CFG has no means to define recursively any exponent over a symbol, but it can be a CFL when generated by a WG, namely thanks to its ability to handle exponents in a very natural way.[15] On the other hand, even very simple WLs are generally undecidable for reasons that I discuss below.

Language decidability is a subject that recruits the highly intuitive Turing machine in the following sense:[16]

[15]For instance, it takes a UG $G = (\{L, R, D\}, \{a\}, S, R)$ with

$$R = \left\{ \begin{array}{c} S \to LaR \\ L \to LD \\ Da \to aaD \\ DR \to R \\ L \to \epsilon \\ R \to \epsilon \end{array} \right\}$$

to generate the language $L(G) = \left\{t^{2^n} | n \geq 0\right\}$. Compare with Example 21.

[16]See Augusto (2021) for an elaboration on Turing machines and core bibliography on the topics briefly discussed in this Section.

Definition 27. A language L is said to be *decidable*, or a *recursive* language (abbr.: RCL), if there is a Turing machine $\mathcal{M}$ that computes the characteristic function $\chi_L : \Sigma^* \longrightarrow \{0,1\}$ defined for any given string $w \in \Sigma^*$ as

$$\chi_L(w) = \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{otherwise} \end{cases}$$

where $x \overset{?}{\in} L$ is known as the *membership problem*. We then define L as

$$L_{DEC}(\mathcal{M}) = \{w \in \Sigma^* | \mathcal{M} \text{ computes } \chi_L(w)\}.$$

The membership problem is so central in formal language theory that it is in fact frequently spoken of as *the decision problem* (for a given language), and it suffices to say that language is decidable to mean that the membership problem has a solution; on the contrary, for other problems decidability is always a relative property, and we say, for instance, the decision problem for emptiness is (un)decidable.

A decision procedure for $\mathcal{M}$ equivalent to the computation of the characteristic function is the canonical enumeration of strings: Let an offline Turing machine $\mathcal{M}$ start with an empty input tape; at the end of the computation by $\mathcal{M}$, for every $x \in L(\mathcal{M})$ at some point the contents of its one-way write-only tape with the left-marker $\triangleright$ will be $\triangleright x_1 \# x_2 \# ... \# x_n \# x \#$ for some $n \geq 0$, $x_1, x_2, ..., x_n \in L(\mathcal{M})$, $x_1, x_2, ..., x_n, x$ are all distinct, and there is nothing printed after the last # (the blank symbol) if L is finite. This $\mathcal{M}$ *enumerates* L. If $\mathcal{M}$ enumerates all the strings of L in canonical order, i.e. shorter strings precede longer strings and strings of the same length are alphabetically ordered, then $L(\mathcal{M})$ is said to be decidable (or recursive) and we write $L_{DEC}(\mathcal{M})$. Fitting the inclusion relation **RCL** $\subseteq$ **REL** in the Chomsky hierarchy, a language is said to be *semi-decidable*, or *recursively enumerable* (r.e.), if there is a Turing machine $\mathcal{M}$ that when given input x halts in an accepting state if $x \in L$ (in which case we say that $\mathcal{M}$ accepts x such that we have $x \in L_{ACC}(\mathcal{M})$). Formally, we have the definition:

Definition 28. A language L is said to be accepted by a Turing machine $\mathcal{M}$ if, given a string $w \in L$, $\mathcal{M}$ halts in an accepting state. Thus, the language accepted by $\mathcal{M}$ is defined as

$$L_{ACC}(\mathcal{M}) = \{w \in \Sigma^* | q_0, w \vdash^*_{\mathcal{M}} q_a, w'\}$$

where

$$\underbrace{q_0, w}_{C_0} \vdash^*_{\mathcal{M}} \underbrace{q_a, w'}_{C_f}$$

denotes that starting in state 0 (q_0) with the input word w (i.e. the initial configuration C_0) after zero or more steps the Turing machine $\mathcal{M}$ has finished processing the input word and halts in an accepting state q_a with the output string $w' \in \Sigma \cup \Gamma$, for Γ the tape alphabet of $\mathcal{M}$ (C_f denotes "final configuration").

However, $\mathcal{M}$ may loop forever if $w \notin L$ instead of outright rejecting it (by, say, halting in a rejecting state). In other words, if a language is a REL we may end up with no solution to the membership problem, but this is a central problem in the theory of formal languages impacting on theoretical (e.g., scientific theories) and practical (e.g., compilation) aspects alike.

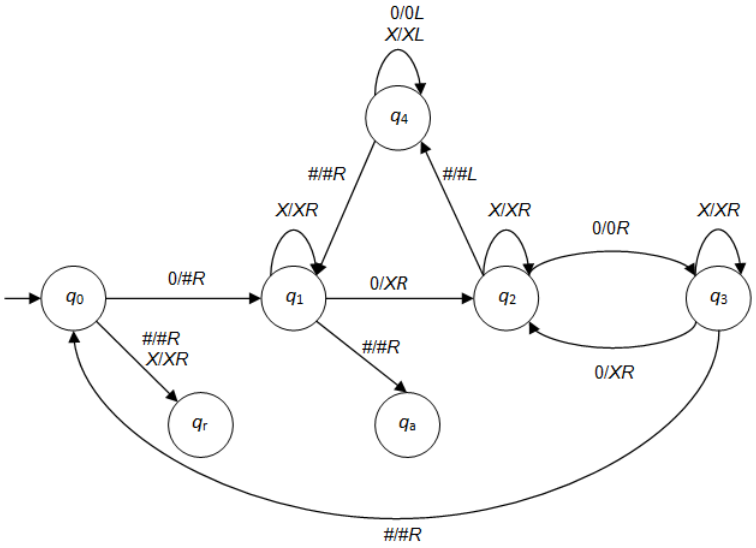


Figure 6: A total Turing machine that decides the language $L = \left\{0^{2^n} | n \geq 0\right\}$ over the alphabet $\Sigma = \{0\}^*$. (Source: Augusto, 2021.)

Example 29. Consider the language $L = \left\{0^{2^n} | n \geq 0\right\}$ over the alphabet $\Sigma = \{0\}^*$ (cf. Example 23). The Turing machine $\mathcal{M}$ of Figure 6 with one accepting state (q_a) and one rejecting state (q_r) decides L, despite not enumerating its strings itself: Given an input of m zeros followed by #, $\mathcal{M}$ either accepts it or rejects it; thus, with the help of $\mathcal{M}$ we can both enumerate the strings of L by increasing length such that we have the enumeration

$$L(\mathcal{M}) = \{0, 00, 0000, 00000000, ...\}$$

and enumerate the strings that are not in L so that we have

$$\overline{L(\mathcal{M})} = \Sigma^* - L(\mathcal{M}) = \{\epsilon, 000, 00000, 000000, 0000000, ...\}.$$

$\mathcal{M}$ halts on every input string $x \in \Sigma^*$, i.e. it is a total Turing machine, and this $L(\mathcal{M})$ is a RCL.

A language L that is not decidable (or recursive) or semi-decidable (or r.e.) is an *undecidable* language; this means that it is impossible to design a Turing machine $\mathcal{M}$ that accepts or recognizes the strings of either L or $\overline{L}$. As the number of Turing machines is countably infinite (more precisely, there are $\aleph_0$ Turing machines), but there are $2^{\aleph_0}$ languages, this leaves us with an uncountably infinite number of undecidable languages.[17]

[17] More strictly, there are exactly $\aleph_0$ partial recursive functions and there are exactly $\aleph_0$ recursive functions, but the number of functions is $2^{\aleph_0}$.

