b with (r, b, r'). Let g be the morphism mapping the triples to their middle component:

$$g(r, b, r') = b.$$

Finally, the substitution σ' is defined by

$$\sigma'(s, a, s') = gT_{s,s'}(L'_a) = gT_{s,s'}T_a(L_a).$$

Since $L_a \in \mathcal{L}$ and $gT_{s,s'}T_a$ is a rational transduction, the substitution σ' is of the required form. Moreover, (**) holds true. The core element on both sides is a word $a_1a_2 \dots a_n \in R$. Considering the left side, T goes from s_0 to s_1 when reading a word in L'_{a_1}. The same output is produced on the right side from (s_0, a_1, s_1) by gT_{s_0,s_1}.

We still state the established equation (**) in the following form.

Theorem 3.10. *The full AFL-closure of any language family $\mathcal{L}$ is obtained by first closing $\mathcal{L}$ under rational transductions, and then closing the resulting family under regular operations.* □

The reader is referred to [47] for further details and bibliographical remarks concerning AFL-theory. *Principal AFL's* constitute a widely studied topic we have not discussed here. By definition, a full AFL $\mathcal{L}$ is *full principal* if it is generated by a single language L:

$$\mathcal{L} = Cl_{fAFL}(L).$$

Principal AFL's and cones are defined similarly. No nondenumerable family can be principal. Recursive languages constitute a nontrivial example of an AFL that is not principal.

4. Wijngaarden (two-level) grammars

4.1 Definitions, examples. The generative power

Wijngaarden grammars or *two-level grammars* were introduced in [136] in order to define the syntax and the semantics of the programming language ALGOL 68, see [138]. They are a very natural extension of context-free grammars. In a Wijngaarden, or shortly W-grammar, the idea to extend a context-free grammar is very different from the idea to extend a context-free grammar to a context-sensitive grammar. A W-grammar has nonterminals with an inner structure, i.e., the name of the nonterminal may depend on some variables, called *hypervariables*. Consequently, the rules that contain such hypervariables are referred to as *hyperrules*. The hyperrules are not used directly for derivations. In order to obtain the rules for derivations, one has to substitute consistently (uniformly) all occurrences of hypervariables in hyperrules by values from some fixed sets associated to hypervariables. These sets could

be context-free languages, too. The resulting set of derivation rules could be infinite. A similar phenomenon occurs in mathematical logic, when the propositional logic is extended to the predicate logic by assigning to propositional variables an inner structure that depends on some variables. Note that the variables that occur in predicates and in formulas are consistently substituted by values from a set (the domain of the model).

A further classical method to define a model is to consider all ground terms as the domain, the well-known Herbrand model. The domain of a Herbrand model has a structure very similar to a context-free language.

The theory of W-grammars was developed slowly. At the first glance, the formalism looks complicated enough. We present here a short and straight way to introduce the notion of a W-grammar. A *context-free scheme* is a context-free grammar without the start symbol, i.e., the start symbol is not defined.

Definition 4.1. *A W-grammar is a triple, $G = (G_1, G_2, z)$, where $G_1 = (M, V, R_M)$ and $G_2 = (H, \Sigma, R_H)$ are context-free schemes such that the elements of H are of the form $< h >$, where $h \in (M \cup V)^*$, $(M \cup V \cup \Sigma) \cap \{<, >\} = \emptyset$ and $z \in H$, $z =< v >$, $v \in V^*$.* □

Terminology. G_1 is the *metalevel* of G and the elements of the sets M, V, R_M are the *metavariables*, the *metaterminals* and respectively the *metarules*. G_2 is the *hyperlevel* of G and the elements of the sets H, Σ, R_H are the *hypervariables*, the *terminals* and respectively the *hyperrules*. The *start symbol* of G is z.

In order to define the language generated by a W-grammar, some preliminaries are necessary. Denote by L_X the context-free language generated by the context-free grammar $G_X = (M, V, X, R_M)$, where $X \in M$, i.e., G_X is the context-free grammar that is obtained from the context-free scheme G_1 by defining the start symbol $X \in M$. Consider the sets: $A = M \cup V \cup \Sigma \cup \{<, >, \longrightarrow\}$ and $B = A - M$. Any function $t : A \longrightarrow B^*$, with the property that, for every $X \in M$, $t(X) \in L_X$, and for every $v \in A - M$, $t(v) = v$, can be extended to a unique morphism $t' : A^* \longrightarrow B^*$. All these morphisms t' define the set of *admissible substitutions*, denoted by T.

Definition 4.2. *The set of strict rules is $R_S = \{t(r) \mid t \in T, r \in R_H\}$, and the set of strict hypervariables is $H_S = \{t(x) \mid t \in T, x \in H\}$.*

If r is a hyperrule, $r \in R_H$, then the set of strict rules corresponding to r is

$$R_S(r) = \{t(r) \mid t \in T\}.$$ □

Note that the strict rules are context-free rules, where the nonterminals are strict hypervariables. However H_S can be infinite and, consequently, the set of strict rules, R_S, can be infinite, too. Perhaps this is the most important difference between context-free grammars and W-grammars.

Definition 4.3. *The derivation relation is defined as for context-free grammars, but using the strict rules instead of the classical rules of a context-free grammar, i.e., for any x, y from $(H_S \cup \Sigma)^*$, $x \Longrightarrow^* y$ iff there exist $n \geq 0$ and u_i in $(H_S \cup \Sigma)^*$, $i = 0, \ldots, n$, such that $u_0 = x, u_n = y$ and for each i, $i = 0, \ldots, n-1$, there are α, β and a strict rule $\gamma \longrightarrow \delta$ in R_S, with $u_i = \alpha\gamma\beta$ and $u_{i+1} = \alpha\delta\beta$.*

The leftmost (rightmost) derivation is defined as in the case of context-free grammars. □

Definition 4.4. *The language generated by a W-grammar G, is*

$$L(G) = \{x \mid x \in \Sigma^*, z \Longrightarrow^* x\}.$$ □

In sequel we consider some examples. Assume that, for all of the next three examples, the metalevel of the W-grammar is in each case the same, i.e., $M = \{I, J, K\}$, $V = \{q, s, \#, t, t', t''\}$ and for any $X \in M$, $L_X = q^*$. Moreover, assume that $z =< s >$, $\Sigma = \{a, b, c\}$ and therefore, for each of the following W-grammars, one has to define only the set R_H of hyperrules.

