ISSN 0105-8517

# Computation sequences:

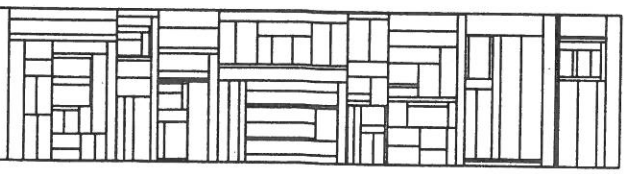
# A way to characterize subclasses of attribute grammars

by

Hanne Riis Nielson

DAIMI PB-138
November 1981

Computer Science Department
**AARHUS UNIVERSITY**
Ny Munkegade – DK 8000 Aarhus C – DENMARK
*Telephone: 06 – 12 83 55*



# COMPUTATION SEQUENCES:
## a way to characterize classes of attribute grammars

Hanne Riis Nielson

Abstract

A computation sequence for a derivation tree specifies a way of walking through the tree evaluating all the attributes of all nodes. By requiring that each derivation tree has a computation sequence with a certain property, it is possible to give simple characterizations of well-known subclasses of attribute grammars. Especially the absolutely non-circular attribute grammars are considered.

## 1. INTRODUCTION

Attribute grammars are introduced by Knuth to associate meanings with strings of context free languages [8]. The meaning of a string is obtained by first constructing a derivation tree for the string and then evaluating the attributes associated with the tree. An evaluator for an attribute grammar is a program that as input takes a string and computes its meaning according to the attribute grammar. It is usually regarded as desirable that the evaluator gives exactly one meaning to each string of the context free language. This can be achieved by putting restrictions on the attribute grammars. An example is to require that the underlying context free grammar is unambiguous as well as that the attribute grammar itself is well-defined ([8], [9]). The well-definedness condition can be replaced by other conditions e.g. that the attribute grammar must be ordered ([3], [6]) or it must be absolutely non-circular ([7]).

We believe that when writing an attribute grammar one naturally thinks in terms of derivation trees and orders of evaluating the attributes at its nodes. Therefore the restrictions imposed by e.g. an evaluator generator system ought to be explained in these terms. For instance the well-definedness condition can be explained by: for each derivation tree it must be possible to find an order of evaluating all the attributes of the tree without violating their dependencies (see e.g. [11]). The property of being ordered (as defined in [6]) has been investigated in [3] and can be explained by: for each symbol of the context free grammar there must be a fixed ordering of its attributes such that they can be evaluated in that order at any node labelled by the symbol in any derivation tree.

Computation sequences (introduced in [11]) are used in [3] to formalize the intuition of walking through a derivation tree evaluating all the attributes of all nodes without violating their dependencies. Engelfriet and Filè ([3]) leave it as an open problem whether there is a natural characterization of the absolutely non-circular attribute grammars ([7]) in terms of computation sequences. In this paper we address this problem as well as we present a general framework for defining subclasses of

attribute grammars. The results of [10] show how a characterization of subclasses of attribute grammars in terms of computation sequences can be used in the definition of evaluators.

In the next section we briefly introduce our notation concerning attribute grammars. In section 3 we define a computation sequence. Although our definition is different from that of [11] (and [3]) it turns out that the concepts are equivalent. Section 3 is closed by giving a general framework for characterizing subclasses of attribute grammars. In section 4 we give a characterization of the absolutely non-circular attribute grammars and prove its correctness. Finally, in section 5 we briefly consider the pass, sweep and visit properties of [4].

## 2. PRELIMINARIES

This section contains a review of Knuth's original definition of an attribute grammar ([8]).

An attribute grammar (abbreviated AG) is an extension of a context free grammar G $(V_N, V_T, P, S)$ consisting of nonterminals, terminals, productions, and initial nonterminal, respectively. To each symbol F of $V_N$ there is associated a finite set $\mathcal{I}(F)$ of inherited attributes and a finite set $\mathcal{S}(F)$ of synthesized attributes. We shall assume that $\mathcal{I}(F)$ and $\mathcal{S}(F)$ are disjoint sets for all symbols F and furthermore that S has no inherited attributes. Each (inherited or synthesized) attribute $\alpha$ takes values in a set $D_\alpha$. To each production $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ ($F_j \in V_N$, $v_j \in V_T^*$ for $0 \leq j \leq k$) of P there is associated a set of semantic functions. For each $\alpha$ in $\mathcal{S}(F_0)$ (resp. in $\mathcal{I}(F_j)$, $1 \leq j \leq k$) there is a (semantic) function $f_{0\alpha}$ (resp. $f_{j\alpha}$) of functionality $D_{\alpha_1} \times \ldots \times D_{\alpha_m} \to D_\alpha$ (m and $\alpha_i$ depend on $\alpha$ and j). Each $\alpha_i$ is an attribute of either $\mathcal{I}(F_0)$ or of $\mathcal{S}(F_\ell)$ for some $\ell$, $1 \leq \ell \leq k$.

The semantic functions are used to assign meanings to derivation trees and thereby strings of the underlying context free language. Consider a derivation tree t and a node n in t. Let $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ be the production applied at n. For each $\alpha$ in $\mathcal{S}(F_0)$ the function $f_{0\alpha}$: $D_{\alpha_1} \times \ldots \times D_{\alpha_m} \to D_\alpha$ associated with p can be used to determine the value of $\alpha$ at n when the values of all the attributes $\alpha_1, \ldots, \alpha_m$ have been determined. Similarly, for $\alpha$ in $\mathcal{I}(F_j)$ ($1 \leq j \leq k$) the function $f_{j\alpha}$ associated with p is used to determine the value of $\alpha$ at the j'th son of n. If it is possible to determine the values of all attributes of any node in t as described above then the meaning of t will be t decorated with these values.