4.2 Recursiveness and Language (Un)Decidability

Taken strictly (in the WG sense), the language

$$L_{\mathsf{fun}}\left(AlphaK^{S}\right) = \{fun, ffuunn, fffuuunnn, ...\}$$

of Example 26 is in the Chomsky language class **REL**. This result – i.e. **REL** $\subseteq$ **WL** – was proven early on in the history of WGs (e.g., Janssen, 1975; Sintzoff, 1967; van Wijngaarden, 1974). While the RELs are already quite expressive with comparison to those languages below them in the Chomsky hierarchy, they are generally undecidable, namely due to infinity. This is in agreement with results formulated by E. Post on recursively enumerable (r.e.) sets of positive integers and their decision problems (Post, 1944). In this article, Post established the following crucial results in recursion theory for sets of positive integers:

Proposition 30. *The decision problem (DP) of a r.e. set is solvable or unsolvable according as the set is, or is not, recursive.*

Proof. Cf. Post (1944). □

Theorem 31. *(Post's Complementation Theorem) A set of positive integers is recursive iff both it and its complement with respect to* $\mathbb{Z}^{+}$ *are r.e.*

Proof. Cf. Post (1944). □

Corollary 32. *The DP of a r.e. set is recursively solvable iff its complement is r.e.*

Proof. Cf. Post (1944). □

Fact 33. *R.e. sets that are not recursive are always infinite.*

The above results hold for any set whose elements can be represented over the positive integers (e.g., by means of Gödel numbers or, more simply, by string length), so the extension of Post's results to languages is natural and we have the following crucial results with respect to RELs:

Theorem 34. *There is a REL L that is not recursive.*

Proof. By Theorem 31, this means that there is a REL L whose complement $\overline{L}$ is not a REL. In effect, given an input alphabet $\Sigma = \{a\}$, there are countably infinitely-many Turing machines $\{\mathcal{M}_i\}_{i=1}^{\aleph_0}$ such that there is some $\mathcal{M}_i$ associated to each REL $L(\mathcal{M}_i)$ over Σ. Consider now a new language L such that, for each $i \geq 1$, with respect to string a^i we have

$$a^i \in L \quad \text{iff} \quad a^i \in L(\mathcal{M}_i).$$

Consider now the complement of L:

$$(1) \qquad \overline{L} = \left\{a^i | a^i \notin L(\mathcal{M}_i)\right\}$$

Both L and $\overline{L}$ are well defined, but the latter is not r.e. This can be shown by contradiction. We assume that $\overline{L}$ is r.e. Then, there must be some $\mathcal{M}_j$ such that:

$$(2) \qquad \overline{L} = L(\mathcal{M}_j)$$

Let us consider now a string a^j. If $a^j \in \overline{L}$, then by (2) $a^j \in L(\mathcal{M}_j)$ which entails by (1) that $a^j \notin \overline{L}$; if $a^j \in L$, then $a^j \notin \overline{L}$, which by (2) entails that $a^j \notin L(\mathcal{M}_j)$, which by (1) entails that $a^j \in \overline{L}$. In either case, we reach a contradiction, so we conclude that $\overline{L}$ is not r.e. Proving that L is a REL is easy and I leave this part of the proof as an exercise. □

Corollary 35. *There is a REL whose DP is recursively unsolvable.*

4.3 Decidability and the Parsing Problem for WGs

The central result in the theory of WGs was expressed in Sintzoff (1967) and it follows directly from Corollary 35 immediately above:

Theorem 36. *The DP for a given WG $\mathcal{G}$ is recursively unsolvable.*

The brief elaboration in 4.1-2 above is meant to convey the fact that the Turing machine establishes the theoretical measure of decidability in formal languages taken as sets of strings; this is known as the Church-Turing Thesis. For practical ends, however, we may (need to) relax this measure and ask instead for the existence of at least one parse.

Definition 37. Given a pair (w, G), we wish to find the set $\{\mathscr{P}_{w,i}\}_{i=1}^{k}$ of parses for string w and grammar G such that $w \in L(G)$. This query is the *parsing problem*.

For convenience, I shall consider any of a finite sequence of configurations of a machine $\mathcal{M}_w$ that recognizes w, a finite tree $\mathcal{T}_w$, or a finite derivation $\mathscr{D}_w$ as equivalent to a parse $\mathscr{P}_w$.

Example 38. Given the language $L(\mathcal{M}) = \left\{0^{2^n} | n \geq 0\right\}$ for the Turing machine $\mathcal{M}$ of Figure 6 with input alphabet $\Sigma = \{0\}$, tape alphabet $\Gamma = \Sigma \cup \{X, \#\}$ and halting set $H = \{q_a, q_r\}$, the following is the finite sequence of configurations of $\mathcal{M}$ when accepting the string 0000:[18]

$$\underbrace{q_0, \underline{0}000\#\#}_{C_0} \vdash q_1, \#\underline{0}00\#\# \vdash q_2, \#X\underline{0}0\#\# \vdash q_3, \#X0\underline{0}\#\#$$

$$q_2, \#X0X\underline{\#}\# \vdash q_4, \#X0\underline{X}\#\# \vdash q_4, \#X\underline{0}X\#\# \vdash q_4, \#\underline{X}0X\#\#$$

$$q_1, \underline{\#}XXX\#\# \vdash q_1, \#\underline{X}0X\#\# \vdash q_1\#X\underline{0}X\#\# \vdash q_2, \#XX\underline{X}\#\# \vdash q_2, \#XXX\underline{\#}\#$$

$$q_4, \#XX\underline{X}\#\# \vdash q_4, \#X\underline{X}X\#\# \vdash q_4, \#\underline{X}XX\#\# \vdash q_1, \underline{\#}XXX\#\# \vdash q_1, \#\underline{X}XX\#\#$$

$$q_1, \#X\underline{X}X\#\# \vdash q_1, \#XX\underline{X}\#\# \vdash q_1, \#XXX\underline{\#}\# \vdash \underbrace{q_a, \#XXX\#\underline{\#}}_{C_f}$$

It is important to remark that the parse of a word generated by a WG $\mathcal{G}$ is completed over the set of hyper-rules in $\mathcal{G}^2$, and *not* over the set R_S in $\mathcal{G}^S$. But whereas, just as in the case of CFGs, in $\mathcal{G}^S$ we have it that there is a one-one correspondence between

[18] Underlined (blank) symbols are those that $\mathcal{M}$ is currently reading. $\vdash_{\mathcal{M}}$ is abbreviated as $\vdash$. It is assumed that the tape of $\mathcal{M}$ is infinite to the right and terminates on the left in the first symbol of the string.

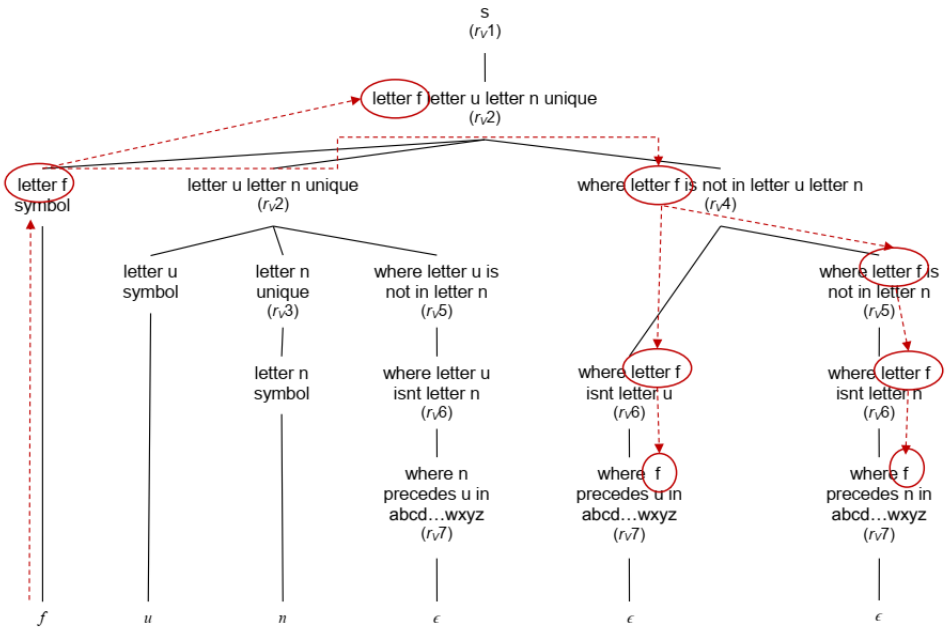


Figure 7: Parse tree of the word $fun \in L\left(NoRep^{S}\right)$ showing metanotion propagation for $URR\left(\mathsf{ALPHA},\mathsf{f}\right)$ in broken directed lines (in red).

leftmost derivations and left parses, in $\mathcal{G}^2$ this does not hold, an aspect that I will discuss and exemplify below. Moreover, the fact that a parse can be constructed for a word generated by a WG does not by itself guarantee that the corresponding WL is decidable, namely because of the complexity of the parse.

Example 39. Figure 7 shows the parse tree of the word $fun \in L\left(NoRep^{S}\right)$ by taking into consideration the direction of the propagation of metanotions, a topic elaborated on in Williams (1985). I give only the propagation of the metanotion ALPHA as given in $URR\left(\mathsf{ALPHA},\mathsf{f}\right)$. The reader can easily do the same for the letters "u" and "n" and then compare with Figure 4 above.