Example 4.1. As a first example, assume that R_H is:

h1. $< s > \longrightarrow < \#q >$
h2. $< I\#J > \longrightarrow < tI > | < J\#IJ >$
h3. $< tqI > \longrightarrow a < tI >$
h4. $< t > \longrightarrow \lambda$

It is easy to observe that the strict rules are: h1, h4,

$$\text{h2}_{i,j}.\ < q^i\#q^j > \longrightarrow < tq^i > | < q^j\#q^{i+j} >,$$

and

$$\text{h3}_k.\ < tq^{k+1} > \longrightarrow a < tq^k >,$$

where $i, j, k \geq 0$.
Note that a strict hypervariable $< tq^n >$ is producing a^n and nothing else, with $n \geq 0$. A derivation starts with z and if the next rules are h1 and the second case of $\text{h2}_{i,j}$ for a number of applications, then one obtains a derivation such as the following:

$$z \Longrightarrow < q^0\#q^1 > \Longrightarrow < q^1\#q^1 > \Longrightarrow < q^1\#q^2 > \Longrightarrow < q^2\#q^3 > \Longrightarrow$$

$$\Longrightarrow < q^3\#q^5 > \Longrightarrow < q^5\#q^8 > \Longrightarrow < q^8\#q^{13} > \Longrightarrow \ldots$$

At any step in the above derivation, one can apply the first case of a strict rule $\text{h2}_{i,j}$ that replaces a strict hypervariable $< q^i\#q^j >$ with a strict hypervariable $< tq^i >$ that derives finally a^i. Therefore, the language generated is:

$$L_1 = \{a^f \mid f \text{ in the Fibonacci sequence}\}.$$ □

Example 4.2. Proceed as in the above example, except that now R_H is:

h1. $< s > \longrightarrow < q\#q >$
h2. $< I\#J > \longrightarrow < tI >< t'J > | < Iq\#JqII >$
h3. $< tqI > \longrightarrow a < tI >$
h4. $< t > \longrightarrow \lambda$
h5. $< t'qJ > \longrightarrow b < t'J >$
h6. $< t' > \longrightarrow \lambda$

Note that a derivation is of the form:

$$z \Longrightarrow < q^1\#q^1 > \Longrightarrow < q^2\#q^4 > \Longrightarrow < q^3\#q^9 > \Longrightarrow$$

$$\Longrightarrow < q^4\#q^{16} > \Longrightarrow < q^5\#q^{25} > \Longrightarrow \ldots$$

Using the first part of h2 and h3–h6 a strict hypervariable $< q^n\#q^m >$ derives the terminal word $a^n b^m$. Hence, it is easy to note that

$$L_2 = \{a^n b^m \mid n > 0 \text{ and } m = n^2\}.$$ □

Example 4.3. Let R_H be:

h1. $< s > \longrightarrow < tIq >< t'Jq >< Iq\#Jq >$
h2. $< I\#IJ > \longrightarrow < I\#J >$
h2′. $< IJ\#I > \longrightarrow < I\#J >$
h2″. $< I\#I > \longrightarrow < t''I >$
h3. $< tqI > \longrightarrow a < tI >$
h3′. $< t'qI > \longrightarrow b < t'I >$
h3″. $< t''qI > \longrightarrow c < t''I >$
h4. $< t > \longrightarrow \lambda$
h4′. $< t' > \longrightarrow \lambda$
h4″. $< t'' > \longrightarrow \lambda$

The reader should observe that h1 starts a derivation and h2, h2′, h2″ describe the algorithm of Euclid to compute the greatest common divisor of two (nonzero) natural numbers. The remaining rules are used to generate the terminal word. Hence, it is easy to see that the language generated is

$$L_3 = \{a^n b^m c^k \mid n, m > 0 \text{ and } k = gcd(n, m)\}.$$ □

The generative power of W-grammars was established by Sintzoff, [125]:

Theorem 4.1. *A language L is recursively enumerable iff there exists a W-grammar G, such that $L = L(G)$.* □

Aad Van Wijngaarden improved the above result in [137]:

Theorem 4.2. *A language L is recursively enumerable iff there exists a W-grammar G such that $L = L(G)$ and G has at most one metavariable.* □

4.2 Membership problem and parsing

Restrictions on W-grammars such that the language generated is a recursive language were studied by P. Deussen and K. Mehlhorn, see [36] and [37]:

Definition 4.5. *A hyperrule* $< u_0 > \longrightarrow a_0 < u_1 > a_1 \ldots < u_n > a_n$ *is*

- *left bounded (lb) iff for any* X *in* M*, if* X *has at least one occurrence in* u_0*, then* X *has at least one occurrence in* $< u_1 > \ldots < u_n >$*.*
- *right bounded (rb) iff for any* X *in* M*, if* X *has at least one occurrence in* $< u_1 > \ldots < u_n >$*, then* X *has at least one occurrence in* $< u_0 >$*.*
- *strict left bounded (slb) iff for any* X *in* M*, the number of occurrences of* X *in* u_0 *is less than or equal to the number of occurrences of* X *in* $< u_1 > \ldots < u_n >$ *and, moreover, the length of* u_0 *with respect to* V *is less than or equal to the length of* $u_1 \ldots u_n$ *with respect to* V*.*
- *strict right bounded (srb) iff for any* X *in* M*, the number of occurrences of* X *in* u_0 *is greater than or equal to the number of occurrences of* X *in* $< u_1 > \ldots < u_n >$ *and, moreover, the length of* u_0 *with respect to* V *is greater than or equal to the length of* $u_1 \ldots u_n$ *with respect to* V*.* □

Definition 4.6. *A W-grammar* G *is of type* L *(*R*), iff all the hyperrules are* λ*-free and lb (rb) and all the hyperrules of the form* $< u > \longrightarrow < v >$ *are slb (srb). The class of languages generated by W-grammars of type* L *(*R*) is denoted by* L_L *(*L_R*).* □

Theorem 4.3.

$$L_L = L_R = EXSPACE,$$

the class of languages acceptable by off-line Turing machines in exponential space. □

L. Wegner used the above restrictions to develop a parsing algorithm for a subclass of W-grammars, see [135]. The algorithm of Wegner can be used for a very large class of languages. In the sequel we present this parsing algorithm. We start by considering some definitions and terminology. Let $G = (G_1, G_2, z)$ be a W-grammar, where $G_1 = (M, V, R_M)$ and $G_2 = (H, \Sigma, R_H)$. A sequence $E_L = r_{i_1}, r_{i_2}, \ldots, r_{i_n}$ of hyperrules is called a *left parse* of $w \in (H_S \cup \Sigma)^*$ if there exists at least one leftmost derivation D of w in G such that $D = \alpha_0, \alpha_1, \ldots, \alpha_n = w$ and for $1 \leq j \leq n-1$, $\alpha_{j-1} \Longrightarrow \alpha_j$ by use of a strict rule from $R_S(r_{i_j})$. Note that for context-free grammars there is a one to one correspondence between leftmost derivations and left parses. However, for W-grammars this is not the case. Moreover, a stronger result does hold:

Theorem 4.4. *It is undecidable whether or not an arbitrary W-grammar* G *has the property that for each leftmost derivation there is at most one left parse.*

Proof. Let $G_i = (V_N^i, V_T^i, S_i, P_i)$, $i = 1, 2$ be two arbitrary context-free grammars and define the W-grammar G such that $M = V_N^1 \cup V_N^2$, $V = V_T^1 \cup V_T^2$, $R_M = P_1 \cup P_2$,
$H = \{z, < S_1 >, < S_2 >\}$, $\Sigma = \{a\}$, $R_H = \{z \longrightarrow < S_1 >, < S_1 > \longrightarrow a, < S_2 > \longrightarrow a\}$, where the start symbol of G is z.

Note that there is a leftmost derivation with more than one left parse iff $L(G_1) \cap L(G_2) \neq \emptyset$. But the emptiness of the intersection of context-free languages is undecidable and hence the theorem holds. □

Definition 4.7. *Let* $G = (G_1, G_2, z)$ *be a W-grammar, where* $G_1 = (M, V, R_M)$ *and* $G_2 = (H, \Sigma, R_H)$*. Let* h *be in* H*. The hypernotion system associated to* h *is the 4-tuple* $HS_h = (M, V, h, R_M)$*, and* h *is referred to as the axiom of* HS_h*. The language defined by* HS_h *is*

$$L(HS_h) = \{t(h) \mid t \in T\}.$$ □

The hypernotion systems are extended in the natural way for axioms $h \in (H \cup \Sigma)^+$.