We shall assume that the context free grammar G has no useless symbols ([ 1 ]). Consider a derivation tree t of G. Each interior node of t is labelled with a symbol from $V_N$ and has (also) associated with it a sequence of positive integers called the location of the node. The location of the root of t is the empty string $\lambda$ and the location of the j'th interior son of a node with location n is $n\S j$ (we abbreviate $\lambda\S j$ by j). We will not distinguish between an interior node and its location. The subtree of t whose root is n is denoted $t_{(n)}$. The part of t with $t_{(n)}$ removed except n itself is denoted $t^{(n)}$.

Finally, we present an AG that will be used as an example throughout this paper. The underlying grammar has two nonterminals A and S with attributed

$$\mathcal{I}(S) = \emptyset \qquad \mathcal{I}(A) = \{\alpha, \beta\}$$
$$\mathcal{S}(S) = \{\epsilon\} \qquad \mathcal{S}(A) = \{\gamma, \delta\}$$

Each attribute takes values in the set of integers. The productions of the underlying grammar are listed below with the associated semantic function, We use the following notation. If $\sigma$ is an attribute of $F_j$ $(0 \leq j \leq k)$ in the production p: $F_0 \rightarrow v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ then we shall call it $[j.\sigma]$. The semantic function $f_j : D_{\sigma_1} \times \ldots \times D_{\sigma_m} \rightarrow D_\sigma$ defining $\sigma$ will be written as an equation

$$[j.\sigma] = f_{j\sigma}([j_1.\sigma_1], \ldots, [j_m.\sigma_m])$$

where $\sigma_i$ is an attribute of $F_{j_i}$ $(1 \leq i \leq m)$.

| | | | |
|---|---|---|---|
| $p_1$: S → AA | $[0.\epsilon] = [1.\gamma] + [2.\gamma]$ | | |
| | $[1.\alpha] = [2.\delta]$ | $[1.\beta] = 1$ | |
| | $[2.\alpha] = [1.\delta]$ | $[2.\beta] = 2$ | |
| $p_2$: A → aA | $[0.\gamma] = [1.\gamma]$ | $[0.\delta] = [1.\delta]$ | |
| | $[1.\alpha] = [0.\alpha]$ | $[1.\beta] = [0.\beta]$ | |
| $p_3$: A → b | $[0.\gamma] = [0.\alpha]$ | $[0.\delta] = 0$ | |
| $p_4$: A → c | $[0.\gamma] = 2$ | $[0.\delta] = [0.\beta]$ | |

## 3. COMPUTATION SEQUENCE

In this section we define the concept of a computation sequence, which is intended to formalize the intuition of an order of evaluating all the attributes of a derivation tree without violating their dependencies. The attributes will be evaluated during a walk through the tree. When arriving at a node we may evaluate some of the (inherited or synthesized) attributes of the node and continue to either the father, a brother or a son of the node or stay at the node itself. In order for a walk to be a computation sequence we will require that

i) each attribute of each node is evaluated exactly once

ii) the order of evaluating the attributes does not violate their dependencies.

We now give formal definitions of the concepts.

Consider a derivation tree t of the underlying grammar of an AG and let n be a node in t where the production $p: F_0 \to v_0F_1v_1\cdots v_{k-1}F_kv_k$ is applied. A walk through $t_{(n)}$ is a string of symbols (m,A) where m is a node of $t_{(n)}$ and A is a set of attributes of that node, defined recursively by

- the empty string $\lambda$ is a walk
- if $s_i$ is a walk through $t_{(n\S j_i)}$ for $1 \le i \le r$, $1 \le j_i \le k$, $(r \ge 0)$, $A_I \subseteq \mathcal{I}(F_0)$ and $A_S \subseteq \mathcal{S}(F_0)$ then $s = (n, A_I)s_1\cdots s_r(n, A_S)$ is a walk through $t_{(n)}$
- if s and s' are walks through $t_{(n)}$ then so is ss'

Consider a walk s through the derivation tree t. Then $s(t_{(n)})$ is the walk through $t_{(n)}$ obtained by removing all pairs (n',A') from s where n' does not occur in $t_{(n)}$, and $s(t^{(n)})$ is the walk through t obtained by removing all pairs (n',A') from s where $n' \neq n$ and n' is a node in $t_{(n)}$. It can easily be verified that $s(t_{(n)})$ and $s(t^{(n)})$ are indeed walks through $t_{(n)}$ and t, respectively.

Let $p: F_0 \to v_0F_1v_1\cdots v_{k-1}F_kv_k$ be the production applied at n. We define a modified restriction s(n,p) to n and its direct sons $n\S 1, \ldots, n\S k$ by by

- removing all pairs (n',A') from s where $n' \neq n\S j$ $(1 \le j \le k)$, and $n' \neq n$
- replacing each of the remaining pairs (n,A) by (0,A) and each of the pairs (n§j,A) by (j,A), $1 \le j \le k$.