Besides high levels of word-parse complexity, WGs are prone to a plethora of causes for ambiguity (see below). In order to eliminate ambiguity we may require that the parser output at most one parse, namely at most one left parse $\mathscr{P}_{w}^{L}$ for the leftmost derivation $\mathscr{D}_{w}^{L}$. This restriction is particularly relevant for WGs given the following *negative* result:

Theorem 40. *(Wegner, 1980) It is undecidable whether an arbitrary WG $\mathcal{G}$ has at most one left parse for each leftmost derivation.*

This result in fact dictates the undecidability of WGs expressed in Theorem 36 above. Before giving the proof for this theorem, it will prove useful to provide an example.

Example 41. Consider the following sets of (abstract) meta- and hyper-rules for a given WG $\mathcal{G}$ in which $n \geq 1$:

$$R_M = \left\{ \begin{array}{cc} (1) & \mathsf{A} \to \mathsf{aA;a.} \\ (2) & \mathsf{B} \to \mathsf{A.} \end{array} \right\}$$

$$R_V = \left\{ \begin{array}{cc} (1) & \mathsf{s} \to \mathsf{A;B.} \\ (2) & \mathsf{A} \to \mathsf{a}^n. \\ (3) & \mathsf{B} \to \mathsf{a}^n. \\ (4) & \mathsf{a}^n \to a. \end{array} \right\}$$

Obviously, by hyper-rule 4 this WG has a single strict language such that $L_{\mathsf{a}} = \{a\}$. Then, there are infinitely many leftmost derivations for the single left parse $\mathscr{P}_a^L = r_V 1, r_V 2, r_V 4$:

$$\mathsf{s} \underset{r_V 1}{\Longrightarrow}{}_l \mathsf{A} \underset{r_V 2}{\Longrightarrow}{}_l \mathsf{a} \underset{r_V 4}{\Longrightarrow}{}_l a$$

$$\mathsf{s} \underset{r_V 1}{\Longrightarrow}{}_l \mathsf{A} \underset{r_V 2}{\Longrightarrow}{}_l \mathsf{a}^2 \underset{r_V 4}{\Longrightarrow}{}_l a$$

$$\vdots$$

$$\mathsf{s} \underset{r_V 1}{\Longrightarrow}{}_l \mathsf{A} \underset{r_V 2}{\Longrightarrow}{}_l \mathsf{a}^i \underset{r_V 4}{\Longrightarrow}{}_l a$$

$$\vdots$$

Conversely, there are two left parses for the leftmost derivation $\mathscr{D}_a^L = \mathsf{s} \overset{2}{\Longrightarrow}_l \mathsf{a} \overset{i}{\Longrightarrow}_l \mathsf{a}^i \Longrightarrow_l a$, to wit:

$$\left(\mathscr{P}_a^L\right)_1 = r_V 1, r_V 2, r_V 4$$

$$\left(\mathscr{P}_a^L\right)_2 = r_V 1, r_V 3, r_V 4$$

The proof of Theorem 40 can now be given:

Proof. Let us consider two arbitrary CFGs $G_1 = (V_1, T_1, S_1, R_1)$ and $G_2 = (V_2, T_2, S_2, R_2)$ such that

$$\mathcal{G} = \left(\underbrace{\underbrace{V_1 \cup V_2}_{M}, \underbrace{T_1 \cup T_2 \cup \{s\}}_{V}, \underbrace{R_1 \cup R_2}_{R_M}}_{\mathcal{G}^1}, \underbrace{\underbrace{\{\langle s\rangle, \langle S_1\rangle, \langle S_2\rangle\}}_{N}, \underbrace{\{a\}}_{T}, \langle s\rangle, R_V}_{\mathcal{G}^2} \right)$$

where

$$R_V = \left\{ \begin{array}{l} \langle s\rangle \to S_1 \\ \langle S_1\rangle \to a \\ \langle S_2\rangle \to a \end{array} \right\}$$

is a WG. Then, there is a (leftmost) derivation with more than one left parse iff $L(G_1) \cap L(G_2) \neq \emptyset$, but this is known to be an undecidable problem for CFGs. □

The inverse property that for each left parse there is exactly one leftmost derivation is also generally undecidable and shown with a similar construction.

5 Taming the Generative Power of WGs: Decidable Restrictions

Given the inclusion relation above and their ability to handle both infinite sets of production rules and infinite alphabets, we have it that WLs are potential candidates to undecidability. In effect, the language $L\left(AlphaK^S\right)$ segregates the class of languages generated by WGs from the Chomsky languages in the sense that the WGs extend the expressive capabilities of the Chomsky grammars. Type-0 grammars, the most powerful Chomsky grammars and the generators of the RELs, are not capable of handling set Q above; more specifically, Q is not a r.e. set if taken in the framework of these grammars, so that we have the inclusion relation **REL** $\subset$ **WL** (Grune, 1993): Given 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 $t_1^n...t_k^n$, every terminal character t_j in $t_1...t_k$, $1 \leq j \leq k \in \mathbb{N}$, is in the range of a total function mapping elements of $\mathbb{N}$ to names of terminal symbols that are given in a representation table. Be it as it may – i.e. whether we have **REL** $\subseteq$ **WL** or **REL** $\subset$ **WL** –, this language inclusion relation indicates that the WLs are largely undecidable. So, the question is how to guarantee decidability while preserving the high expressiveness and generative power of WGs. Luckily, we have at hand a few syntactic restrictions to WGs that give us this desideratum in the sense that restricted-WG generated strings can be effectively and unambiguously parsed. In particular, eliminating ambiguity and ϵ-productions in WGs has proven to generally give us decidability for WLs, but there are other methods that I also discuss below.

5.1 Staying within the Types 3 through 1 of the Chomsky Hierarchy

As seen above, sets that are r.e. but not recursive are always infinite (cf. Fact 33); however, infinity alone does not suffice for undecidability. One of the advantages of using grammars of such high complexity as the WGs is that of having infinite sets of production rules that are recursive. The easiest way to be assured of decidability for WLs is by restricting them syntactically to Chomsky grammars of Types 3 through 1: As is well known, the membership problem for the languages generated by these grammars is decidable, if with increasing computational complexity. Decidability can be secured by means of some syntactic restrictions that have mainly to do with the occurrence of specific notions in the rules of a WG.

The Chomsky class **CSL** is particularly of interest for its expressive capabilities – which, however, in the strict framework of the Chomsky hierarchy dictate their inefficient parsing. Recall 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 other words, a CSG is typically an ϵ-free grammar, with ϵ being exceptionally allowed in a rule $S \to \epsilon$ if S does not occur in the RHS of any other rule in the production set of the grammar – which obviously either gives us the uninteresting CSL $L = \{\epsilon\}$ or a redundant rule. In the terminology of WGs, this property can be defined as below for the hypernotions (exclusively) in a hyper-rule. This requires some abbreviations that will be recurrent in other places below. Recall from Remark 8 the basic form of a hyper-rule.

Remark 42. Given a hyper-rule

$$(r_V j) \qquad \langle X_{j,0}\rangle \to \overset{\circ}{X}_{j,1}; \overset{\circ}{X}_{j,2}; ...; \overset{\circ}{X}_{j,m}$$

for each i-th meta-alternative in $\left\{\overset{\circ}{X}_{j,i}\right\}_{i=1}^{m}$ we have

$$(r_V j_i) \qquad \langle X_{j,0}\rangle \to \gamma \langle X_{j,i_1}\rangle \delta ... \gamma \langle X_{j,i_k}\rangle \delta$$

if the i-th meta-alternative of $r_V j_i$ has a sequence of k non-empty hypernotions, $k \geq 1$ (if $k = 0$, we have a hyper-rule of the form $\langle X\rangle \to \epsilon$). To simplify, I omit the subscripts for rule j and write simply $\langle X_0\rangle \to \langle X_1\rangle ... \langle X_k\rangle$, given that γ, δ are not factored in in the following computations.

Definition 43. A WG $\mathcal{G}$ is said to be *lossless* if for each hyper-rule $\langle X_0\rangle_{\neq\langle s\rangle} \to \langle X_1\rangle ... \langle X_k\rangle$ we have

$$\ell_V (\langle X_0\rangle) \leq \ell_V \left(\sum_{i=1}^{k} \langle X_i\rangle \right)$$

where ℓ_V denotes the length of constituents with respect to the set V (i.e. ignoring metavariables)[19] and for each $W \in \langle X_0\rangle$ we have

$$|\langle X_0\rangle|_W \leq \left| (\langle X_i\rangle)_{i=1}^{k} \right|_W$$

where $|\langle X_i\rangle|_W$ denotes the number of occurrences of W in a hypernotion $\langle X_i\rangle \in N^+$.