Definition 4.8. *Let* $HS_h = (M, V, h, R_M)$ *be a hypernotion system, where* $h = X_1 X_2 \ldots X_n$, $X_i \in (M \cup V), 1 \leq i \leq n$*.* HS_h *is uniquely assignable (u.a.) if for all* $w \in L(HS_h)$ *there is exactly one decomposition* $w = w_1 w_2 \ldots w_n$ *such that* $t(X_i) = w_i$, $1 \leq i \leq n$, $t \in T$*.* □

Definition 4.9. *A W-grammar* G *is left-hand side uniquely assignable (lhs u.a.) if for every hyperrule* $(h_0 \longrightarrow h_1 h_2 \ldots h_n) \in R_H$ *the hypernotion system* (M, V, h_0, R_M) *is uniquely assignable.*

A W-grammar G *is right-hand side uniquely assignable (rhs u.a.) if for every hyperrule* $(h_0 \longrightarrow h_1 h_2 \ldots h_n) \in R_H$ *the hypernotion system* $(M, V \cup \Sigma, h_1 h_2 \ldots h_n, R_M)$ *is uniquely assignable.* □

Definition 4.10. *A W-grammar* G *is locally unambiguous if*

(i) for every leftmost derivation D *of* α *in* G *there is exactly one left parse* E *of* α *in* G *and given* D *one can effectively find* E*, and*
(ii) for every left parse E *of* α *in* G *there is exactly one leftmost derivation* D *of* α *in* G *and given* E *one can effectively find* D*.* □

Theorem 4.5. *For every W-grammar* G *one can effectively find a locally unambiguous W-grammar* G' *such that* $L(G) = L(G')$*.* □

The next step is to define a canonical context-free grammar which simulates the derivations in a W-grammar under special assumptions.

Definition 4.11. *Let* $G = (G_1, G_2, z)$ *be a W-grammar, where* $G_1 = (M, V, R_M)$ *and* $G_2 = (H, \Sigma, R_H)$ *and let* r *be a hyperrule,*

$$r : h_0 \longrightarrow h_1 h_2 \ldots h_m, \text{ where } h_i \in H \cup \Sigma, 1 \leq i \leq m.$$

A cross-reference of r is an m-tuple $(x_1, x_2, \ldots, x_m)$ such that

(i) if $h_i \in \Sigma$, then $x_i = h_i$; otherwise $x_i = h_0'$ for some hyperrule $h_0' \longrightarrow \alpha$ in R_H or $x_i = \lambda$, if $\lambda \in L(M, V, h_i, R_M)$, and

(ii) $L(M, V, h_1 h_2 \ldots h_m, R_M) \cap L(M, V, x_1' x_2' \ldots x_m', R_M) \neq \emptyset$ where x_i' is obtained from x_i by renaming metanotions in x_i such that they are distinct from those in x_j for $i \neq j$, $1 \leq i, j \leq m$. □

Definition 4.12. *Let $G = (G_1, G_2, z)$ be a W-grammar, where $G_1 = (M, V, R_M)$ and $G_2 = (H, \Sigma, R_H)$. The skeleton (canonical) grammar associated to G is the context-free grammar $G_{sk} = (V_N, \Sigma, z, P)$ defined as follows:*

$$V_N = \{h_0 \mid (h_0 \longrightarrow \alpha) \in R_H\},$$

$$P = \bigcup_{r \in R_H} R_{sk}(r), \text{ where}$$

$$R_{sk}(r) = \{h_0 \longrightarrow x_1 x_2 \ldots x_m \mid r = (h_0 \longrightarrow h_1 h_2 \ldots h_m) \in R_H,\ h_i \in (H \cup \Sigma)$$

$$\text{for } 1 \leq i \leq m \text{ and } (x_1, x_2, \ldots, x_m) \text{ a cross-reference of } r\}.$$ □

Definition 4.13. *A skeleton grammar G_{sk} associated to a W-grammar G is proper if for all $r, r' \in R_H$ with $r \neq r'$, $R_{sk}(r) \cap R_{sk}(r') = \emptyset$.* □

Theorem 4.6. *A skeleton grammar G_{sk} associated to a W-grammar G is proper iff all hyperrules $r = (h_0 \longrightarrow h_1 h_2 \ldots h_m)$ and $r' = (h_0' \longrightarrow h_1' h_2' \ldots h_n')$ have pairwise distinct cross-references.* □

Definition 4.14. *Let G_{sk} be the skeleton grammar associated to a W-grammar G. Let $E = r_{i_1}, r_{i_2}, \ldots, r_{i_n}$ be a left parse of $w \in L(G)$ and let $E' = r'_{i_1}, r'_{i_2}, \ldots, r'_{i_n}$ be a left parse of $w \in L(G_{sk})$. E' is referred as corresponding to E if for all $1 \leq j \leq n$, $r'_{i_j} \in R_{sk}(r_{i_j})$.* □

Theorem 4.7. *For every W-grammar G and for every left parse E of $w \in L(G)$ in G, there exists exactly one corresponding left parse E' of $w \in L(G_{sk})$ in G_{sk}.* □

Corollary 4.1. *If G is a W-grammar and if G_{sk} is its associated skeleton grammar, then $L(G) \subseteq L(G_{sk})$.* □

Corollary 4.2. *If G is a W-grammar such that the skeleton grammar G_{sk} of G is proper, then for each two left parses E_1, E_2 of $w \in L(G)$*

$$E_1 = E_2 \text{ iff } E_1' = E_2',$$

where E_i' denotes the corresponding left parse of E_i, $i = 1, 2$. □

Definition 4.15. *A W-grammar G is ambiguous if for some $w \in L(G)$ there is more than one leftmost derivation of w in G. Otherwise G is unambiguous.* □

Theorem 4.8. *If G is a locally unambiguous W-grammar and if the skeleton grammar associated to G, G_{sk}, is proper and unambiguous, then G is unambiguous.* □

The above results lead to the following parsing algorithm for a restricted class of W-grammars, see [135]:

Algorithm:

Input: A W-grammar G locally unambiguous, the proper and unambiguous skeleton grammar G_{sk} associated to G, a word $w \in \Sigma^*$.

Output: "yes" if $w \in L(G)$, "no" otherwise.

Method:

Step 1. Apply any of the known parsing algorithms for context-free grammars, see for instance [2, 3], to G_{sk} and w to obtain the left parse E'. If $w \notin L(G_{sk})$ then the output is "no".

Step 2. Compute the left parse E in G starting from the start symbol z (if G is left-hand side u.a. and rightbound) or from w (if G is right-hand side u.a. and leftbound). Apply hyperrule r to those strict notions which correspond to the terminals and nonterminals used in the derivation step in which skeleton rule $r' \in R_{sk}(r)$ was applied. Because G is locally unambiguous, giving the handle and hyperrule is sufficient to reduce α_i to α_{i-1} or, respectively, to extend α_i to α_{i+1}. The output is "no" and stop if r cannot be applied or, otherwise, if the derivation in G is complete, then the output is "yes" and stop.

Observe that the above algorithm requires as input the unambiguous skeleton-grammar G_{sk}. This condition cannot be removed, since G_{sk} is a context-free grammar and for context-free grammars it is undecidable whether the grammar is unambiguous or not.