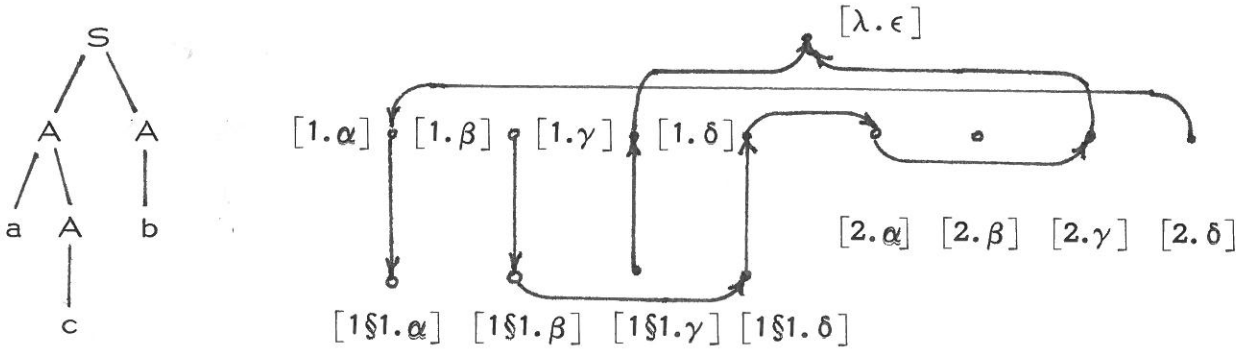
Thus $s(n,p)$ is a sequence of the form $(h_1,A_1)\ldots(h_r,A_r)$ where $0 \le h_i \le k$ and $A_i \subseteq \mathcal{I}(F_{h_i}) \cup \mathcal{S}(F_{h_i})$ for $1 \le i \le r$. For any sequence $\pi_p$ of that form and for $0 \le j \le k$ define

$$\pi_p(j) = A_{i_1} \ldots A_{i_m}$$

where $i_1 < \ldots < i_m$ and $\{\ell \mid h_\ell = j,\ 1 \le \ell \le r\} = \{i_1,\ldots,i_m\}$.
We write $s(n)$ for $s(n,p)(0)$.

Example 1

To illustrate the concepts defined so far consider the following derivation tree t of the example AG. We have also shown a graph indicating the dependencies between the attributes of t; $[n.\sigma]$ is the node for the attribute $\sigma$ of the symbol labelling n.




$$s_1 = (\lambda,\emptyset)\,(1,\{\alpha\})(1\S1,\{\alpha,\beta\})(1\S1,\{\gamma,\delta\})(1,\{\gamma,\delta\})$$
$$(2,\{\alpha,\beta\})(2,\{\delta\})(2,\{\alpha\})(2,\{\gamma\})(\lambda,\emptyset)$$

is a walk through t. We have

$$s_1(t_{(1)}) = (1,\{\alpha\})(1\S1,\{\alpha,\beta\})(1\S1,\{\gamma,\delta\})(1,\{\gamma,\delta\})$$

$$s_1(t^{(1)}) = (\lambda,\emptyset)(1,\{\alpha\})(1,\{\gamma,\delta\})(2,\{\alpha,\beta\})(2,\{\delta\})(2,\{\alpha\})(2,\{\gamma\})(\lambda,\emptyset)$$

$$s_1(1,p_2) = (0,\{\alpha\})(1,\{\alpha,\beta\})(1,\{\gamma,\delta\})(0,\{\gamma,\delta\})$$

$$s_1(1) = \{\alpha\}\ \{\gamma,\delta\}$$

□

In order for a walk to satisfy requirement i) above we define (ordered) partitions of attributes of symbols and productions. A <u>partition of the attributes of a symbol</u> F ([3]) is a non-empty finite sequence

$$\pi = A_1 \ldots A_{2m}$$

satisfying

- $A_{2i-1} \subseteq \mathcal{I}(F)$ and $A_{2i} \subseteq \mathcal{S}(F)$ for $1 \leq i \leq m$
- $A_1 \cup \ldots \cup A_{2m} = \mathcal{I}(F) \cup \mathcal{S}(F)$
- $A_i \cap A_j = \emptyset$ for $i \neq j$, $1 \leq i, j \leq 2m$

Let $\Pi_F$ be the set of partitions for the symbol F. A <u>partition of the attributes of a production</u> p: $F_0 \rightarrow v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ is a sequence

$$\pi_p = (h_1, A_1) \ldots (h_r, A_r)$$

where $0 \leq h_i \leq k$ for $1 \leq i \leq r$ and for $0 \leq j \leq k$ $\pi_p(j)$ is a partition of the attributes of $F_j$. Let $\Pi_{p,\pi}$ be the set of partitions $\pi_p$ of attributes of p with $\pi_p(0) = \pi$.

<u>Example 2</u>

For the walk $s_1$ through t of example 1 we have that neither $s_1(1)$ nor $s_1(2)$ are partitions of the attributes of A but $s_1(1 \S 1)$ is. $s_1(\lambda, p_1)$ is not a partition of the attributes of $p_1$. The following walk $s_2$ through t is such that $s_2(1)$, $s_2(1 \S 1)$ as well as $s_2(2)$ are partitions of the attributes of A:

$$s_2 = (\lambda, \emptyset)(1, \{\alpha, \beta\})(1 \S 1, \{\alpha, \beta\})(1 \S 1, \{\gamma, \delta\})(1, \{\gamma, \delta\})$$
$$(2, \{\beta\})(2, \{\delta\})(2, \{\alpha\})(2, \{\gamma\})(\lambda, \{\epsilon\})$$

Note that $s_2$ mentions all the attributes of all nodes of t exactly once (requirement i)).

However, an order of evaluating the attributes of t must evaluate the attribute $\delta$ of node 2 before evaluating the attribute $\alpha$ of node 1 in order not to violate the dependencies between the attributes.

