ISSN 0105-8517

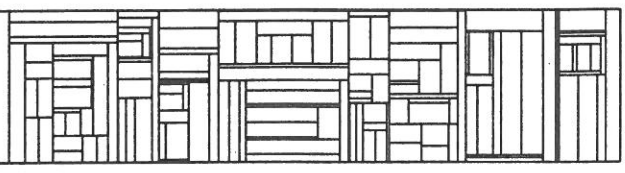
Using computation sequences to define evaluators for attribute grammars

by

Hanne Riis Nielson

DAIMI PB-139
November 1981

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



Using computation sequences to define evaluators for attribute grammars

Abstract

An evaluator for an attribute grammar takes a derivation tree as input and produces a computation sequence for it as output. We give a simple but general construction of an evaluator for any well-defined attribute grammar and we prove its correctness. Evaluators for several subclasses of attribute grammars can be constructed by changing a preprocessing stage of the evaluator. As an example we consider the absolutely well-defined attribute grammars.

Keywords

well-defined attribute grammars, absolutely well-defined attribute grammars, evaluators, computation sequences, proof of correctness.

1. INTRODUCTION

Attribute grammars are introduced by Knuth as a tool for associating meanings with strings of context free languages ([9]). An attribute grammar is an extension of a context free grammar: each symbol has associated with it a fixed number of attributes and each production has associated with it a set of semantic rules defining some of the attributes of the symbols in terms of others. To each node of a derivation tree we associate attributes corresponding to those of the symbol labelling it. The semantic rules of the productions are used to evaluate the attributes of the nodes. The meaning of a string is obtained by first constructing a derivation tree for it, then evaluating the attributes of the tree and finally, taking the tree decorated with attribute-values as the meaning. We will consider a part of this process, namely that of determining an order in which to evaluate all the attributes of a derivation tree without violating their dependencies. Such an ordering is called a computation sequence ([10], [11]).

A device that takes a derivation tree as input and produces a computation sequence as output will be called an evaluator. An evaluator can be thought of as a recursive routine taking a node of the derivation tree as parameter - we say that it visits the node. At a node the evaluator can perform two kinds of actions: it can append a set of attributes to the computation sequence it constructs or it can call itself with one of the sons of the node as parameter. The evaluator can perform any sequence of actions before it returns. Examples of evaluators that obviously follow this scheme are given by [2], [3], [6], [7], [8] and [12].

In this paper we present a simple but general evaluator. We show how to construct an evaluator for any well-defined AG and we prove its correctness. We compare our evaluator with others, especially that of [8]. We claim that our evaluator is simpler in that evaluators for several subclasses of attribute grammars can be constructed by just modifying a preprocessing stage. The nature of this preprocessing stage is intimately connected with one way of characterising subclasses of attribute grammars ([10]).

2. PRELIMINARIES

This section contains a review of Knuth's original definition of an attribute grammar ([9]) together with the definition of a computation sequence (adapted from [10]).

An attribute grammar (abbreviated AG) is an extension of a context free grammar $G = (V_N, V_T, P, S)$. 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 α takes values in a set D_α. Each production $p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k$ ($F_j \in V_N$, $v_j \in V_T^*$ for $0 \le j \le k$) of P has associated a set of semantic functions. For each α in $\mathcal{S}(F_0)$ (resp. in $\mathcal{I}(F_j)$, $1 \le j \le k$) there is a (semantic) function $f_{0\alpha}$ (resp. $f_{j\alpha}$) of functionality $D_{\alpha_1} \times \cdots \times D_{\alpha_m} \to D_\alpha$ (m and α_i depend on α and j). Each α_i is an attribute of either $\mathcal{I}(F_0)$ or of $\mathcal{S}(F_\ell)$ for some ℓ, $1 \le \ell \le k$.