There are other interesting results in the area of W-grammars. Some of these results concern normal forms for W-grammars, see for example S. Greibach, [55], J. van Leeuwen, [87], P. Turakainen, [131]. The parsing problem is studied also in J. L. Baker, [11], P. Dembinsky and J. Maluszynski, [33], A. J. Fisher, [42], C. H. A. Koster, [80]. The generative power of W-grammars is studied in J. de Graaf and A. Ollongren, [54]. Many other formalisms can be expressed in terms of W-grammars, such as the first order logic, P. Deussen, [36], W. Hesse, [59], the recursively enumerable functions, I. Kupka, [82]. For a bibliography of van Wijngaarden grammars the reader is referred to [38].

4.3 W-grammars and complexity

W-grammars provide a framework to evaluate the complexity of recursively enumerable languages, see [92, 91, 95]. Our presentation below requires some basic knowledge about recursive functions on the part of the reader.

Assume that S is a fixed alphabet and let ω be the set of natural numbers.

Definition 4.16. *A criterion over S is a recursive sequence, $C = (C_n)_{n\in\omega}$, such that for any $n \in \omega$, $C_n \subseteq S^*$ is a recursive language.* □

Let $D = (D_n)_{n\in\omega}$ be a given class of generative devices such that, for any language L over S, L is a recursively enumerable language iff there is a d in D such that $L(d) = L$, where $L(d)$ is the language generated by d.

Definition 4.17. *A generative Blum space (GBS) is an ordered system $\mathcal{B} = (S, D, A, C)$ where S, D, C are as above and $A = (A_i)_{i\in\omega}$ is the set of cost functions, i.e., a set of partial recursive functions that satisfy the following two axioms:*

GB1. $A_i(n) < \infty$ iff $L(d_i) \cap C_n \neq \emptyset$.
GB2. the predicate $R(i, x, y) = 1$ iff $A_i(x) = y$ and $R(i, x, y) = 0$ otherwise, is a total recursive predicate. □

Let W be the Gödel enumeration, $W = (w_i)_{i\in\omega}$, of all recursively enumerable languages over S using W-grammars as generative devices. For a W-grammar, G, let K_n be the n-th Kleene's iteration of the W system associated to G (the extension for W-grammars of the algebraic system of equations associated to a context-free grammar, see [81]). Define the cost functions, $TW = (TW_i)_{i\in\omega}$, such that they measure the "time" necessary for the device w_i with respect to the criterion C, to generate the language $L(w_i)$. If there is an $r \in \omega$, such that $K_r^i \cap C_n \neq \emptyset$ then $TW_i(n) = \min\{r \mid K_r^i \cap C_n \neq \emptyset\}$, else $TW_i(n) = \infty$.

Note that the ordered system $\mathcal{B}_T = (S, W, TW, C)$ is a GBS.

Definition 4.18. *Let $\mathcal{B} = (S, D, A, C)$ be a GBS and let f be a recursive function. The generative complexity class bounded by f is*

$$C_{\mathcal{B}}(f(n)) = \{L \mid \text{there is an } i \in \omega \text{ such that}$$

$$L(d_i) = L \text{ and } A_i(x) \leq f(x) \text{ almost everywhere}\}.$$ □

If the GBS is $\mathcal{B}_T$ then the generative complexity class bounded by f is denoted by $TIMEW(f(n))$.

Theorem 4.9. *(the linear speed-up theorem) For every GBS, $\mathcal{B}_T = (S, W, TW, C)$, for every recursive function f and for every constant $q > 0$,*

$$TIMEW(f(n)) = TIMEW(qf(n)).$$ □

Definition 4.19. *A W-grammar G is related to the graph of a function $f : \omega \longrightarrow \omega$ if $\{i, \#\} \subseteq V$, $L_X = i^*$ for every $X \in M$, $R_H \subset H \times H^*$ and, for all $n, m \in \omega$*

$$z \Longrightarrow_G^* < i^n \# i^m > \text{ iff } f(n) = m.$$ □

Definition 4.20. *A function $f : \omega \longrightarrow \omega$ is TW-self bounded if f is a constant function or there is a W-grammar G related to the graph of f and there are constants $n_0, c_1, c_2 \in \omega$ such that for every $n \geq n_0$, $f(n) > 1$ and, moreover, if the hyperrule $< i^n \# i^{f(n)} > \longrightarrow a$ is added to the hyperrules of G, then for all $p \in \omega$, $K_p = \{a\}$, where $p = c_1 f(n) + c_2$.*

□

Let TWSB be the class of all TW-self bounded functions. Intuitivelly, a function f in TWSB has the property that the graph of f can be generated by a W-grammar that is W-time bounded by f.

Theorem 4.10. *(the TW-separation property) Let f be in TWSB such that $f(n) > \log_2 n$ almost everywhere and let g be a function such that $inf_{n\to\infty}(g(n)/f(n)) = 0$. There exists a language L such that, with respect to the criterion $C = (a^n b^*)_{n\in\omega}$, L is in $TIMEW(f(n))$ but L is not in $TIMEW(g(n))$.*

Proof. Consider the language

$$L = \{a^n b^m \mid m = 2^{f(n)}, n \in \omega\}.$$

We prove that $L \in TIMEW(f(n))$. Since $f \in TWSB$, there exists a W-grammar $G = (G_1, G_2, z)$, where $G_1 = (M, V, R_M)$ and $G_2 = (H, \Sigma, R_H)$, such that G is related to the graph of f and let $n_0, c_1, c_2 \in \omega$ be the constants for which the Definition 4.20 is satisfied for f.

Define the W-grammar G', such that $M' = M \cup \{N, N_1\}$, $V' = V \cup \{\alpha, \beta\}$, $\Sigma' = \Sigma \cup \{a, b\}$, $R_{M'} = R_M \cup \{A \longrightarrow B \mid A \in \{N, N_1\}, B \in M\}$, $z' = z$ and $R'_H = R_H \cup \{h_1, \ldots, h_6\}$ where:

ι_1. $< N \# N_1 > \longrightarrow < \alpha N >< \beta N_1 >$
h_2. $< \alpha N N > \longrightarrow < \alpha N >< \alpha N >$
$\quad_3$. $< \alpha N N i > \longrightarrow a < \alpha N >< \alpha N >$
ι_4. $< \alpha > \longrightarrow \lambda$
h_5. $< \beta N i > \longrightarrow < \beta N >< \beta N >$
h_6. $< \beta > \longrightarrow b$

Observe that $L(G') = L$ and note that a terminal derivation in G' is equivalent with a derivation of the following form:

$$z' \Longrightarrow_{G'}^* < i^n \# i^{f(n)} > \Longrightarrow_{h_1} < \alpha i^n >< \beta i^{f(n)} > \Longrightarrow_{h_2 - h_6}^* a^n b^{2^{f(n)}}.$$

One can easily see that $K'_{p_1}(< \alpha i^n >) = \{a^n\}$ and $K'_{p_2}(< \beta i^{f(n)} >) = \{b^{2^{f(n)}}\}$, where $p_1 = \log_2 n + 1$ and $p_2 = f(n) + 1$. Let p_3 be $max\{p_1, p_2\} + 1$.

Since $f(n) \geq \log_2 n$, it follows that $p_3 = f(n) + 2$. Now observe that for all $n \geq n_0$, $K'_p = \{a^n b^{2^n}\}$, where $p = (c_1 + 1)f(n) + c_2 + 2$ and hence $L \in TIMEW(f(n))$.