For the computation above, it should be obvious why the start hypernotion $\langle s\rangle$ is removed; in any case, we consider a hyper-rule with $\langle s\rangle$ on the LHS to be lossless as long as the RHS is not empty or we have $\overset{\circ}{X} \in N^+$. In effect, the solvability of the decision problem for lossless WGs lies on the fact that sentential form ϑ_i in a derivation $\mathscr{D}_w$ is of at least the same length as ϑ_{i-1}.

Example 44. It can be easily verified that *Alpha2Imp* (cf. Example 23) is a lossless WG. The WG *NoRep* of Example 24 is not lossless, despite its hyper-rules $r_V 1$ through $r_V 6$ being lossless: In effect, no WG with a hyper-rule of the form $\langle X\rangle \to \epsilon$ is lossless.

Theorem 45. *(Baker, 1970) The family of languages generated by lossless WGs is precisely the family of CSLs.*

Proof. (Idea) 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. □

Greibach (1974) discusses a family of WGs – the normal regular-based WGs, denoted by **NRB** – that are also included in **CSL**.

Definition 46. A WG $\mathcal{G}$ is said to be *normal* if a hyper-rule $r_V i$ satisfies the conditions

$$\langle X_0\rangle_{r_V i} = W \in M \text{ or } v \in V$$

and

$$\mathrm{RHS}\,(r_V i) = Y \in N^+.$$

[19] Alternatively, one can consider the total number of occurring protovariables.

1. A normal WG $\mathcal{G}$ is said to be *factored* if (i) for $W \in M$ we have it that $L_W \neq \emptyset$, (ii) for all $W_i, W_j \in M$ it is the case that either $L_{W_i} = L_{W_j}$ or $L_{W_i} \cap L_{W_j} = \emptyset$, and (iii) if $\langle X \rangle \in N^+$ occurs in any hyper-rule, then there is a $W \in M$ such that $L_{\langle X \rangle} \subseteq L_W$ where $L_{\langle X \rangle} = \left\{ h\left(W_i\right) \mid \{W_i\}_{i=1}^{k} \subseteq M \right\}$.

Consider now a grammar G to be regular if every production is of the form $X \to uY$ or $X \to v$, where $X, Y \in V$, $u \in T^*$, and $v \in T^+$.

Definition 47. A WG $\mathcal{G}$ whose metagrammar $\mathcal{G}^1$ simulates a regular grammar G, denoted by $\mathcal{G}_R^1$, is called a *regular-based WG*.

In order to carry out this simulation we may have the $\mathcal{G}_R^1$ combinations shown in Table 4.

Table 4: $\mathcal{G}_R^1$-rule forms for a regular-based WG.

LHS		**RHS**
W_i	::	xW_j
W	::	v

Definition 48. A WG $\mathcal{G}$ that is both normal and regular-based is a *normal regular-based WG* (abbr.: NRB).

The following result follows obviously:

Proposition 49. CFG $\subset$ NRB.

Proof. Left as an exercise. □

Theorem 50. *Given a NRB $\mathcal{G}$, we can construct a lossless factored NRB $\mathcal{G}'$ such that $L\left(\mathcal{G}^S\right) = L\left(\mathcal{G}'^S\right)$.*

Corollary 51. *If $\mathcal{G}$ is a NRB, then $L\left(\mathcal{G}^S\right)$ is context-sensitive.*

Corollary 52. *If $\mathcal{G}$ is a NRB, then membership in $L\left(\mathcal{G}^S\right)$ is decidable.*

Recall that the class **CSL** is decidable only for membership. The reader should note that the proper inclusion in Proposition 49 provides the languages in the class **NRB** with the additional decidability properties for the class **CFG**, to wit, membership, finiteness, and emptiness. Thus, and by Theorem 53 below, the class **NRB** gives us CSLs that are also decidable for finiteness and emptiness by an extension of corollary 52. For the subject of the present article, the following is the crucial result:

Theorem 53. NRB $\subset$ CSL.

Proof. A CSL $L(G)$ for a CSG G can be recognized (in linear space and time) by a non-deterministic Turing machine $\mathcal{M}$ that needs at most $|w|$ tape squares to accept w. $\mathcal{M}$ equally recognizes a NRB language $L\left(\mathcal{G}^S\right)$ for a NRB grammar $\mathcal{G}$. □

Example 54. Grammar *Alpha2Imp* of Example 23 is a NRB.

5.2 (Strict) Boundedness and Recursiveness

P. Deussen and K. Melhorn (Deussen, 1975; Deussen & Melhorn, 1977) devised some restrictions that guarantee that the generated sets of words constitute decidable languages. Remark 42 holds in the following discussion.

Definition 55. We say that a hyper-rule $\langle X_0\rangle_{\neq\langle s\rangle} \to \langle X_1\rangle \ldots \langle X_k\rangle$ is:

1. *left-bounded* (*right-bounded*) iff for any $W \in M$ if $|\langle X_0\rangle|_W \geq 1$ (if $\left|(\langle X_i\rangle)_{i=1}^k\right|_W \geq 1$), then $\left|(\langle X_i\rangle)_{i=1}^k\right|_W \geq 1$ ($|\langle X_0\rangle|_W \geq 1$, respectively);
2. *strict left-bounded* (*strict right-bounded*) if for all $W \in M$ we have

$$|\langle X_0\rangle|_W \leq \left|(\langle X_i\rangle)_{i=1}^k\right|_W$$

$$\left(|\langle X_0\rangle|_W \geq \left|(\langle X_i\rangle)_{i=1}^k\right|_W, \text{respectively}\right)$$

and moreover

$$\ell_V(\langle X_0\rangle) \leq \ell_V\left(\sum_{i=1}^k \langle X_i\rangle\right)$$

$$\left(\ell_V(\langle X_0\rangle) \geq \ell_V\left(\sum_{i=1}^k \langle X_i\rangle\right), \text{respectively}\right).$$

The above properties are abbreviated as *lb*, *rb*, *slb*, and *srb*, respectively.[20]

Definition 56. A WG $\mathcal{G}$ is said to be of type $\mathcal{L}$ ($\mathcal{R}$) iff their hyper-rules are (i) ϵ-free and (ii) *lb* (*rb*, respectively), and (iii) all their hyper-rules of the form $\langle X_0\rangle \to \langle X_1\rangle$ are *slb* (*srb*, respectively).[21]

WLs of type $\mathcal{L}$ ($\mathcal{R}$) are denoted by $L_{\mathcal{L}}$($L_{\mathcal{R}}$, respectively).

Theorem 57. *The WLs of types $L_{\mathcal{L}}$ and $L_{\mathcal{R}}$ are decidable.*

Proof. (Idea) The cited authors show that these WLs are acceptable by an offline Turing machine in exponential time. □

5.3 Handling ϵ-rules

Although they are generally very useful and in some cases even indispensable, ϵ-productions are a cause for ambiguity in CFGs; for WGs, ϵ-rules are seen by some authors (e.g., Visser, 1997) as one of the causes of WL undecidability. In particular, they may impact negatively on the construction of parses.

Definition 58. A parse tree $\mathcal{T}_w$ is said to be an *ϵ-tree*, denoted by $\mathcal{T}_{w,\epsilon}$, if it has empty leaves or interior nodes with no descendants.

[20] Note that a strict left-bounded WG just is a lossless WG (see Def. 43 above).
[21] See below for the elimination of ϵ-rules.

While ϵ-trees have nothing wrong in themselves, they typically cause trouble in compiling CFGs, and, as said, may dictate the undecidability of the parsing problem for WGs. In any case, a parse tree $\mathcal{T}_w$ is an ϵ-tree iff there is at least one ϵ-hyper-rule $r_V j \in R_V$.

Definition 59. A hyper-rule $r_V j$ is said to be an *ϵ-hyper-rule* if it has the form $\langle X_{j,0} \rangle \to \epsilon$ for one of its $1 \leq i \leq m$ hyper-alternatives.

Example 60. The parse tree $\mathcal{T}_{fun}$ for string $fun \in L\left(\widehat{NoRep}\right)$ (cf. Example 24) is actually a tree $\mathcal{T}_{fun,\epsilon}$. This ϵ-tree property is caused by the application of ϵ-hyper-rule $r_V 7$, which is required by the predicates in $\mathrm{LHS}\,(r_V 7)$ for checking that a word $w = t_1 t_2 ... t_k$ does indeed satisfy the property that $t_i \neq t_j$ whenever $i \neq j$, $1 \leq i, j \leq k$.

Predicates do indeed require ϵ to be in the RHS of some hyper-rule of the WG in which they feature. However, often ϵ-hyper-alternatives can be eliminated while preserving the language generated by the WG.