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 where the production $p: F_0 \to v_0F_1v_1 \cdots \cdots v_{k-1}F_kv_k$ is applied. For each α in $\mathcal{S}(F_0)$ the function $f_{0\alpha}: D_{\alpha_1} \times \cdots \times D_{\alpha_m} \to D_\alpha$ associated with p can be used to determine the value of α at n when the values of all the attributes $\alpha_1, \ldots, \alpha_m$ have been determined. Similarly, for α in $\mathcal{I}(F_j)$ ($1 \le j \le k$) the function $f_{j\alpha}$ associated with p is used to determine the value of α 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 then the meaning of t will be t decorated with these values.

The semantic functions associated with the production $p: F_0 \to v_0F_1v_1 \cdots \cdots v_{k-1}F_kv_k$ define a dependency graph $D(p)$ for p. $D(p)$ has one node $[j.\alpha]$ for each attribute α of $\mathcal{I}(F_j) \cup \mathcal{S}(F_j)$, $0 \le j \le k$. If $f_{j\alpha}: D_{\alpha_1} \times \cdots \times D_{\alpha_m} \to D_\alpha$ is a semantic function associated with p and α_i is an attribute of F_{j_i} ($1 \le i \le m$) then there will be an arc from $[j_i.\alpha_i]$ to $[j.\alpha]$ in $D(p)$. These are the only arcs of $D(p)$.

A partition of the attributes of a symbol F ([5]) 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 Π_F be the set of partitions for the symbol F. For the production $p: F_0 \rightarrow v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ consider any sequence

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

where $0 \leq h_i \leq k$ and $A_i \subseteq \mathcal{I}(F_{h_i}) \cup \mathcal{S}(F_{h_i})$ for $1 \leq i \leq r$. For $0 \leq j \leq k$ define

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

where $i_1 < \ldots < i_m$ and $\{\ell \mid h_\ell = j,\ 1 \leq \ell \leq r\} = \{i_1, \ldots, i_m\}$. The sequence π_p is a partition of the attributes of p if for $0 \leq j \leq k$ $\pi_p(j)$ is a partition of the attributes of F_j. π_p satisfies the dependency graph $D(p)$ for p if for all ℓ, $1 \leq \ell \leq r$, and for all α in A_ℓ

> if there is an arc from $[i.\beta]$ to $[h_\ell.\alpha]$ in $D(p)$
> then for some ℓ', $\ell' < \ell$, $\beta \in A_{\ell'}$ and $i = h_{\ell'}$

Let $\Pi_{p,\pi}$ be the set of partitions of attributes of p satisfying $D(p)$ and with $\pi_p(0) = \pi$.

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 of 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 λ and the location of the j'th son of a node with location n is

$n\S j$. We will not distinguish between a node and its location. The subtree of t whose root is n is denoted $t_{(n)}$. If the production $p: F_0 \to v_0F_1v_1 \ldots \ldots v_{k-1}F_kv_k$ is applied at n, we will write $t_{(n)} = F_0[t_{(n\S 1)} \ldots t_{(n\S k)}]$. The part of t with $t_{(n)}$ removed except n itself is denoted $t^{(n)}$. DT(G) is the set of derivation trees for G.

Let $p: F_0 \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$ be the production applied at a node n in a derivation tree t. A <u>walk s through $t_{(n)}$</u> is defined recursively by

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

Consider a node n' in $t_{(n)}$. 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 $t_{(n')}$. Similarly, $s(t^{(n')})$ is the walk through $t_{(n)}$ obtained by removing all pairs (n'', A'') from s where $n'' \neq n'$ and n'' is a node in $t_{(n')}$. Let $p': F'_0 \to v'_0F'_1v'_1 \ldots \ldots v'_{k-1}F'_{k'}v'_{k'}$ 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 1) removing all pairs (n'', A'') from s where $n'' \neq n'$ and $n'' \neq n'\S j$ $(1 \le j \le k')$ and 2) replacing each of the remaining pairs (n', A) by (0, A) and each of the pairs $n'\S j, A)$ by (j, A), $1 \le j \le k$. Furthermore define <u>s(n')</u> as s(n', p')(0).

The walk s through $t_{(n)}$ is a <u>computation sequence</u> 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 $1 \le j \le k$.