Now we show that $L \notin TIMEW(g(n))$. Assume by contrary that there exists a W-grammar G'' such that $L(G'') = L$ and $TW_{G''}(n) \leq g(n)$ almost everywhere. Define the constants:

$$j_1 = max\{\mid \sigma_0 \ldots \sigma_k \mid\mid (< u_0 > \longrightarrow \sigma_0 < u_1 > \sigma_1 \ldots < u_k > \sigma_k) \in R''_H\}$$

$$j_2 = max\{\mid \alpha \mid\mid (< u > \longrightarrow \alpha) \in R''_H, \alpha \in \Sigma^*\}$$

$$j_3 = max\{q \mid (< u_0 > \longrightarrow \sigma_0 < u_1 > \sigma_1 \ldots < u_q > \sigma_q) \in R''_H\}.$$

Note that for every $n \geq 1$, for every strict hypernotion $< h >$ and for every $v \in K''_n(< h >)$

$$\mid v \mid \leq j_3^{n-1} j_2 + j_1 \sum_{i=0}^{n-2} j_3^i.$$

Let j be $max\{j_1, j_2, j_3\}$. It follows that

$$\mid v \mid \leq j^n + j \sum_{i=0}^{n-2} j^i = \sum_{i=1}^{n} j^i \leq n j^n \leq 2^n j^n = (2j)^n.$$

Denote $s = 2j$ and note that for every $v \in K''_n$, $\mid v \mid \leq s^n$. Therefore, for each $v \in K''_{g(n)}$ it follows that $\mid v \mid \leq s^{g(n)}$. Since $TW_{G''}(n) \leq g(n)$ almost everywhere we deduce that $K''_{g(n)} \cap C_n \neq \emptyset$ almost everywhere. Hence $a^n b^{2^{f(n)}} \in K''_{g(n)}$ almost everywhere. Thus

$$2^{f(n)} < \mid a^n b^{2^{f(n)}} \mid \leq s^{g(n)} \text{ almost everywhere.}$$

Hence $\log_s 2 \leq g(n)/f(n)$ almost everywhere. But this is contrary to the assumption that $inf_{n\to\infty}(g(n)/f(n)) = 0$ and, consequently $L \notin TIMEW(g(n))$. □

Theorem 4.11. *(a dense hierarchy) If $C = (a^n b^*)_{n\in\omega}$, then for all real numbers $0 < \alpha < \beta$, $TIMEW(n^\alpha)$ is a proper subset of $TIMEW(n^\beta)$.* □

5. Attribute grammars

5.1 Definitions and terminology

Attribute grammars were introduced by D. Knuth, see [78, 79]. Informally, *attribute grammars* consist of:

1) a context-free grammar $G = (V_N, V_T, S, P)$ such that S does not occur on the right side of any production.
2) two disjoint sets of symbols associated with each symbol of G, called *inherited* and *synthesized attributes*, where S must have at least one synthesized attribute and no inherited attributes.
3) a *semantic domain* D_α for each attribute α.
4) a set of *semantic rules* associated with each production. Each semantic rule is a function

$$\varphi_{pi\alpha} : D_{\alpha_1} \times \ldots \times D_{\alpha_k} \longrightarrow D_\alpha$$

which defines the value of an attribute α of an instance i of a symbol appearing in a production p in terms of the values of attributes $\alpha_1, \ldots, \alpha_k$ associated with other instances of symbols in the same production.

Attribute grammars are also referred to as *K-systems*. In the sequel, symbols with indices denote (rename) the symbols without indices. For instance, X_1, X_2 rename the symbol X.

We consider the first example in [78]. The same example is considered also in [17, 26, 27].

Example 5.1. Let $G = (V_N, V_T, S, P)$ be the context-free grammar with: $V_N = \{B, L, N\}$, $V_T = \{0, 1, \cdot\}$, $S = N$. The productions from P are listed in sequence together with the semantic rules associated:

Productions ; Semantic rules

1. $B \longrightarrow 0$; $v(B) = 0$
2. $B \longrightarrow 1$; $v(B) = 2^{s(B)}$
3. $L \longrightarrow B$; $v(L) = v(B)$, $s(B) = s(L)$, $l(L) = 1$
4. $L_1 \longrightarrow L_2 B$; $v(L_1) = v(L_2) + v(B)$, $s(B) = s(L_1)$, $s(L_2) = s(L_1) + 1$, $l(L_1) = l(L_2) + 1$
5. $N \longrightarrow L$; $v(N) = v(L)$, $s(L) = 0$
6. $N \longrightarrow L_1 \cdot L_2$; $v(N) = v(L_1) + v(L_2)$, $s(L_1) = 0$, $s(L_2) = -l(L_2)$ □

It is easy to observe that the language generated by G is the set of all rational numbers in binary notation.

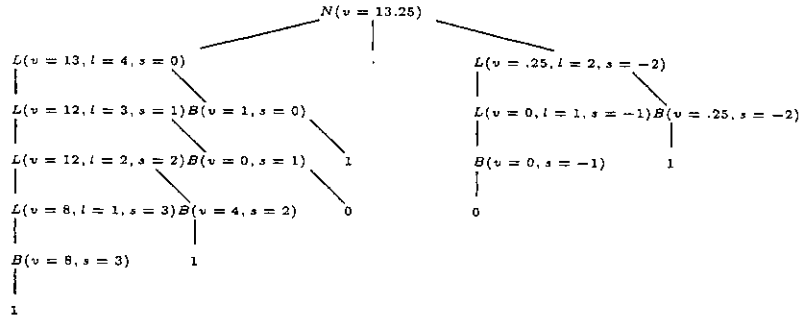
The main purpose of the semantic rules is to define the meaning of each string from $L(G)$.

The synthesized attributes are $v(B)$, $v(L)$, $l(L)$ and $v(N)$, based on the attributes of the descendants of the nonterminal symbol. Inherited attributes are $s(B)$ and $s(L)$, and they are based on the attributes of the ancestors.

Synthesized attributes are evaluated bottom up in the tree structure, while the inherited attributes are evaluated top down in the tree structure.

For instance, consider the string 1101.01 from $L(G)$.

The evaluated structure corresponding to this string is:



Note that $v(N) = 13.25$ and that the binary representation of 13.25 is the string 1101.01.

Here the semantic domains are: Z (the set of integer numbers) for $s(B)$, $s(L)$, $l(L)$ and Q (the set of rational numbers) for $v(B)$, $v(L)$ and $v(N)$.

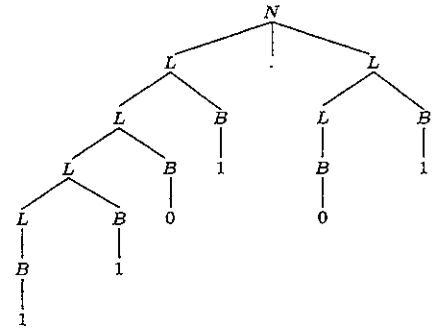
5.2 Algorithms for testing the circularity

The semantic rules are *well defined* or *noncircular* iff they are formulated in such a way that all attributes can always be computed at all nodes, in any possible derivation tree.