□

The dependencies between the attributes of a production $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ can be expressed in a dependency graph for p. A dependency graph for p will have one node denoted $[j.\alpha]$ for each attribute $\alpha$ of $\mathcal{I}(F_j) \cup \mathcal{S}(F_j)$, $0 \leq j \leq k$, and maybe some arcs. The semantic functions associated with p define the dependency graph D(p) for p as follows. If $f_{j\alpha}: D_{\alpha_1} \times \ldots \times D_{\alpha_m} \to D_\alpha$ is a semantic function associated with p and $\alpha_i$ is an attribute of $F_{j_i}$ $(1 \leq i \leq m)$ then there will be an arc from $[j_i.\alpha_i]$ to $[j.\alpha]$ in D(p). These are the only arcs of D(p).

Let D be a dependency graph for p.
The partition $\pi_p = (h_1, A_1) \ldots (h_r, A_r)$ of the attributes of p satisfies the dependency graph D if for all $\ell$, $1 \leq \ell \leq r$:

$$\forall \alpha \in A_\ell: \text{ if there is an arc from } [i.\beta] \text{ to } [h_\ell.\alpha] \text{ in D}$$
$$\text{then for some } \ell', \ \ell' < \ell, \ \beta \in A_{\ell'} \text{ and } i = h_{\ell'}.$$

Example 3

For the walk $s_2$ of example 2 we have that $s_2(\lambda, p_1)$ does not satisfy $D(p_1)$ but $s_2(1\S1, p_4)$ does satisfy $D(p_4)$. Let s be the walk

$$s = (\lambda, \emptyset)(2, \{\beta\})(2, \{\delta\})$$
$$(1, \{\alpha, \beta\})(1\S1, \{\alpha, \beta\})(1\S1, \{\gamma, \delta\})(1, \{\gamma, \delta\})$$
$$(2, \{\alpha\})(2, \{\gamma\})(\lambda, \{\epsilon\}).$$

A comparison with the graph of example 1 shows that the dependencies between the attributes are respected (requirement ii)).

From the definition below it will follow that s is a computation sequence of the tree of example 1.

□

Let t be a derivation tree and let $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ be the production applied at a node n in t. A walk s through $t_{(n)}$ is a computation sequence for $t_{(n)}$ if

- s(n, p) is a partition of the attributes of p satisfying D(p)
- $s(t_{(n\S j)})$ is a computation sequence for $t_{(n\S j)}$ for $1 \leq j \leq k$.

The above conditions formalize requirements i) and ii) mentioned in the beginning of this section. The definition of a computation sequence presented in [11] (and [3]) can be proven equivalent to that given above.

An AG is well-defined if it satisfies the circularity test of [9]. We have the following characterization of the well-defined AGs:

> An AG is <u>well-defined</u> if and only if each derivation tree has a computation sequence.

The correctness of this characterization follows easily by modifying the proof for the correctness of the circularity test of [9].

We will use a common framework when characterizing subclasses of AGs. We will first define a certain property $\mathcal{P}$ of computation sequences. Next we define the AG to have property $\mathcal{P}$ if each derivation tree has a computation sequence with property $\mathcal{P}$. Finally, we have to prove that the class of AGs with property $\mathcal{P}$ is equal to the subclass of AGs we want to characterize.

The ordered AGs of [6] are characterized by computation sequences in [3]. Intuitively, an AG is ordered if for each symbol of the underlying context free grammar there is a fixed ordering of its attributes such that they can be evaluated in that order at any node labelled by the symbol in any derivation tree. Define an <u>assignment of partitions to the symbols</u> of $V_N$ as any family $A = \{A_F\}_{F \in V_N}$ of partitions satisfying $A_F \in \Pi_F$ for $F \in V_N$.

Consider now a derivation tree t with the production p: $F_0 \rightarrow v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ applied at a node n in t. A computation sequence s for $t_{(n)}$ is <u>ordered with respect to G</u> if

- $s(n) = \mathcal{G}_{F_0}$
- $s(t_{(n\S j)})$ is a computation sequence for $t_{(n\S j)}$ that is ordered with respect to $\mathcal{G}$.

We can now give the following characterization of the ordered AGs ([3]):

> An AG is <u>ordered</u> if and only if there exists an assignment $\mathcal{G}$ of partitions to the symbols of $V_N$ such that any derivation tree has a computation sequence that is ordered with respect to $\mathcal{G}$.

In the next section we consider the absolutely non-circular AGs and in the final section we consider subclasses of AGs with pass, sweep and visit properties.

## 4. ABSOLUTELY NON-CIRCULAR AGs

The characterizations of the well-defined and the ordered AGs given in the previous section have been more or less straightforward. The reason for this is that the subclasses have been defined with some intuitive idea of a computation sequence in mind. A formal definition of the subclass has then been given in terms of a circularity test of a rather complicated dependency graph. The absolutely non-circular AGs are different in that respect.

In order to define the subclass of absolutely non-circular AGs, we need some concepts concerning dependency graphs. A dependency graph for a symbol $F$ will have one node denoted $[\alpha]$ for each attribute $\alpha$ of $\mathcal{I}(F) \cup \mathcal{S}(F)$ and maybe some arcs. Given the dependency graph $D(p)$ for the production $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ and given dependency graphs $\Gamma_j$ for the symbols $F_j$ for $1 \leq j \leq k$, the composition

$$D(p)[\![ \Gamma_1 \ldots \Gamma_k ]\!]$$

of $D(p)$ with $\Gamma_1, \ldots, \Gamma_k$ is constructed as follows: $D(p)[\![ \Gamma_1 \ldots \Gamma_k ]\!]$ is a dependency graph for $p$ and there will be an arc from $[j.\alpha]$ to $[i.\beta]$ if and only if either there is an arc from $[j.\alpha]$ to $[i.\beta]$ in $D(p)$ or $i = j$ and there is an arc from $[\alpha]$ to $[\beta]$ in $\Gamma_j$. From a dependency graph $\Gamma_p$ for $p$ we can derive a dependency graph $\Gamma_0$ for $F_0$. There will be an arc from $[\alpha]$ to $[\beta]$ in $\Gamma_0$ if and only if there is a non-empty path from $[0.\alpha]$ to $[0.\beta]$ in $\Gamma_p$.