The proof of the following theorem provides a simple algorithm to remove ϵ-hyper-rules:

Theorem 61. *For every WG $\mathcal{G}$ there is a WG $\mathcal{G}'$ such that $L(\mathcal{G}) = L(\mathcal{G}')$ and $\mathcal{G}'$ is ϵ-free.*

Proof. For every ϵ-hyper-rule $\langle X_{j,0} \rangle \to \epsilon$ in $\mathcal{G}^2$ select a hyper-rule $r_V l$, $j \neq l$, with a hypernotion $\langle X_{l,i} \rangle_{i=1,...,k}$ containing or equal to $\langle X_{j,0} \rangle$. Then, (1) add to $r_V l$ the hyper-alternative $\langle X_{l,k+1} \rangle - \langle X_{j,0} \rangle$ and (2) remove $\langle X_{j,0} \rangle \to \epsilon$ from $\mathcal{G}^2$. □

This simple algorithm is guaranteed to work only for hyper-rules that are context-free; beyond that, it is ineffectual.

Example 62. In the WG *AlphaK* of Example 26 it is easy to identify $r_V 4$ as an ϵ-hyper-rule. We have $\langle X_0 \rangle_{r_V 4} = \mathsf{K}$, it being the case that the metanotion K occurs on $\mathrm{RHS}\,(r_V 2)$ and $\mathrm{RHS}\,(r_V 3)$. By applying the simple algorithm in the proof of Theorem 61 we obtain the following set of hyper-rules with the added hyper-alternatives (underlined for easy identification):

$(r_V 1)$	s	:	N i tail.
$(r_V 2)$	N K tail	:	N K, N K i tail; <u>N, N tail</u>; EMPTY.
$(r_V 3)$	N n K	:	K symbol, N K; <u>N</u>.

But the occurrence of yet another ϵ-hyper-rule, to wit, $r_V 2$, makes the first handled ϵ-hyper-rule rather superfluous. Moreover, the hyper-alternatives N for r_V2-3 and N tail for $r_V 2$ make no sense, as N denotes merely a number of occurrences (of no letter). This is remedied by an ϵ-metarule (see below) such that we have $URR(\mathsf{N}, \epsilon)$. Clearly, the hyper-rules of this grammar are not context-free, reason why the ϵ-elimination given in the algorithmic proof above fails – even though the grammar could be saved by this last URR, but this is a solution that increases to a great extent the complexity of word parsing for this grammar.

However, identification of empty hyper-alternatives is not always so obvious, reason why following Wegner (1980) we can speak of hidden ϵ-hyper-rules.

Definition 63. Given a hyper-rule $\langle X_0\rangle \to \langle X_1\rangle \ldots \langle X_k\rangle$, for $1 \leq i \leq k$ we say that $\langle X_i\rangle$ has a *hidden empty notion* if there is a $URR^{rv}_{r_M}(W, \epsilon)$ for $W \in \langle X_i\rangle$.

Example 64. The grammar *AlphaK* of Example 26 has a hidden empty notion such that we have $URR^{1,2,3}_1(\mathsf{N}, \epsilon)$.

The main problem with hidden ϵ-notions is that we can get two leftmost derivations for a left parse, as the following example from Wegner (1980) shows:[22]

Example 65. The WL $L\left(\mathcal{G}^S_{AB}\right) = \{a, ab\}$ can be generated by the following WG $\mathcal{G}^D_{AB}$:

(r_M1)	BETY	::	b; EMPTY.
(r_M2)	EMPTY	::	.
(r_V1)	s	:	a BETY.
(r_V2)	a	:	a symbol, BETY.
(r_V3)	b	:	b symbol.

Consider the strict language $L_{\mathsf{ab}} = \{ab\}$. Recall from footnote 6 that a $URR(W_i, v)$ holds for hyper-rules taken individually, and not for the whole set R_V. In this particular case, the parse $\mathscr{P}^L_{ab}$ gives rise to the two following leftmost derivations depending on whether one chooses $URR^1_1(\mathsf{BETY}, \mathsf{b})$ or $URR^1_1(\mathsf{BETY}, \epsilon)$, but one selects $URR^2_1(\mathsf{BETY}, \mathsf{b})$:

$$\left(\mathscr{D}^L_{ab}\right)_1 = \mathsf{s} \underset{r_V1}{\Longrightarrow} \mathsf{a\,EMPTY} \underset{r_V2}{\Longrightarrow} a, \mathsf{b} \underset{r_V3}{\Longrightarrow} ab$$

$$\left(\mathscr{D}^L_{ab}\right)_2 = \mathsf{s} \underset{r_V1}{\Longrightarrow} \mathsf{a\,b} \underset{r_V2}{\Longrightarrow} a, \mathsf{b} \underset{r_V3}{\Longrightarrow} ab$$

The proof of the following theorem gives us an algorithm to eliminate hidden empty notions.

Theorem 66. *(Wegner, 1980) For every WG $\mathcal{G}$ there is a WG $\mathcal{G}'$ such that $L(\mathcal{G}) = L(\mathcal{G}')$ and $\mathcal{G}'$ is free of hidden empty notions.*

Proof. Let $\mathcal{G} = (M, V, R_M, N, T, S, R_V)$ and $\mathcal{G}' = \left(M', V', R'_M, N', T', S', R'_V\right)$. Set $V' = V \cup \{\dagger\}$ where $V \cap \{\dagger\} = \emptyset$. Let $M = M'$, $R_M = R'_M$, $T = T'$, and make

$$N' = \{\langle \dagger X\rangle \,|\, \langle X\rangle \in N\}.$$

In order to obtain the equality $L(\mathcal{G}) = L(\mathcal{G}')$ stated in the theorem proceed in the following way: (1) Make R'_V the set of hyper-rules obtained from R_V by replacing each hypernotion $\langle X\rangle \in N$ by the corresponding $\langle \dagger X\rangle \in N'$; (2) add to R'_V a hyper-rule $\langle \dagger\rangle \to \epsilon$; and (3) set $S' = \{\langle \dagger s\rangle\}$. □

[22] I slightly changed Wegner's example.

5.4 Eliminating Ambiguity

5.4.1 Some General Remarks on WG Ambiguity

Just as in the case of CFLs, we shall consider a WL to be ambiguous if there are more than one leftmost derivation for a given string. Formally:

Definition 67. A WG $\mathcal{G}$ is said to be *ambiguous* if there is a string $w \in L(\mathcal{G})$ with two or more derivations $\mathscr{D}_w^L$ in $\mathcal{G}$. Otherwise, $\mathcal{G}$ is *unambiguous*.

The causes for ambiguity in WGs are many and possibly not all identified, but a few rules of thumb can be given. To begin with, if both X and Y are metanotions, then XY (or YX) should not be a third different notion, so as not to be confused with the concatenation of X and Y. For instance, if NOTION and EMPTY are two metanotions defined as usually, i.e. as a token (letter, digit, etc.) and as the empty syntactic mark ϵ, then the third metanotion NOTIONEMPTY will originate ambiguity, regardless of how it is defined in the WG.

Besides concatenation, prefixes and suffixes can be a source of ambiguity. As an affix grammar, a WG uses both prefixation and suffixation.

Example 68. Consider a WG $\mathcal{G}$ with the metarules "NOTION :: a; b." and "EMPTY :: .". In the following two hyper-rules the paranotion option is used both as a prefix (in the first hyper-rule) and as a suffix (in the second hyper-rule):

- option NOTION : NOTION; EMPTY.
- NOTION option : NOTION; EMPTY.

Given a $URR(\mathsf{NOTION}, \mathsf{x})$ for $\mathsf{x} = \mathsf{a}, \mathsf{b}$, from the first hyper-rule, we can derive the production rules

$$R_{\mathsf{S}} = \left\{ \begin{array}{l} \mathsf{option\,a} : \mathsf{a};\ . \\ \mathsf{option\,b} : \mathsf{b};\ . \end{array} \right\}.$$

Let us now change prefixation into suffixation: From the second hyper-rule, we can derive the production rules

$$R_{\mathsf{S}} = \left\{ \begin{array}{l} \mathsf{a\,option} : \mathsf{a};\ . \\ \mathsf{b\,option} : \mathsf{b};\ . \end{array} \right\}.$$

With the avoidance of ambiguity in mind intermixing prefixes and suffixes should not be allowed, as shown in the following example.

Example 69. Consider the following two hyper-rules of a WG:

- option NOTION : NOTION; EMPTY.
- NOTION pack : left parenthesis symbol, NOTION, right parenthesis symbol.