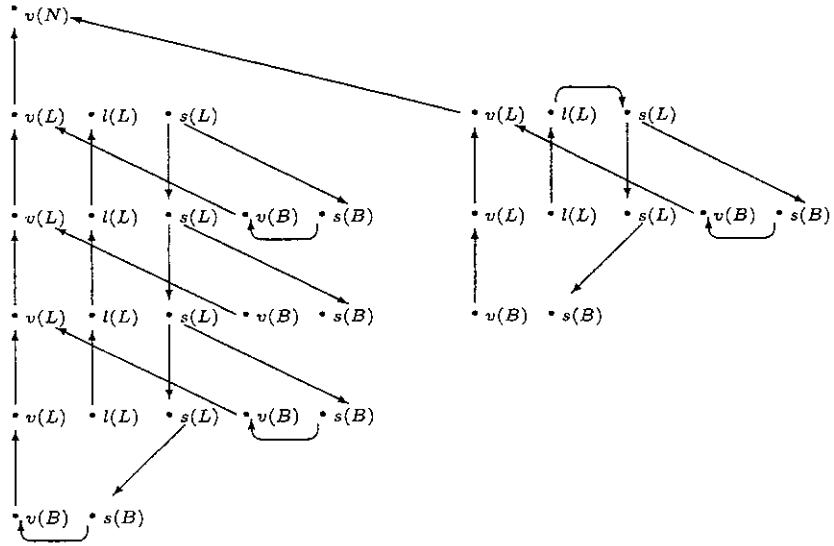
Note that the definition of semantic rules can lead to circular definitions of attributes and consequently such attributes cannot be evaluated. Moreover, since in general there are infinitely many derivation trees, it is important to have an algorithm for deciding whether or not a given grammar has well-defined semantic rules. An algorithm for testing this property was given in [78, 79].

Let G be an attribute grammar. Without loss of generality, we may assume that the grammar contains no useless productions, i.e., that each production of G appears in the derivation of at least one terminal string. Let $\mathcal{T}$ be any derivation tree obtainable in the grammar, allowed to have any symbol of V as the label of the root. Define the directed graph $\mathcal{D}(\mathcal{T})$ corresponding to $\mathcal{T}$ by taking the ordered pairs (X, α) as vertices, where X is a node of $\mathcal{T}$ and α is an attribute of the symbol which is the label of the node X. The arcs of $\mathcal{D}(\mathcal{T})$ go from (X_1, α_1) to (X_2, α_2) if and only if the semantic rule for the value of the attribute α_2 depends directly on the value of the attribute α_1.

For instance, assume G is the attribute grammar from Example 5.1 and consider the derivation tree $\mathcal{T}$:



The directed graph $\mathcal{D}(\mathcal{T})$ is:



The semantic rules are well defined iff no directed graph $\mathcal{D}(\mathcal{T})$ contains an oriented cycle.

Assume that for some $\mathcal{D}(\mathcal{T}_1)$ there is an oriented cycle. Since G contains no useless productions, there is an oriented cycle in some $\mathcal{D}(\mathcal{T})$ for which the root of $\mathcal{T}$ has the label S. $\mathcal{T}$ is a derivation tree of the language for which it is not possible to evaluate all the attributes.

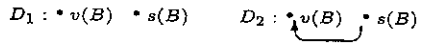
Let p be the production

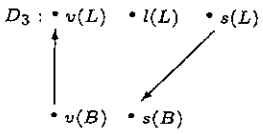
$$X_{p_0} \longrightarrow X_{p_1} \dots X_{p_{n_p}}.$$

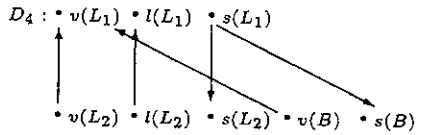
Corresponding to p, consider the directed graph $\mathcal{D}_p$ defined as follows:

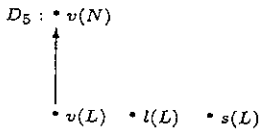
the vertices are (X_{p_j}, α) where α is an attribute associated to X_{p_j}, and arcs emanate from $(X_{p_{k_j}}, \alpha_i)$ to (X_{p_j}, α) for $0 \leq j \leq n_p$ and α is a synthesized attribute, if $j = 0$, and α is an inherited attribute of X_{p_j}, if $j > 0$.

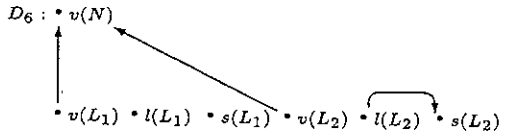
For example, the six productions of G from Example 5.1 have the following six directed graphs:









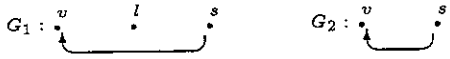


Note that each directed graph $\mathcal{D}(\mathcal{T})$ is obtained as the superposition of such directed graphs $\mathcal{D}_p$.

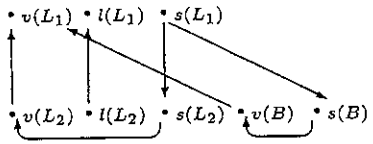
Let A_X be the set of attributes of X. Consider a production p and assume that G_j, $1 \leq j \leq n_p$, is any directed graph whose vertices are a subset of $A_{X_{p_j}}$.

Denote by $\mathcal{D}[G_1, \dots, G_p]$ the directed graph obtained from $\mathcal{D}_p$ by adding an arc from (X_{p_j}, α) to (X_{p_j}, α') whenever there is an arc from α to α' in G_j.

For instance, assume that



and if $\mathcal{D}_4$ is the directed graph defined above, then $\mathcal{D}_4[G_1, G_2]$ is



Theorem 5.1. *There exists an algorithm to decide whether or not the semantic rules of an attribute grammar are well defined.*

Proof. Let X be in $V = V_N \cup V_T$ and let $S(X)$ be a set of directed graphs on the vertices A_X. Initially $S(X) = \emptyset$ if $X \in V_N$, and $S(X)$ is the single directed graph with vertices A_X and no arcs if $X \in V_T$.

New directed graphs are added to the sets $S(X)$ until no further directed graphs can be added.

Let m be $card(P)$ and let the k-th production be:

$$X_{k0} \longrightarrow X_{k1} X_{k2} \dots X_{kn_k}.$$

For each integer p, $1 \leq p \leq m$, and for each j, $1 \leq j \leq n_p$, choose a directed graph $\mathcal{D}'_j$ in $S(X_{pj})$. Add to the set $S(X_{p0})$ the directed graph whose vertices are $A_{X_{p0}}$ and whose arcs go from α to α' if and only if there is an oriented path from (X_{p0}, α) to (X_{p0}, α') in the directed graph

$$\mathcal{D}_p[\mathcal{D}'_1, \dots, \mathcal{D}'_{n_p}].$$

Note that only finitely many directed graphs are possible and hence this procedure cannot continue forever.

If $\mathcal{T}$ is a derivation tree with root X, let $\mathcal{D}'(\mathcal{T})$ be the directed graph with vertices A_X whose arcs go from α to α' iff there is an oriented path from (X, α) to (X, α') in $\mathcal{D}(\mathcal{T})$.

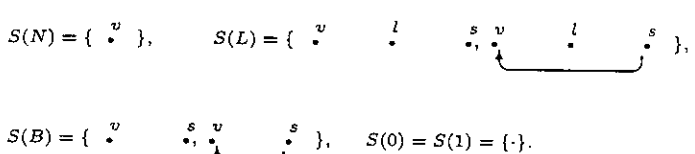
We claim that for all $X \in V$, $S(X)$ is the set of $\mathcal{D}'(\mathcal{T})$, where $\mathcal{T}$ is a derivation tree with root X.