The dependency graph $D(F)$ for a symbol $F$ is the least graph such that for any production $p: F \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ the graph derived from $D(p)[\![ D(F_1) \ldots D(F_k) ]\!]$ is a subgraph of $D(F)$. It is not difficult to verify that the graphs $D(F)$ exist. We can now define

> An AG is absolutely non-circular if for each production $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ there are no cycles in $D(p)[\![ D(F_1) \ldots D(F_k) ]\!]$.

This definition is equivalent to that given in [7].

Consider a derivation tree t with the production p: $F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ applied at the node n in t. Does t have a computation sequence with some special properties when the AG is absolutely non-circular ? Intuitively, the graph $D(F_j)$ puts together all possible dependencies between the attributes of the root of a derivation tree with root labeled $F_j$. Thus our guess will be that t has a computation sequence s where $s(n\S j)$ is independent of the shape of the subtree $t_{(n\S j)}$ $(1 \leq j \leq k)$. Therefore at n, given the partition s(n) of the attributes of $F_0$ it should be possible to determine s(n, p) without knowing anything about the subtrees $t_{(n\S 1)}, \cdots, t_{(n\S k)}$. This motivates the following definitions:

A <u>top-down assignment of partitions to the symbols</u> of $V_N$ is a family $A = \{A_p\}_{p \in P}$ of mappings such that for $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$

$$A_p = \Pi_{F_0} \to \Pi_{F_1} \times \cdots \times \Pi_{F_k}$$

For a partition $\pi$ of the attributes of $F_0$ we define a computation sequence s for the subtree $t_{(n)}$ to be <u>$\pi$-uniform with respect to A</u> if

- $s(n) = \pi$
- $A_p(\pi) = (s(n\S 1), \ldots, s(n\S k))$
- $s(t_{(n\S j)})$ is a computation sequence for $t_{(n\S j)}$ that is $s(n\S j)$-uniform with respect to A $(1 \leq j \leq k)$.

Following our general framework we now define: An AG is <u>$\pi_0$-uniform</u> for a partition $\pi_0$ of the attributes of S if there exists a top-down assignment of partitions to the symbols of $V_N$ such that any derivation tree with root labelled S has a computation sequence that is $\pi_0$-uniform with respect to A.

Finally, an AG is <u>uniform</u> if it is $\pi_0$-uniform for some $\pi_0$.

Example 4

Our example AG is uniform: choose for instance $\pi_0 = \emptyset\{\varepsilon\}$ and A:

$A_{p_1}$ : $\emptyset\{\varepsilon\} \rightarrow (\{\alpha\ \beta\}\{\gamma\ \delta\},\{\beta\}\{\delta\}\{\alpha\}\{\gamma\})$

$A_{p_2}$ : $\{\alpha,\beta\}\{\gamma,\delta\} \rightarrow (\{\alpha,\beta\}\{\gamma,\delta\})$

$\{\beta\}\{\delta\}\{\alpha\}\{\gamma\} \rightarrow (\{\beta\}\{\delta\}\{\alpha\}\{\gamma\})$

$A_{p_3}, A_{p_4}$ : $\{\alpha,\beta\}\{\gamma,\delta\} \rightarrow ( )$
$\{\beta\}\{\delta\}\{\alpha\}\{\gamma\} \rightarrow ( )$

The computation sequence s of example 3 is $\pi_0$-uniform with respect to A.

□

We will now prove that the property of uniformity of computation sequences correctly characterizes the absolutely non-circular AGs:

Theorem 1

An AG is absolutely non-circular if and only if it is uniform.

□

The theorem follows from a series of lemmas. The first lemma shows that for any absolutely non-circular AG we can construct a top-down assignment of partitions to the symbols satisfying certain requirements. Next we show how to construct a computation sequence for a tree from computation sequences for its subtrees as well as a partition of the attributes of the production applied at the root. By putting these results together we get that any absolutely non-circular AG is uniform.

A partition $\pi = A_1,\ldots,A_{2m}$ of the attributes of a symbol F satisfies a dependency graph $\Gamma$ for F if for $1 \leq l \leq 2m$:

$\forall \alpha \in A_l$ : if there is an arc from $[\beta]$ to $[\alpha]$ in $\Gamma$
then for some $l'$, $l' < l$, $\beta \in A_{l'}$,

Lemma 1

For an absolutely non-circular AG there exists a top-down assignment $A$ of partitions to the symbols of $V_N$ such that if $p: F \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ and $\pi$ $(\neq \lambda)$ is a partition of the symbols of F satisfying D(F) then there exists a partition $\pi_p$ of the attributes of p satisfying

i) $\pi_p(0) = \pi$ and $\pi_p(j) \neq \lambda$ for $1 \leq j \leq k$

ii) $\pi_p$ satisfies $D(p)\ [\![D(F_1)\ldots D(F_k)]\!]$

iii) $A_p(\pi) = (\pi_p(1),\ldots,\pi_p(k))$

Proof

We will give an algorithm that for a production $p: F \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ and a partition $\pi$ $(\neq \lambda)$ of the attributes of F satisfying D(F) constructs a partition $\pi_p$ of the attributes of p satisfying i) and iii). We then define

$$A_p(\pi) = (\pi_p(1),\ldots,\pi_p(k))$$

Clearly the lemma will hold.