The set of walks through complete derivation trees of G is denoted W(G). Finally, an AG is <u>well-defined</u> if each derivation tree has a computation sequence ([11]). In this paper we shall only consider well-defined AGs.

We close this section by presenting an AG that will be used in examples throughout this paper. The underlying grammar has two non-terminals A and S with attributes:

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

Each attribute has the set of integers associated. The productions of the underlying grammar are listed below together with the associated semantic functions.

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

3. THE EVALUATOR

In this section we will motivate our definition of an evaluator and compare it with other evaluators, primarily that of [8].

As mentioned we think of an evaluator as a recursive routine that takes a node of a derivation tree as parameter. When visiting a node the evaluator will perform a sequence of actions before returning. The main difference between the evaluators in the literature is how they determine which sequence of actions to perform. In the approach taken by Kennedy and Warren ([8]) the sequence of actions is determined from two types of information. There will be a flag (called a quiescent state by [8]) at each node telling essentially what has happened at the previous visits to the node (if any). The second information will be given by an extra parameter of the evaluation routine. It is a state that summarizes what has happened since the last visit to the node. The state is called an input-set in [8]; our terminology is motivated in the tree-automata theory (see e.g. [4]). From the flag at the node and the current state the evaluator chooses one of a fixed set of sequences of actions and determines which flag to set at the node when returning. Briefly, the evaluator will consist of a set Σ of flags, a set Q of states and two tables GOTO and PLAN. Given a flag and a state the GOTO-table determines an index (called an entry state in [8]) which is an element of a set I . Using this index, the PLAN-table gives the sequence of actions to be performed and the new flag. Finally, there is a relabelling r determining the initial flags of the nodes of each derivation tree and there is an initial state q_0. For a production p: $F_0 \rightarrow v_0F_1v_1 \cdots v_{k-1}F_kv_k$ an action is either an entity $[j.\alpha]$ where α is an attribute of F_j $(0 \leq j \leq k)$ or it is a visit instruction VISIT(j, q) where q is a state $(1 \leq j \leq k)$. The evaluator for the example AG is in table 1.

Table 1. The Kennedy & Warren evaluator for the example AG

$\Sigma = \{\sigma_j \mid 1 \leq j \leq 14\}$

r : a relabelling determined by

if $t = S[t_{(1)}t_{(2)}]$ then $r(t) = (S,\sigma_1)[r(t_{(1)})\ r(t_{(2)})]$

if $t = A[t_{(1)}]$ then $r(t) = (A,\sigma_3)[r(t_{(1)})]$

if $t = A[\ ]$, p_3: A → b is applied then $r(t) = (A,\sigma_7)[\ ]$

if $t = A[\ ]$, p_4: A → c is applied then $r(t) = (A,\sigma_{11})[\ ]$

$I = \{i_j \mid 1 \leq j \leq 13\}$

$Q = \{q_j \mid 0 \leq j \leq 3\}$

q_0 is initial

GOTO: $\Sigma \times Q \rightarrow I$ (a partial mapping)

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}
q_0	i_1	-	-	-	-	-	-	-	-	-	-	-	-	-
q_1	-	-	i_2	-	-	-	i_6	-	-	-	i_{10}	-	-	-
q_2	-	-	i_3	-	-	-	i_7	-	-	-	i_{11}	-	-	-
q_3	-	-	-	i_4	i_5	-	-	i_8	i_9	-	-	i_{12}	i_{13}	-