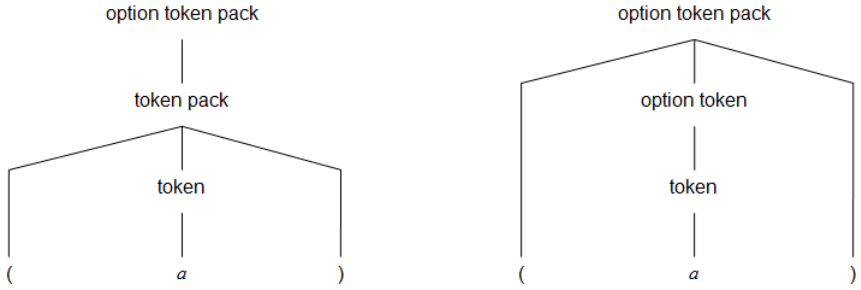
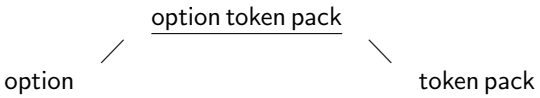
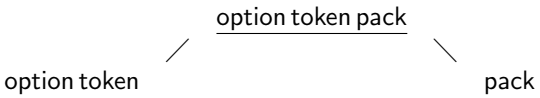


Figure 8: Two parse trees for the string "(a)".

By applying a $URR\,(\mathsf{NOTION}, \mathsf{token})$ simultaneously to the LHSs of both hyper-rules we obtain the protonotion option token pack. Then, there are two ways to parse the protonotion option token pack, to wit, as



or



In the first case, we have an optional set of balanced parentheses with a token (a letter) inside, and in the second case we have an optional token inside balanced parentheses. Figure 8 shows the two different parse trees for the string "(a)". Clearly, this WG is ambiguous.

In the case of the CFGs, one way to avoid ambiguity is to establish leftmost derivation (vs. rightmost derivation) as the default parsing. In the same way, WGs must opt for prefixation or suffixation, with the latter being preferred in the literature (e.g., Cleaveland & Uzgalis, 1977; van Weingaarden et al., 1976).

However, these and the above ambiguity restrictions do not suffice to guarantee decidability with respect to a WL.

5.4.2 Wegner's Algorithm

Wegner (1980) provides a means to secure decidability via parsing by actually setting out from the property of (strict) boundedness discussed in Section 5.2 above. Wegner's core concept is that of a locally unambiguous WG defined as follows:

Definition 70. A WG $\mathcal{G}$ is said to be *locally unambiguous* if, given the pair $(w, \mathcal{G})$:

1. For every $\mathscr{D}^L_w$ there is exactly one $\mathscr{P}^L_w$ and given $\mathscr{D}^L_w$ we can effectively find $\mathscr{P}^L_w$.
2. For every $\mathscr{P}^L_w$ there is exactly one $\mathscr{D}^L_w$ and given $\mathscr{P}^L_w$ we can effectively find $\mathscr{D}^L_w$.

Recall that the RHS of an ϵ-free hyper-rule $r_V j$ is constituted by $m \geq 1$ hyper-alternatives $\left\{\mathring{X}_{j,i}\right\}_{i=1}^{m} \subseteq \left((N \cup T)^{+} \cup \{,\}\right)$. Local unambiguity turns the inherent non-determinism in any generative rewriting system into a choice between the hyper-alternatives in $\left\{\mathring{X}_{j,i}\right\}_{i=1}^{m} = \mathrm{RHS}\left(r_V j\right)$ (rather than multiple partitions of strict notions or choices resulting from unbound metanotions).

The conditions that suffice for local unambiguity in a WG $\mathcal{G}$ free from hidden ϵ-notions (see Section 5.3 above) are (1) boundedness, (2) unique assignability, and (3) disjointness. Boundedness was discussed in Section 5.2 above and I now approach the two remaining properties as elaborated on – but with some changes in notation – in Wegner (1980). A somewhat long series of definitions and statements will be required, in order to grasp Wegner's Algorithm, which provides a decision procedure for the DP of locally unambiguous WGs.

Definition 71. Let $H \in N^{+}$ contain the metanotions $W_1, W_2, ..., W_n$. A *hypernotion system* associated to H is a 4-tuple $\mathcal{H} = (M, V, R_M, H)$ where H is the *hypernotion axiom*. The *language of* $\mathcal{H}$ is defined as:

$$L\left(\mathcal{H}\right) = \left\{h\left(H\right) | h\left(v\right) = v \text{ for } v \in V, h\left(W_i\right)_{i=1}^{n} \in L\left(\mathcal{G}^1\right)\right\}$$

$\mathcal{H} = (M, V, R_M, \langle X_0 \rangle)$ is a *LHS hypernotion system* and $\mathcal{H} = (M, V, R_M, \mathfrak{H})$, $\mathfrak{H} = H_1 H_2 ... H_k \in (M \cup P \cup \{,\})^{*}$, is a *RHS hypernotion system*.

Definition 72. Let $\mathcal{H}$ be a hypernotion system where $H = X_1 X_2 ... X_n$ for $X_i \in N^{+}$, $1 \leq i \leq n$. $\mathcal{H}$ is said to be *uniquely assignable* (abbr.: *ua*) if for all $w \in L\left(\mathcal{H}\right)$ there is exactly one partition $(w_1, w_2, ..., w_n)$ such that we have the word factorization $w_1 w_2 ... w_n = w$ and $h\left(X_i\right) = w_i$.

Example 73. Consider the following set of (abstract) metarules:

$$R_M = \left\{ \begin{array}{c} \mathsf{A} \rightarrow \mathsf{aA}; \mathsf{a}. \\ \mathsf{A}_1 \rightarrow \mathsf{A}. \\ \mathsf{A}_2 \rightarrow \mathsf{A}. \end{array} \right\}$$

We have, for instance, the following hypernotion systems:

$$\mathcal{H}_1 = \left(\{\mathsf{A}\}, \{\mathsf{a}\}, R_M, \mathsf{AA}\right)$$

$$\mathcal{H}_2 = \left(\{\mathsf{A}\}, \{\mathsf{a}\}, R_M, \mathsf{A_1 A_2}\right)$$

$$\mathcal{H}_3 = \left(\{\mathsf{A}\}, \{\mathsf{a}\}, R_M, \mathsf{A_1 A_2 A_1}\right)$$

$$\mathcal{H}_4 = \left(\{\mathsf{A}\}, \{\mathsf{a}, \mathsf{b}\}, R_M, \mathsf{A_1 b A_2 A_1}\right)$$

We have it that $\mathcal{H}_1$ and $\mathcal{H}_4$ are *ua*; $\mathcal{H}_2$ and $\mathcal{H}_3$ are not *ua*.

As a preliminary result, hypernotion systems have already some interesting decidability properties:

Proposition 74. *(Wegner, 1980) Let $L\left(\mathcal{H}\right)$ be the language of a hypernotion system $\mathcal{H}$. Then, the problems $w \overset{?}{\in} L\left(\mathcal{H}\right)$ and $L\left(\mathcal{H}\right) \overset{?}{=} \emptyset$ are decidable. The problems $L\left(\mathcal{H}_1\right) \cap L\left(\mathcal{H}_2\right) \overset{?}{=} \emptyset$ and $L\left(\mathcal{H}_1\right) \overset{?}{=} L\left(\mathcal{H}_2\right)$ are in general undecidable for hypernotion systems with context-free metarules.*

Further decidability results in Wegner (1980) are as follows:

Proposition 75. *For an arbitrary hypernotion system* $\mathcal{H} = (M, V, R_M, H)$ *it is (1) undecidable whether* $\mathcal{H}$ *is* ua *if* R_M *is context-free, and (2) decidable whether* $\mathcal{H}$ *is* ua *if* R_M *is regular and no metanotion occurs more than once in* H.

Propositions 74 and 75 are left unproven in Wegner (1980). I build up now on the property of being uniquely assignable for a WG:

Definition 76. A WG $\mathcal{G}$ is said to be *LHS uniquely assignable* (abbr.: LHSua) if in every hyper-rule $\langle X_0 \rangle \to \mathring{X}$ the LHS hypernotion system $\mathcal{H} = (M, V, R_M, \langle X_0 \rangle)$ is *ua*. $\mathcal{G}$ is said to be *RHS uniquely assignable* (abbr.: RHSua) if in every hyper-rule $\langle X_0 \rangle \to \left(\mathring{X} = \mathfrak{H}\right)$, the RHS hypernotion system $\mathcal{H} = (M, V, R_M, \mathfrak{H})$ is *ua*.

I now move forward to the concept of disjointness.