Let $\Gamma = D(p)\ [\![D(F_1)\ldots D(F_k)]\!]$ and $\pi = A_1\ldots A_{2m}$. For $B \subseteq \{[\,j.\alpha] \mid \alpha \in I(F_j) \cup S(F_j),\ 0 \leq j \leq k\}$ and $1 \leq j \leq k$ define

NEW-inh $(j, B, \Gamma)$ =

$\{[j.\alpha] \mid \alpha \in \mathcal{I}(F_j) - B$ and if there is an arc from $[\ell.\beta]$ to $[j.\alpha]$ in $\Gamma$ then $[\ell.\beta] \in B\}$

NEW-syn $(j, B, \Gamma)$ =

$\{[j.\alpha] \mid \alpha \in \mathcal{S}(F_j) - B$ and if there is an arc from $[\ell.\beta]$ to $[j.\alpha]$ in $\Gamma$ then $[\ell.\beta] \in B\}$

The algorithm is as follows:

$B := \emptyset$; $\pi_p := \lambda$;
FOR i = 1 TO m DO
BEGIN $B := B \cup \{[0.\alpha] \mid \alpha \in A_{2i-1}\}$; $\pi_p := \pi_p(0, A_{2i-1})$;
IF i = 1 THEN
BEGIN FOR each j with $\vartheta(F_j) = \emptyset$ or NEW-syn$(j, \emptyset, \Gamma) \neq \emptyset$, $1 \leq j \leq k$ DO
BEGIN $B_S := \{\alpha \mid [j.\alpha] \in$ NEW-syn $(j, \emptyset, \Gamma)\}$;
$B := B \cup$ NEW-syn $(j, \emptyset, \Gamma)$; $\pi_p := \pi_p\,(j, \emptyset)\,(j, B_S)$;
END;
END;
WHILE there is a j with NEW-inh $(j, B, \Gamma) \neq \emptyset$, $1 \leq j \leq k$ DO
BEGIN determine the least j with NEW-inh $(j, B, \Gamma) \neq \emptyset$, $1 \leq j \leq k$;
$B_I := \{\alpha \mid [j.\alpha] \in$ NEW-inh $(j, B, \Gamma)\}$; $B := B \cup$ NEW-inh $(j, B, \Gamma)$;
$B_S := \{\alpha \mid [j.\alpha] \in$ NEW-syn $(j, B, \Gamma)\}$; $B := B \cup$ NEW-syn $(j, B, \Gamma)$;
$\pi_p := \pi_p\,(j, B_I)\,(j, B_S)$;
END;
$B := B \cup \{[0.\alpha] \mid \alpha \in A_{2i}\}$; $\pi_p := \pi_p\,(0, A_{2i})$;
END

Observe that after each assignment to $\pi_p$ it holds that

$$B = \{[j.\alpha] \mid \text{there is a tuple } (j, C) \text{ in } \pi_p \text{ with } \alpha \in C,\ 0 \leq j \leq k\}.$$

By a rather technical proof (e.g. induction in the length of a path in $\Gamma$) one can prove that $\pi_p$ is a partition of the attributes of p satisfying i) and ii).

We omit the detailed proof and give instead some of the important observations. The mappings NEW-inh and NEW-syn are used to ensure that all the attributes on which a given attribute $\alpha$ depends have been added to $\pi_p$ before $\alpha$ is added. Because $\Gamma$ has no cycles the IF-clause in the algorithm ensures that all attributes of p are in the final $\pi_p$ and thereby that $\pi_p$ is a partition of the attributes of p. Since $\pi$ satisfies any subgraph of D(F) and so the graph derived from $\Gamma$, we get that $\pi_p$ satisfies $\Gamma$. It is easy to see that $\pi_p(0) = \pi$. The test on $I(F_j)=\emptyset$ in the IF-clause ensures that even if $S(F_j)=\emptyset$ we will get $\pi_p(j) \neq \lambda$. □

Lemma 2

Consider a derivation tree t of an AG and let p: $F_0 \to v_0F_1v_1\ldots v_{k-1}v_k$ be the production applied at the node n. Let $\pi_p$ be a partition for p satisfying D(p) and with $\pi_p(j) \neq \lambda$ for $0 \leq j \leq k$. Furthermore let $s_j$ be a computation sequence for $t_{(n\S j)}$ with $s_j(n\S j) = \pi_p(j)$ for $1 \leq j \leq k$. Then there exists a computation sequence s for $t_{(n)}$ with

- $s(n,p) = \pi_p$
- $s(t_{(n\S j)}) = s_j$ for $1 \leq j \leq k$

Proof Let $\pi_p(j) = A^j_1 \ldots A^j_{2m_j}$ for $0 \leq j \leq k$. Since $s_j(n\S j) = \pi_p(j)$ we can split $s_j$ into $m_j$ walks $s^1_j, \ldots, s^{m_j}_j$ such that $s^i_j(n\S j) = A^j_{2i-1}A^j_{2i}$ for $1 \leq i \leq m_j$, $1 \leq j \leq k$. Now let $u_1, \ldots, u_{m_0}$ be determined by

$$\pi_p = (0,A^0_1)u_1(0,A^0_2) \ldots (0,A^0_{2m_0-1})u_{m_0}(0,A^0_{2m_0})$$

The sequence $u_\ell$ consists of pairs of the form $(j,A^j_{2i-1})(j,A^j_{2i})$ where $1 \leq i \leq m_j$, $1 \leq j \leq k$. From $\pi_p$ we construct a walk s through $t_{(n)}$ by replacing each pair $(j,A^j_{2i-1})(j,A^j_{2i})$ in $u_1, \ldots, u_{m_0}$ by the walk $s^i_j$ and by replacing each $(0,A^0_\ell)$ in $\pi_p$ by $(n,A^0_\ell)$.