PLAN : $I \rightarrow \{\text{sequence of actions}\} \times \Sigma$, (a total mapping)

	sequence of actions	Σ
i_1	$[0.-]\ [1.\beta]$ VISIT$(1,q_2)$ $[2.\alpha]$ VISIT$(2,q_1)[2.\beta]$ VISIT$(2,q_3)$ $[1.\alpha]$ VISIT$(1,q_3)$ $[0.\epsilon]$	σ_2
i_2	$[1.\alpha]$ VISIT$(1,q_2)$ $[0.\gamma]$	σ_4
i_3	$[1.\beta]$ VISIT$(1,q_2)$ $[0.\delta]$	σ_5
i_4	$[1.\beta]$ VISIT$(1,q_3)$ $[0.\delta]$	σ_6
i_5	$[1.\alpha]$ VISIT$(1,q_3)$ $[0.\gamma]$	σ_6
i_6	$[0.\gamma]$	σ_8
i_7	$[0.\delta]$	σ_9
i_8	$[0.\delta]$	σ_{10}
i_9	$[0.\gamma]$	σ_{10}
i_{10}	$[0.\gamma]$	σ_{12}
i_{11}	$[0.\delta]$	σ_{13}
i_{12}	$[0.\delta]$	σ_{14}
i_{13}	$[0.\gamma]$	σ_{14}

□

The evaluator behaves as follows when applied to a derivation tree t:

1. Each node in t is flagged by its initial flag as determined by r;
2. the algorithm EVALUATE(λ, q_0, s) below is called; s will be a computation sequence for t.

Algorithm EVALUATE(n, q, s); call-by-value: n, q; call-by-result: s

1. let σ be the flag of the node n, and let $s = \lambda$;
   let i = GOTO(σ, q) and let PLAN(i) = ($a_1 \ldots a_m, \sigma'$)
2. for ℓ = 1 to m do
   if $a_\ell = [0.\alpha]$ then append $(n, \{\alpha\})$ to s;
   if $a_\ell = [j.\alpha]$, $j \neq 0$ then append $(n\S j, \{\alpha\})$ to s;
   if a_ℓ = VISIT(j, q') then call EVALUATE($n\S j, q', s'$) and append s' to s
3. let σ' be the flag of n

The main difference between the approach of [8] sketched above and ours is that we do not allow the evaluator to change the flags at the nodes. All information needed in order to choose the wanted sequence of actions to be performed must be given by the (initial) flag of the node and the current state. Briefly, our evaluator will consist of a set Σ of flags, a set Q of states and a <u>transition mapping</u> δ that given a flag and a state determines a sequence of actions (for a production). Furthermore there is a relabelling r putting flags on the nodes of the derivation trees and there is an initial state q_0. The evaluator behaves almost as that of [8], the main difference being that the flag is not changed.

In our formal definition we will make a slight change in the form of a sequence of actions, Let p: $F_0 \rightarrow v_0F_1v_1 \cdots v_{k-1}F_kv_k$ be a production. A <u>sequence of actions</u> for p has the form

$$(A_I, (j_1, q_1) \cdots (j_m, q_m), A_S)$$

where $A_I \subseteq \mathcal{I}(F_0)$, $A_S \subseteq \mathcal{S}(F_0)$ and (j_i, q_i) is an abbreviation for $VISIT(j_i, q_i)$. This means that when visiting a node we will first "evaluate" some inherited attributes, next we will visit some of the sons of the node and finally we will "evaluate" some synthesized attributes. In practice other interpretations of a visit may be useful, however we believe that most of them can be simulated by our interpretation. Formally we define

An <u>evaluator</u> for an AG with underlying context free grammar $G = (V_N, V_T, P, S)$ is a tuple

$$M = (Q, \Sigma, \delta, \hat{q}_0, r)$$

where

- Q is a finite set of states
- Σ is an alphabet of flags
- δ is a family of transition mappings $\{\delta_p\}_{p \in P}$

  For $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$, δ_p is a partial mapping of functionality

  $$\delta_p : Q \times \Sigma \to \mathcal{P}(\mathcal{I}(F_0)) \times ([1,k] \times Q)^* \times \mathcal{P}(\mathcal{S}(F_0))$$

  where $[1,k] = \{j \mid 1 \leq j \leq k\}$ and $\mathcal{P}(A)$ is the set of subsets of the set A.