Definition 77. For some WG $\mathcal{G}$, let $r_V i, r_V j \in R_V, i \neq j$ and $\mathrm{LHS}(r_V i; r_V j) \neq \mathsf{s}$. Then, $r_V i$ and $r_V j$ are said to be *disjoint* if $R_S(r_V i) \cap R_S(r_V j) = \emptyset$. $\mathcal{G}$ is said to be disjoint if for all pairs of hyper-rules $(r_V i, r_V j)$ we have it that $R_S(r_V i) \cap R_S(r_V j) = \emptyset$.

Intuitively, this means that in a disjoint WG there is at most one hyper-rule $r_V k$ for any given strict production rule $r_S l$ such that $r_S l \in R_S(r_V k)$ for some $k \geq 2$ if $\mathrm{LHS}(r_V 1) = \mathsf{s}$. The following further specifications assure us that no two strict production rules have the same LHS unless they are derived from two hyper-rules with the same LHS but with different RHSs, and – a stronger restriction – the RHSs of strict production rules derived from different hyper-rules are different.

Definition 78. A WG $\mathcal{G}$ is *left-disjoint* if for any pair of hyper-rules $(r_V i, r_V j)$, $i \neq j$ and $\mathrm{LHS}(r_V i; r_V j) \neq \mathsf{s}$, we have:

1. If $(X_0)_{r_V i} \neq (X_0)_{r_V j}$, $(\alpha \to \beta) \in R_S(r_V i)$ and $(\gamma \to \delta) \in R_S(r_V j)$, then $\alpha \neq \gamma$;
2. If $(X_0)_{r_V i} = (X_0)_{r_V j}$, $(\alpha \to \beta) \in R_S(r_V i)$ and $(\gamma \to \delta) \in R_S(r_V j)$, then $\beta \neq \delta$.

$\mathcal{G}$ is *right-disjoint* if for all hyper-rule pairs $(r_V i, r_V j)$ with $i \neq j$, and for all $(\alpha \to \beta) \in R_S(r_V i)$, $(\gamma \to \delta) \in R_S(r_V j)$ we have $\beta \neq \delta$.

And the important results follow:

Theorem 79. *(Wegner, 1980) If a WG* $\mathcal{G}$ *is disjoint, free of hidden empty notions, and either (a) LHSua and* lb*, or (b) RHSua and* rb*, then* $\mathcal{G}$ *is locally unambiguous.*

Proof. Cf. Wegner (1980). □

Theorem 80. *(Wegner, 1980) For every WG* $\mathcal{G}$*, we can construct a locally unambiguous WG* $\mathcal{G}'$ *such that* $L(\mathcal{G}) = L(\mathcal{G}')$.

Proof. (Idea; cf. Wegner, 1980) For a given WG $\mathcal{G}$ the corresponding non-deterministic Turing machine $\mathcal{M}$ is constructed (see Baker, 1972); in turn, the equivalent Chomsky grammar G is constructed for $\mathcal{M}$. Then, G is made to be left-disjoint by adding a unique label to each hypernotion and each hypernotion is made to be *ua* by inserting some marker (e.g., \$); two hyper-rules are added to move the marker to the left and to the right. Denote this modified grammar by $\mathcal{G}'$. $\mathcal{G}'$ is locally unambiguous and it can easily be shown that $L(\mathcal{G}) = L(\mathcal{G}')$. □

Neither of these two theorems, however, is the main result in Wegner (1980); the following theorem is:

Theorem 81. *If $\mathcal{G}$ is a locally unambiguous WG and $\mathcal{S}$ is the proper and unambiguous skeleton grammar of $\mathcal{G}$, then $\mathcal{G}$ is unambiguous.*

Proof. The proof will follow easily from the contents below. □

The property of local unambiguity was discussed above; I approach now the skeleton grammar of a WG $\mathcal{G}$. What is aimed at with the complex procedure to be explained below is a canonical CFG G that models, or simulates, under certain restrictions, the derivations in a WG $\mathcal{G}$, so that we are assured of the solvability of DP. The aimed at CFG is constructed from cross-references:

Definition 82. Given a pair $(r, \mathcal{G})$ where $r = r_V l$ is a hyper-rule in $\mathcal{G}^2$ with the usual abstract form $\langle X_0 \rangle \to \left(\mathring{X} = \mathfrak{H} \right)$, the *cross-reference* of r is the k-tuple $(x_1, x_2, ..., x_k) = \mathfrak{x}$ such that:

1. if $H_i \in T$ for $1 \leq i \leq k$, then $x_i = H_i$; otherwise,
   (a) $x_i = \left\langle X_0^{'} \right\rangle$ for some LHS (r) such that $r \in R_V$, or
   (b) $x_i = \epsilon$ if $\epsilon \in L(M, V, R_M, H_i)$;
2. $L(M, V, R_M, \mathfrak{H}) \cap L(M, V, R_M, \mathfrak{x}') \neq \emptyset$, where $x_i^{'}$ in $\mathfrak{x}'$ is obtained from x_i in $\mathfrak{x}$ by renaming metanotions in x_i such that they are distinct from those in x_j, $i \neq j$, $1 \leq i, j \leq k$.

Intuitively, we say that x_i refers to $\left\langle X_0^{'} \right\rangle$ or H_i in $\mathfrak{H}$, $1 \leq i, j \leq k$.

Definition 83. The *skeleton grammar* $\mathcal{S}$ associated to $\mathcal{G}$ is the CFG $G_{\mathcal{G}} = (V, T', S', R') = \mathcal{S}$ where:

- $V = \left\{ \langle X_0 \rangle \mid \left(\langle X_0 \rangle \to \mathring{X} \right) \in R_V \right\} \subseteq N$
- $T' = T$
- $S' = S$
- $R' = \bigcup \tilde{r} = \bigcup R_{\mathcal{S}}(r)$ where

$$R_{\mathcal{S}}(r) = \left\{ \underbrace{\langle X_0 \rangle \to \mathfrak{x}}_{\tilde{r}} | r = (\langle X_0 \rangle \to \mathfrak{H}) \in R_V \right\}$$

where $H_i \in (N \cup T)$ and $\mathfrak{x} = (x_i)_{i=1}^{k}$ is a cross-reference.

Definition 84. Let $\mathcal{S}$ be the skeleton grammar of a WG $\mathcal{G}$. Then, we say that $\mathcal{S}$ is *proper* iff for all $r, r' \in R_V$ such that $r \neq r'$ we have $R_{\mathcal{S}}(r) \cap R_{\mathcal{S}}(r') = \emptyset$.

Lemma 85. *$\mathcal{S}$ is proper iff all rules $r = (\langle X_0 \rangle \to \mathfrak{H})$ and $r' = \left(\left\langle X_0^{'} \right\rangle \to \mathfrak{H}'\right)$ with $\langle X_0 \rangle = \left\langle X_0^{'} \right\rangle$ in R_V of a WG $\mathcal{G}$ have pairwise distinct cross-references.*

Proof. (a) Obviously, we have $R_{\mathcal{S}}(r) \cap R_{\mathcal{S}}(r') = \emptyset$ if $\langle X_0 \rangle \neq \left\langle X_0^{'} \right\rangle$ and (b) only identical LHS combinations give identical RHSs in $\mathcal{S}$. □

The relevance of working with a proper skeleton grammar is that, given a parse $\mathscr{P}_w^{\mathcal{S},L}$ in $\mathcal{S}$, abbreviated as $\tilde{\mathscr{P}}_w^L$, we can uniquely determine a sequence of hyper-rules in $\mathcal{G}$ and from $\mathscr{P}_w^L$ we can uniquely determine a derivation $\mathscr{D}_w^L$ if $\mathcal{G}$ is locally unambiguous. The following definition establishes the link between $\tilde{\mathscr{P}}_w^L$ and $\mathscr{P}_w^L$:

Definition 86. Given a pair $(\mathcal{G}, \mathcal{S})$ of a WG $\mathcal{G}$ and its associated skeleton grammar $\mathcal{S}$, let $\mathscr{P}_w^L = r_{i_1}, r_{i_2}, ..., r_{i_n}$ be a left parse of $w \in L(\mathcal{G})$ and $\tilde{\mathscr{P}}_w^L = \tilde{r}_{i_1}, \tilde{r}_{i_2}, ..., \tilde{r}_{i_n}$ be a left parse of $w \in L(\mathcal{S})$. Then, we speak of $\tilde{\mathscr{P}}_w^L$ as *corresponding to* $\mathscr{P}_w^L$ if we have $\tilde{r}_{i_j} \in R_{\mathcal{S}}\left(r_{i_j}\right)$ for all $1 \leq j \leq n$.