It is now easy to see that $s(t_{(n\S j)}) = s_j$ for $1 \leq j \leq k$ and $s(n,p) = \pi_p$. This proves that s is a computation sequence with the required properties.

□

Proof of the first part of theorem 1: Any absolutely non-circular AG is uniform. Let A be the top-down assignment of partitions to the symbols given by lemma 1. We can now show

(*) for any derivation tree t with root labelled F and for any partition $\pi$ ($\neq \lambda$) of the attributes of F satisfying D(F) there exists a computation sequence s for t that is $\pi$-uniform with respect to A.

This result follows easily by induction on the height of t using the properties of $A$ given by lemma 1 and that lemma 2 holds. Note that it is essential that $\pi \neq \lambda$ because the empty string need not be a computation sequence for t even if $I(F) = S(F) = \emptyset$.

From (*) it follows that the AG is $\pi$-uniform for any $\pi$ in $\Pi_S$.

In order to prove the second part of the theorem we first prove that the property of uniformity with respect to an assignment is preserved under substitution. A property $\mathcal{P}$ of computation sequences is <u>preserved under substitution</u> if for any derivation trees t and t' with root labelled S and any computation sequence s for t with property $\mathcal{P}$ the following holds: if for some node n, $t^{(n)} = t'^{(n)}$ then t' has a computation sequence s' with property $\mathcal{P}$ and with $s'(n) = s(n)$.

<u>Lemma 3</u>

Consider a $\pi_0$-uniform AG with top-down assignment $A$ of partitions to the symbols of $V_N$. Then the property of $\pi_0$-uniformity with respect to $A$ is preserved under substitution.

<u>Proof</u> Let t and t' be derivation trees with root labelled S and with $t^{(n)} = t'^{(n)}$ for some node n. Both t and t' have computation sequences s and s' respectively that are $\pi_0$-uniform with respect to $A$. By induction on the length of the path from the root of the trees to n we can easily show that $s(n) = s'(n)$. □

The proof of the second part of theorem 1 uses the following iterative definition of the dependency graphs D(F) for the symbols:

- $D_0(F)$ is the graph for F with no arcs
- $D_{i+1}(F)$ is the least graph such that for all productions $p: F \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$ the graph derived from $D(p)\,[\![D_i(F_1) \ldots D_i(F_k)]\!]$ is a subgraph of $D_{i+1}(F)$, $i \geq 0$.
- if for all F, $D_i(F) = D_{i+1}(F)$ then $D(F) = D_i(F)$.

Clearly the two definitions agree.

Lemma 4

Consider an AG and assume that there exists a property $\mathcal{P}$ of computation sequences such that i) each derivation tree with root labelled S has a computation sequence with property $\mathcal{P}$, and ii) property $\mathcal{P}$ is preserved under substitution. Then the AG is absolutely non-circular.

Proof By induction on 'i' we show

(**) for any derivation tree t with a computation sequence s that has property $\mathcal{P}$ we have that s(n) satisfies $D_i(F)$ for any node n in t labelled F.

For i = 0 it clearly holds. For the induction step consider a node n in t where the production $p: F \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$ is applied. From the induction hypothesis we have that s(n§j) satisfies $D_i(F_j)$ for $1 \leq j \leq k$. Since s(n,p) satisfies D(p) we get that s(n,p) satisfies $D(p)[\![D_i(F_1) \ldots D_i(F_k)]\!]$. This gives that s(n) satisfies the graph derived from $D(p)[\![D_i(F_1) \ldots D_i(F_k)]\!]$. Now we replace the subtree $t_{(n)}$ with another tree where the production $p': F \to v_0'F_1'v_1' \ldots v_{k'-1}'F_{k'}'v_{k'}'$ is applied. Let t' be the resulting tree

and let s' be the computation sequence that has property $\mathcal{P}$. Since $t(n) = t'(n)$ we get $s(n) = s'(n)$. Arguments as above give that $s'(n)$ satisfies the graph derived from $D(p')[\![D_i(F'_1) \ldots D_i(F'_{k'})]\!]$ and thereby $s(n)$ satisfies $D_{i+1}(F)$.

(**) it follows that at any node n of t, s(n) satisfies D(F) where F is el of n. Let again $p: F \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ be the production at n. Since s(n,p) satisfies D(p) and s(n§j) satisfies $D(F_j)$ for e get that s(n,p) satisfies $D(p)[\![D(F_1) \ldots D(F_k)]\!]$. But then annot contain any cycles and the AG is absolutely non-circular. s the proof of the lemma.

□

Proof of second part of theorem 1: Any uniform AG is absolutely non-circular. This follows immediately from lemmas 3 and 4.

□

From the lemmas above we also get the following characterization of the absolutely non-circular AGs:

Theorem 2

An AG is absolutely non-circular if and only if there is a property $\mathcal{P}$ of computation sequences such that i) every derivation tree with root labelled S has a computation sequence with property $\mathcal{P}$, and ii) property $\mathcal{P}$ is preserved under substitution.