- $\hat{q}_0$ is a finite nonempty sequence of initial states (an element) of Q^*
- r is a relabelling

  $r : DT(G) \to DT_\Sigma(G)$

  where $DT_\Sigma(G)$ is the set of derivation trees for G except that the nodes are labelled by elements of $V_N \times \Sigma$ instead of V_N.

We have a sequence of initial states because we want to allow for the possibility of several visits to the root of the derivation tree. We now formalize the algorithm EVALUATE considered above by defining the behaviour mapping $\hat{\delta}$:

The behaviour of the evaluator M is defined by the mapping

$$\hat{\delta} : Q \times DT(G) \rightarrow W(G)$$

Let t be a derivation tree and consider a node n of t where the production $p: F_0 \rightarrow v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is applied. Let (F_0, σ) be the label of the node n in $r(t)$. Then for $q \in Q$

$$\hat{\delta}(q, t_{(n)}) = s$$

if and only if

- $\delta_p(q,\sigma) = (A_1, (j_1, q_1) \cdots (j_m, q_m), A_S)$
- $\hat{\delta}(q_j, t_{(n\S j_i)}) = s_i$ for $1 \leq i \leq m$
- $s = (n, A_1)s_1 \cdots s_m(n, A_S)$

The evaluator M defines a translation from derivation trees of G to walks through derivation trees:

$$\mathfrak{T}(M) = \{(t, s) \mid s = \hat{\delta}(q_1, t) \cdots \hat{\delta}(q_m, t),\ \hat{q}_0 = q_1 \cdots q_m \text{ and } t \in DT(G) \text{ has root labelled } S\}$$

The evaluator M is a correct evaluator for the AG if for all t in DT(G) with root labelled S there is a computation sequence s such that $(t, s) \in \mathfrak{T}(M)$.

In table 2 we have given our evaluator for the example AG.

Table 2. The evaluator M for the example AG

$$M = (Q, \Sigma, \delta, \hat{q}_0, r)$$

where

$$Q = \{q_1, q_2\}$$

$$\Sigma = \{\sigma_i \mid 0 \leq i \leq 4\}$$

δ :
$$\delta_{p_1}(q_1, \sigma_0) = (\emptyset\ ,\ (1,q_1)(2,q_1)(2,q_2)(1,q_2),\ \{\epsilon\})$$
$$\delta_{p_2}(q_1, \sigma_1) = (\{\alpha\}\ ,\ (1,q_1),\ \{\gamma\})$$
$$\delta_{p_2}(q_2, \sigma_1) = (\{\beta\}\ ,\ (1,q_2),\ \{\delta\})$$
$$\delta_{p_2}(q_1, \sigma_2) = (\{\beta\}\ ,\ (1,q_1),\ \{\delta\})$$
$$\delta_{p_2}(q_2, \sigma_2) = (\{\alpha\}\ ,\ (1,q_2),\ \{\gamma\})$$
$$\delta_{p_3}(q_1, \sigma_3) = (\{\alpha\}\ ,\ \lambda\ ,\ \{\gamma\})$$
$$\delta_{p_3}(q_2, \sigma_3) = (\{\beta\}\ ,\ \lambda\ ,\ \{\delta\})$$
$$\delta_{p_4}(q_1, \sigma_4) = (\{\beta\}\ ,\ \lambda,\ \{\delta\})$$
$$\delta_{p_4}(q_2, \sigma_4) = (\{\alpha\}\ ,\ \lambda,\ \{\gamma\})$$