To prove this claim, note that the construction adds to $S(X)$ only such directed graphs $\mathcal{D}'(\mathcal{T})$. Moreover, from each such $\mathcal{D}'(\mathcal{T})$ one can obtain an appropriate derivation tree $\mathcal{T}$.

Conversely, if $\mathcal{T}$ is a derivation tree, we can show by induction on the number of nodes in $\mathcal{T}$ that $\mathcal{D}'(\mathcal{T})$ is in the corresponding $S(X)$. □

Corollary 5.1. *The semantic rules of an attribute grammar are well defined iff none of the graphs $\mathcal{D}_p[\mathcal{D}'_1, \ldots, \mathcal{D}'_{n_p}]$, for any choice of p and $\mathcal{D}'_j \in S(X_{p_j})$, $1 \leq j \leq n_p$ contains an oriented cycle.* □

Example 5.2. Consider the attribute grammar G from Example 5.1. Using the above algorithm we obtain



Now one can apply the above corollary in order to verify that the semantic rules of G are well defined. □

The above algorithm, also referred to as the algorithm for testing the noncircularity of definitions in an attribute grammar, requires exponential time, see [69, 74].

5.3 Restricted attribute grammars

A more restrictive property of attribute grammars, called the *strong noncircularity* was studied in [26, 27]. The property of strong noncircularity is decidable in polynomial time. Most attribute grammars arising in applications such as compiler construction have this property, see [34].

Notation. Let $G = (V_N, V_T, S, P)$ be an attribute grammar. The set of synthesized attributes associated to a symbol $X \in V = V_N \cup V_T$ is denoted by A^s_X and the set of inherited attributes associated to X is A^h_X. Moreover, $A_X = A^s_X \cup A^h_X$, $A^s = \bigcup_{X \in V} A^s_X$, $A^h = \bigcup_{X \in V} A^h_X$, $A = A^s \cup A^h$.

Definition 5.1. *Let $p \in P$ be a production*

$$p : X \longrightarrow X_1 X_2 \ldots X_n.$$

Consider the sets

$$W_1(p) = \{a(X_i) \mid 1 \leq i \leq n, a \in A_X\},$$

$$W_0(p) = \{a(X) \mid a \in A_X\},$$

and

$$W(p) = W_1(p) \cup W_0(p).$$

Define on $W(p)$ a binary relation, denoted $\longrightarrow_p$, by

$w \longrightarrow_p w'$ iff w' occurs on the right-hand side of a semantic rule defining w and the semantic rule is associated to p. □

Definition 5.2. *An argument selector is a mapping*

$$\gamma : A^s \times V_N \longrightarrow \mathcal{P}(A^h)$$

such that:

$\gamma(a, X) \subseteq A^h_X$, if $a \in A^s_X$ and $\gamma(a, X) = \emptyset$, otherwise. □

Argument selectors are ordered componentwise, i.e., $\gamma \subseteq \gamma'$ iff $\gamma(a, X) \subseteq \gamma'(a, X)$ for all $a \in A^s$ and all $X \in V_N$.

Definition 5.3. *For each argument selector γ, we define the binary relation, $\longrightarrow_\gamma$, on $W_1(p)$ as:*

$w \longrightarrow_\gamma w'$ iff $w = a(X_i)$, $w' = y(X_i)$ and $y \in \gamma(a, X_i)$ for some $1 \leq i \leq n$, $a \in A^s_{X_i}$ and $y \in A^h_{X_i}$. □

Denote by $\longrightarrow_{p,\gamma}$ the relation $\longrightarrow_p \cup \longrightarrow_\gamma$ considered only on $W_1(p)$.

Definition 5.4. *An attribute grammar is strongly noncircular if there exists an argument selector γ such that*

(1) γ is closed, i.e., for all $p \in P$, $p : X \longrightarrow X_1 X_2 \ldots X_n$, for all $y \in A^h_X$ and $a \in A^s_X$, whenever $a(X) \longrightarrow_p y(X)$ or $a(X) \longrightarrow_p w \longrightarrow^*_{p,\gamma} w' \longrightarrow_p y(X)$ for some $w, w' \in W_1(p)$, then $y \in \gamma(a, X)$.

(2) γ is noncircular, i.e., for all $p \in P$ as in (1), the relation $\longrightarrow_{p,\gamma}$ has no cycles, that is, there exists no $w \in W_1(p)$ such that $w \longrightarrow^+_{p,\gamma} w$. □

Definition 5.5. *Let t be a derivation tree and let $W(t)$ be the set of all attributes associated to vertices of t. The binary relation γ_t is defined on $W(t)$ by $w \longrightarrow_t w'$ iff w' occurs on the right-hand side of an equation defining w.* □

We still consider some related subclasses of attribute grammars.

Let a be in A^s, let y be in A^h and let X be in $V^a_N \cap V^y_N$. The attribute a *may call* y at X iff $a(X) \longrightarrow^*_t y(X)$ for some derivation tree t.

Let $CALL(a, X)$ denote the set of inherited attributes that a may call at $X \in V$. It is easy to see that $CALL(a, X) \subseteq \gamma(a, X)$ whenever $\gamma(a, X)$ is a closed argument selector.

Definition 5.6. *An attribute grammar is benign if $CALL$, as an argument selector, is noncircular.*

An attribute grammar is ordered if it is noncircular and if there exists a family $(\theta_X)_{X \in V_N}$, where θ_X is a partial order on A_X such that for all derivation trees t, for all occurrences u in t, whenever $a(u) \longrightarrow^*_t b(u)$, then $b \theta_Y a$, where Y is the label of the root of t. □

The following theorems, see [26, 27], show the relations between these families of attribute grammars.

Theorem 5.2. *Ordered attribute grammars are contained in strongly noncircular ones. Strongly noncircular attribute grammars are contained in benign ones. Benign attribute grammars are contained in noncircular ones. All of the above containments are proper.* □

Theorem 5.3. *The property of an attribute grammar being ordered (strongly noncircular, benign, noncircular) is decidable and moreover,*

(i) whether an attribute grammar is ordered or strongly noncircular can be decided in polynomial time,
(ii) deciding whether an attribute grammar is benign or noncircular requires exponential time. □

Theorem 5.4. *For any attribute grammar there exists a minimal closed argument selector γ_0. The attribute grammar is strongly noncircular iff γ_0 is noncircular.* □

Using the above theorem, the following polynomial time algorithm for deciding the property of strong noncircularity was introduced in [26]. In the sequel an argument selector γ is considered as a set of triples:

$$R = \{(y, a, X) \mid y \in \gamma(a, X)\} \subseteq A \times A \times V_N.$$

The argument selector corresponding to R is denoted by $\gamma(R)$.

Algorithm:

Step 1. Set $R_0 = \emptyset$.

Step 2. For all $i \geq 1$ compute:

$$R_i = R_{i-1} \cup \{(y, a, X) \mid \exists p \in P, p : X \longrightarrow \alpha, \text{ such that } y \in A_X^h, a \in A_X^s$$
$$\text{and } a(X) \longrightarrow_{p,\gamma(R_{i-1})}^* y(X)\}.$$

Step 3. Stop as soon as $R_i = R_{i-1}$, and the result is $\gamma_0 = \gamma(R_i)$.

After the computation of γ_0, one has to check its noncircularity which can be also done in polynomial time.

Definition 5.7. *An attribute grammar is purely synthesized if $A^h = \emptyset$. An attribute grammar is nonnested if inherited attributes are defined only in terms of inherited attributes.* □

Clearly, every purely synthesized attribute grammar is a nonnested attribute grammar.