□

To summarize the results of this section, [3] and [11] we have the following characterizations: An AG is

| | |
|---|---|
| non-circular | • each derivation tree has a computation sequence |
| absolutely non-circular | there exists a top-down assignment A of partitions to the symbols, and a partition $\pi_0$ of the attributes of S such that each derivation tree has a computation sequence s that is $\pi_0$-uniform with respect to A |

| | |
|---|---|
| ordered | • there exists an assignment $\mathbb{G}$ of partitions to symbols, such that each derivation tree has a computation sequence that is ordered with respect to $\mathbb{G}$. |

Note that with these characterizations it follows trivially that any ordered AG is absolutely non-circular and any absolutely non-circular AG is well-defined.

## 5. PASSES, SWEEPS AND VISITS

We have used computation sequences to model the intuition of walking through a derivation tree evaluating all its attributes without violating their dependencies. As the usual definition of the absolutely non-circular AGs is not obviously connected to a definition in terms of computation sequences we have especially paid attention to that class.

In [4] the properties of m-visit, m-sweep, m-alternating pass and m-pass are defined both in the pure (well-defined) case and in the simple (ordered) case. These properties can easily be defined in terms of computation sequences and by combining them with the uniformity we can define new subclasses of AGs.

The simplest of the four properties is the multipass (m-pass) property originally introduced by [2]. The multipass property requires that the attributes are evaluated during a sequence of left-to-right traversals over the tree. Consider a derivation tree t and let n be a node in t where the production p: $F_0 \rightarrow v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is applied. A left-to-right traversal of $t_{(n)}$ is a walk s through $t_{(n)}$ where

- $s = (n, A_l)s_1 \ldots s_k(n, A_S)$ for some $A_l$ and $A_S$
- $s_j$ is a left-to-right traversal of $t_{(n\S j)}$ for $1 \leq j \leq k$.

And we can define an m-pass computation sequence for $t_{(n)}$ as a computation sequence s composed of m left-to-right traversals, i.e. $s = s^1 \ldots s^m$ where $s^i$ is a left-to-right traversal of $t_{(n)}$ for $1 \leq i \leq m$.

Similar definitions can easily be obtained for the m-alternating pass (originally introduced by [5]) and the m-sweep properties. We define for instance a sweep over $t_{(n)}$ as a walk s through $t_{(n)}$ where

- $s = (n, A_I)s_{i_1} \ldots s_{i_k} (n, A_S)$ for some $A_I$ and $A_S$ and some permutation $(i_1, \ldots, i_k)$ of $(1, \ldots, k)$

- $s_{i_j}$ is a sweep over $t_{(n\S j)}$ for $1 \leq j \leq k$.

In contrary to the m-pass, m-alternating pass and m-sweep properties the m-visit property does not put restrictions on the way the derivation tree is traversed. It only requires that each node is visited at most m times (as the other three properties also do). So an m-visit computation sequence for $t_{(n)}$ is a computation sequence s for $t_{(n)}$ where

- $|s(n)| \leq 2m$

- $s(t_{(n\S j)})$ is an m-visit computation sequence for $t_{(n\S j)}$ for $1 \leq j \leq k$.

These definitions are straightforward formalizations of those given in [4].

As mentioned above, Engelfriet and Filè ([4]) combine these four properties with the well-defined and ordered properties of AGs. For instance the subclass of pure m-sweep AGs is defined by requiring that each derivation tree has an m-sweep computation sequence, and the subclass of simple m-pass AGs is defined by requiring that there exists an assignment A of partitions to the symbols such that each derivation tree has an m-pass computation sequence that is ordered with respect to A.

In the same style we can define subclasses of absolutely non-circular AGs called uniform m-visit AGs, uniform m-sweep AGs, uniform m-alternating pass AGs and uniform m-pass AGs. For instance, an AG is uniform m-sweep if

> there exists a top-down assignment A of partitions to the symbols of $V_N$ and a partition $\pi_0$ of the attributes of S such that each derivation tree with root labelled S has an m-sweep computation sequence that is $\pi_0$-uniform with respect to A.

We believe that the new subclasses fit nicely into the results of [4].

Acknowledgement

I wish to thank Joost Engelfriet for his helpful comments, especially his suggestion of theorem 2. Brian Mayoh and Flemming Nielson commented on a previous version.

References

1. Aho, A.V. and Ullman, J.D.: The Theory of Parsing. Translation and Compiling, Volume I: Parsing. Prentice-Hall Inc. (1972).

2. Bochmann, G.V.: Semantic evaluation from left to right, CACM 19 (1976), 55-62.

3. Engelfriet, J. and Filè, G.: Simple multi-visit attribute grammars, JCSS 24 (1982), 283-314.

4. Engelfriet, J. and Filè, G.: Passes, sweeps and visits. In: Automata, Languages and Programming (G. Goos and J. Hartmanis, Eds.), Lecture Notes in Computer Science, Vol. 115, 193-207. Springer Verlag (1981).

5. Jazayeri, M. and Walter, K.G.: Alternating semantic evaluator, Proc. ACM 1975 Annual Conf. (1975).

6. Kastens, U.: Ordered attribute grammars, Acta Inf. 13 (1980), 229-256.

7. Kennedy, K. and Warren, S.K.: Automatic generation of efficient evaluators for attribute grammars, Conf. Record of the Third ACM Symp. on Principles of Programming Languages (1976), 32-49.

8. Knuth, D.E.: Semantics of context free languages, Math. Syst. Theory 2 (1968) 127-145.

9. Knuth, D.E.: Semantics of context free languages: correction, Math. Syst. Theory 5 (1971) 95-96.

10. Nielson, H.R.: Using Computation sequences to define evaluators for attribute grammars, DAIMI PB-139, Aarhus University, Denmark (1981).

11. Riis, H. and Skyum, S.: k-visit attribute grammars, to appear in Math. Syst. Theory 15(2)(1981), 17-28.