Theorem 87. *For any pair $(\mathcal{G}, \mathcal{S})$ of a WG $\mathcal{G}$ and its associated skeleton grammar $\mathcal{S}$, for a parse $\mathscr{P}_w^L$ in $\mathcal{G}$ there is exactly one corresponding parse $\tilde{\mathscr{P}}_w^L$ in $\mathcal{S}$.*

Proof. Obvious from the above. □

With respect to this theorem, it is important to emphasize that $\mathcal{G}$ must be free of hidden empty notions (cf. Section 5.3 above). The following example shows that this theorem might not hold if $\mathcal{G}$ contains empty notions.

Example 88. Consider again the WG $\mathcal{G}_{AB}$ of Example 65. The rules in the set $R_{\mathcal{S}}$ obtained from R_V of $\mathcal{G}_{AB}$ are as follows:

$$R_{\mathcal{S}}(r_V 1) = \left\{ \begin{array}{ll} (\tilde{r}_{1.1}) & \mathsf{s} \to \mathsf{a}. \\ (\tilde{r}_{1.2}) & \mathsf{s} \to \mathsf{a\ b}. \end{array} \right\}$$

$$R_{\mathcal{S}}(r_V 2) = \left\{ \begin{array}{ll} (\tilde{r}_{2.1}) & \mathsf{a} \to a. \\ (\tilde{r}_{2.2}) & \mathsf{a} \to a\ \mathsf{b}. \end{array} \right\}$$

$$R_{\mathcal{S}}(r_V 3) = \left\{ \begin{array}{ll} (\tilde{r}_3) & \mathsf{b} \to b. \end{array} \right\}$$

For $\mathscr{P}_{ab}^L = r_V 1, r_V 2, r_V 3$ there are at least two corresponding left parses, to wit, $\left(\tilde{\mathscr{P}}_{ab}^L\right)_1 = \tilde{r}_{1.1}, \tilde{r}_{2.2}, \tilde{r}_3$ and $\left(\tilde{\mathscr{P}}_{ab}^L\right)_2 = \tilde{r}_{1.2}, \tilde{r}_{2.1}, \tilde{r}_3$.

Corollary 89. *For any pair $(\mathcal{G}, \mathcal{S})$ of a WG $\mathcal{G}$ and its associated skeleton grammar $\mathcal{S}$ we have $L(\mathcal{G}) \subseteq L(\mathcal{S})$.*

Proof. Follows easily from the above. □

Corollary 90. *For any pair $(\mathcal{G}, \mathcal{S})$ such that $\mathcal{S}$ is a proper skeleton grammar associated to the WG $\mathcal{G}$, for two parses $\left(\mathscr{P}_w^L\right)_1$ and $\left(\mathscr{P}_w^L\right)_2$ we have:*

$$\left(\mathscr{P}_w^L\right)_1 = \left(\mathscr{P}_w^L\right)_2 \quad \textit{holds iff} \quad \left(\tilde{\mathscr{P}}_w^L\right)_1 = \left(\tilde{\mathscr{P}}_w^L\right)_2$$

Proof. ($\Rightarrow$) Since $\mathcal{S}$ is proper, every application of a rule $\tilde{r}$ (i.e. every parse step in $\tilde{\mathscr{P}}_w^L$) determines uniquely a hyper-rule application in $\mathscr{P}_w^L$. ($\Leftarrow$) Follows trivially from Theorem 87. □

Algorithm 1 is Wegner's Algorithm or the Skeleton-Grammar Parsing Algorithm.[23]

Algorithm 1 Wegner's Parsing Algorithm

Input: The 3-tuple $(\mathcal{G}, \mathcal{S}, w)$ where $\mathcal{G}$ is a locally unambiguous WG, $\mathcal{S}$ is the proper and unambiguous skeleton grammar associated with it, and $w \in T^*$

Output: "Yes" if $w \in L(\mathcal{G})$; "No" otherwise

Step 1: Obtain $\tilde{\mathscr{P}}_w^L$ by applying any of the parsing algorithms for CFGs to $\mathcal{S}$.[a] If $w \notin L(\mathcal{G})$, then output "No."

Step 2: Obtain $\mathscr{P}_w^L$ starting either (i) from $\langle s \rangle$ (if $\mathcal{G}$ is LHSua and rb) such that we have a derivation

$$\mathscr{D}_w^L = (\langle s \rangle = \alpha_0) \Longrightarrow_l ... \Longrightarrow_l \alpha_i ... \Longrightarrow_l (\alpha_n = w \in T^*)$$

or (ii) from w (if $\mathcal{G}$ is RHSua and lb) such that we have a derivation

$$\mathscr{D}_w^L = (w \in T^* = \alpha_n) \Longrightarrow_l ... \Longrightarrow_l \alpha_i ... \Longrightarrow_l (\alpha_0 = \langle s \rangle)$$

Apply hyper-rule r to those strict notions corresponding to the sentential form $\alpha \in (V \cup T)$ in the derivation step in which the skeleton rule $\tilde{r}_i \in R_{\mathcal{S}}(r)$ was applied. $\mathcal{G}$ is locally unambiguous, so it suffices to give the handle and hyper-rule to (i) reduce α_i to α_{i-1} or, respectively, (ii) to extend α_i to α_{i+1}. If r cannot be applied, then output "No" and stop; otherwise, output "Yes" and stop.

[a] For example, the CYK algorithm. See Augusto (2021).

And the final result, stating that Wegner's Algorithm provides a decision procedure for the DP of WGs, follows:

Theorem 91. *(Wegner, 1980) For arbitrary $w \in T^*$, the Skeleton-Grammar Parsing Algorithm correctly determines in a backtrack-free way whether $w \in L(\mathcal{G})$ or $w \notin L(\mathcal{G})$.*

Proof. Because (by Theorem 81) $\mathcal{G}$ is unambiguous there is at most one derivation $\mathscr{D}_w^L$ for $w \in T^*$. Suppose now that the parse $\mathscr{P}_w^L$ obtained from $\tilde{\mathscr{P}}_w^L$ in $\mathcal{S}$ is not the left parse for w, i.e. there is a parse $\left(\mathscr{P}_w^L\right)'$ such that $\left(\mathscr{P}_w^L\right)' \neq \mathscr{P}_w^L$. By Corollary 90, there is a parse $\left(\tilde{\mathscr{P}}_w^L\right)'$ such that $\left(\mathscr{P}_w^L\right)' \neq \left(\tilde{\mathscr{P}}_w^L\right)'$, but then there are two left parses for w, contradicting the fact that $\mathcal{S}$ is unambiguous. □

Wegner (1980) provides the step-by-step example of the application of Algorithm 1 to a WG.

[23] Recall that a derivation of the form

$$\mathscr{D}_w = (w \in T^* = \alpha_n) \Longrightarrow_l ... \Longrightarrow_l \alpha_i ... \Longrightarrow_l (\alpha_0 = \langle s \rangle)$$

is more correctly called a *reduction* and a parser carrying out a reduction requires *handles* (see Augusto, 2021).

References

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

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

Augusto, L. M. (2024; forthcoming). Little ado about meaning: The intrinsic semantics of van Wijngaarden grammars. *Omega - Journal of Formal Languages*, *2*.

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., 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.

Clayton, R., Rugaber, S., & Wills, L. (1998). On the knowledge required to understand a program. In *Proceedings of the Fifth Working Conference on Reverse Engineering* (pp. 69-78). IEEE.

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.

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

Deussen, P. & Melhorn, K. (1977). Van Wijngaarden grammars and space complexity class EXSPACE. *Acta Informatica*, *8*, 193-199.

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.

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

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

Post, E. L. (1944). Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, *50*, 284-316.

Sebesta, R. W. (2012). *Concepts of programming languages*. 10th ed. Boston, etc.: Pearson.

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

Soloway, E. & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595-609.

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 (ed.), A., Mailloux, B. J., Peck, J. E. L., & Koster, C. H. A. (1969). Report on the algorithmic language ALGOL 68. *Numerical Mathematics*, *14*, 79-218.

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.

Wegner, L. M. (1980). On parsing two-level grammars. *Acta Informatica*, *14*, 175-193.

Williams, M. H. (1985). Structuring two-level grammar specifications. *Computer Journal*, *28*(3), 250-256.

Cite this article as:[a]

Augusto, Luis M. (2023). The van Wijngaarden grammars: A syntax primer with decidable restrictions. *Journal of Knowledge Structures & Systems*, *4*(2), 1-39.

EDITORIAL INFORMATION

Editor-in-chief: Luis M. Augusto
Received: May 2023; **Reviewed**: Awaiting review reports

[a]This article may undergo editing in the next few months. See footnote 2 for an account.