Theorem 5.5. *Every nonnested attribute grammar is strongly noncircular.*

Proof. Assume G is a nonnested attribute grammar and let γ be the argument selector:

$$\gamma(a, X) = A_X^h.$$

Note that γ is closed. Consider now the sequence:

$$w_1 \longrightarrow_{p,\gamma} w_2 \longrightarrow_{p,\gamma} \cdots \longrightarrow_{p,\gamma} w_m.$$

It may be either $a(X) \longrightarrow_p y(X)$ for some $a \in A^s$ and $y \in A^h$, or

$$a(X) \longrightarrow_p b(X_i) \longrightarrow_{p,\gamma} y(X_j) \longrightarrow_p z(X)$$

for some $a, b \in A^s$, $y, z \in A^h$, $X, X_i, X_j \in V_N$, or it is a subsequence of the last one. Consequently, there cannot be cycles. Therefore, γ is noncircular. Hence, G is strongly noncircular. □

Corollary 5.2. *Every purely synthesized attribute grammar is strongly noncircular.* □

5.4 Other results

In [17] it is proved that each attribute grammar can be converted to an equivalent purely synthesized one that possibly uses the μ operator, i.e., the least fixed point of a function. The idea is to consider the semantic rules as a system of equations. After this the inherited attributes are eliminated using the so-called recursion theorem.

Purely synthesized attribute grammars can be related to initial algebra semantics, see [17] and see [53] for initial algebra semantics.

Attribute grammars have been used in parsing theory and practice. *Attribute translation grammars* were introduced in [89] as a device to specify a translation. The translated symbols can have a finite set of attributes. An attributed translation grammar is in some sense a generalization of a context-free grammar. A context-free grammar is first extended to a translation grammar and then the attributes are added. Attributed translation grammars apply also ideas from syntax directed translations.

Attribute grammars were extended to *attributed tree grammars* in [12]. An algorithm for testing noncircularity of attributed tree grammars is described in [12].

LR-attributed grammars are a combination between $LR(k)$ and attribute grammars. They have both features, an efficient parsing algorithm and an algorithm to evaluate the attributes. Informally, an *LR*-attributed grammar is an attribute grammar that has the possibility of attribute evaluation during the *LR* parsing by an *LR parser/evaluator*. An *LR parser/evaluator* is a deterministic algorithm which is an extension of a shift/reduce *LR* parsing algorithm in such a way that it evaluates all synthesized attribute instances of a node in a derivation tree as soon as it has recognized this node and

has parsed the subtree rooted in this node. The LR parser/evaluator also computes all inherited attributes of this node.

Various families of LR-attributed grammars are studied and compared in [4]. See also [134] for an early approach of this subject.

Attribute grammars are used to study natural languages, too. They can be used to encode an algebraic specification of a natural language and attributed translation is used to compute representations of the "meaning" of a sentence at different levels of abstraction, including a treatment of semantic ambiguities. For more details, as well as for an example of a parser that translates a significant subset of English into predicate logic, the reader is referred to [107].

In [88] a system is presented, called *Vinci*, for generating sentences and phrases in natural languages. The system is based on attribute grammars and has been used for experiments in linguistics, and for teaching linguistic theory. Attribute grammars provide the basis of the syntax definition and the use of attributes ensures the "semantic agreement" between verbs and their subjects and objects, nouns and adjectives, etc. For instance, the sentence "The bear ate the banana" is accepted, whereas the sentence "The table ate colourless green ideas" is not accepted.

For a large bibliography on attribute grammars see [34].

6. Patterns

6.1 Erasing and nonerasing patterns

The study of patterns goes back to the beginning of this century. Some of the classical results of Axel Thue, see [128], can be expressed in terms of patterns and pattern languages. Recent research concerning inductive inference, learning theory, combinatorics on words and term rewriting is also related to the study of patterns. Classical devices such as grammars and automata characterize a language completely. Patterns provide an alternative method to define languages, giving more leeway.

Let Σ and V be two disjoint alphabets. Σ is the alphabet of *terminals* and V is the alphabet of *variables*. A *pattern* is a word over $\Sigma \cup V$. The language defined by a pattern α consists of all words obtained from α by leaving the terminals unchanged and substituting a terminal word for each variable X. The substitution has to be *uniform*, i.e., different occurrences of X have to be replaced by the same terminal word. In the case when variables have to be replaced always by *nonempty* words, the pattern will be referred to as a *nonerasing pattern*, or *NE-pattern*. These patterns have been studied systematically first by Dana Angluin, see [5, 6].

However, the situation is essentially different if the empty word is allowed in the substitutions. The study of such patterns, referred to as *erasing patterns*, or *E-patterns*, was initiated in [70].

The formal definitions are the following. Denote by $H_{\Sigma,V}$ the set of all morphisms from $(\Sigma \cup V)^*$ to $(\Sigma \cup V)^*$.

Definition 6.1. *The language generated by an E-pattern $\alpha \in (\Sigma \cup V)^*$ is defined as*

$$L_{E,\Sigma}(\alpha) = \{w \in \Sigma^* \mid w = h(\alpha) \text{ for some } h \in H_{\Sigma,V}$$
$$\text{such that } h(a) = a \text{ for each } a \in \Sigma\}.$$

The language generated by an NE-pattern $\alpha \in (\Sigma \cup V)^*$ is defined as

$$L_{NE,\Sigma}(\alpha) = \{w \in \Sigma^* \mid w = h(\alpha) \text{ for some } \lambda\text{-free } h \in H_{\Sigma,V}$$
$$\text{such that } h(a) = a \text{ for each } a \in \Sigma\}. \quad \square$$

Whenever Σ is understood we use also the notations $L_E(\alpha)$ and $L_{NE}(\alpha)$. Note that in the sequel the symbol "$\subset$" denotes the proper inclusion.

A *sample* is a finite nonempty language. For many purposes, both theoretical and practical, given a sample F it is useful to find a pattern α such that $F \subset L_{NE}(\alpha)$ or $F \subset L_E(\alpha)$ and, furthermore, α describes F as closely as possible. Consider the following example, see [70].

Example 6.1. Assume that the terminal alphabet is $\Sigma = \{0, 1\}$ and let F be the following sample

$$F = \{0000, 000000, 001100, 00000000, 00111100, 0010110100, 0001001000\}.$$

For each E-pattern α from the following list, F is a subset of the language $L_E(\alpha)$.

$$\alpha_1 = XYX,\ \alpha_2 = XYYX,\ \alpha_3 = XXYXX,$$
$$\alpha_4 = XXYYXX,\ \alpha_5 = 00X00,\ \alpha_6 = 00XX00. \quad \square$$

The above list can be continued with other examples of patterns as well as with descriptions of the form XX^R, where X^R is the mirror image of X, $00XX^R00$ or for instance XX^P, where X^P is some permutation of X. In the sequel we will not consider the mirror image or the permutation of variables.

Finding a pattern common to all words in a given sample is especially appropriate if the sample set is growing, for instance through a *learning process*, or if one may enquire whether or not some specified words belong to the set. For instance, a positive answer to the query 00011000 will reject the patterns α_4 and α_6, but not the patterns α_1, α_2, α_3, α_5.

Also, trying to infer a pattern common to all words in a given sample is a very typical instance of the process of *inductive inference*, that is, the process of inferring general rules from specific examples. For more details on this line see for instance [7] or [65].

For an interconnection between patterns and *random numbers* see [70].