$$\hat{q}_0 = q_1$$

r a relabelling defined by

if $t = S[t_{(1)}t_{(2)}]$ then $r(t) = (S,\sigma_0)[r''(t_{(1)})\ r'(t_{(2)})]$

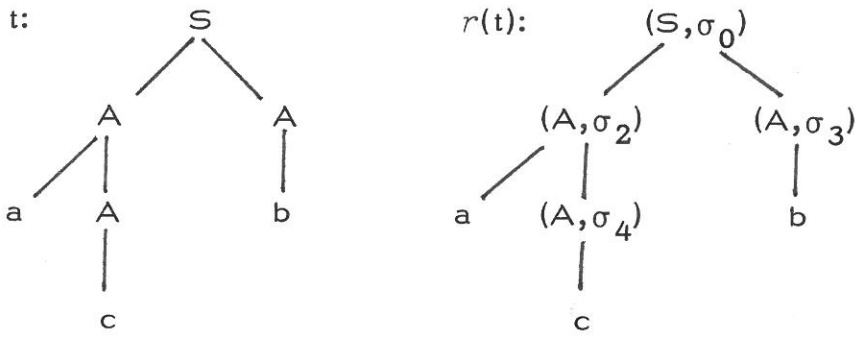
if $t = A[t_{(1)}]$ then $r'(t) = (A,\sigma_1)[r'(t_{(1)})]$ and $r''(t) = (A,\sigma_2)[r''(t_{(2)})]$

if $t = A[\ ]$ then $r'(t) = (A,\sigma_3)[\ ]$ and $r''(t) = (A,\sigma_4)[\ ]$

□

Example

Consider the following derivation tree t of the example AG:



The evaluator of table 2 constructs the following computation sequence s for t:

$$s = (\lambda,\emptyset)\ (1,\{\beta\})(1\S1,\{\beta\})(1\S1,\{\delta\})(1,\{\delta\})(2,\{\alpha\})(2,\{\gamma\})$$
$$(2,\{\beta\})(2,\{\delta\})(1,\{\alpha\})(1\S1,\{\alpha\})(1\S1,\{\gamma\})(1,\{\gamma\})(0,\{\epsilon\})$$

□

The basic idea of the evaluators of [3] and [12] is the same as that of [8]: To each node of the derivation tree there is associated a flag that is changed during the evaluation process. The evaluators for the ordered AGs defined by [7] do not associate flags with the nodes at all because of the restrictions imposed on the AGs. The same holds for the pass-oriented evaluators of [2] and [6].

4. CONSTRUCTION OF THE EVALUATOR

In this section we show how to construct an evaluator for a well-defined AG and we prove its correctness.

Let us start by analyzing the behaviour of a correct evaluator $M = (Q, \Sigma, \delta, \hat{q}_0, r)$ when applied to a derivation tree t. Let $(t,s) \in \mathfrak{T}(M)$ and consider a node n in t where the production $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ is applied. Suppose that $s(n) = A_1 \ldots A_{2m}$ so that the node n has been visited m times. At the ℓ'th visit $(n, A_{2\ell-1})$ and $(n, A_{2\ell})$ have been added to s. If the label of n in $r(t)$ is (F_0, σ) and the state at the ℓ'th visit is q then we have

$$\delta_p(q,\sigma) = (A_{2\ell-1}, v_\ell, A_{2\ell})$$

for some v_ℓ. We can arrange that v_ℓ can be determined from $s(n,p)$. Define the sequences $u_1, \ldots, u_m$ by

$$s(n,p) = (0,A_1)u_1(0,A_2) \ldots (0,A_{2m-1})u_m(0,A_{2m})$$

Let $s(n\S j) = A_1^j \ldots A_{2m}^j$ for $1 \le j \le k$. Then u_ℓ is a sequence of pairs where each pair has the form $(j,A_{2i-1}^j)(j,A_{2i}^j)$ and specifies the i'th visit to the j'th son. Let $\hat{u}_\ell$ be the sequence obtained from u_ℓ by replacing each pair $(j,A_{2i-1}^j)(j,A_{2i}^j)$ by (j,i). Since $\hat{u}_\ell$ mentions the numbers of the visits to the sons of n it can be used for v_ℓ in the definition of $\delta_p(q,\sigma)$ provided we let a state denote the number of a visit. Thus

$$\delta_p(q,\sigma) = (A_{2q-1}, \hat{u}_q, A_{2q})$$

By assuming that $\sigma = s(n,p)$ we can determine $\delta_p(q,\sigma)$ directly from σ and q.

This analysis motivates the following decisions when constructing the evaluator M: The states (elements of Q) are numbers of visits and the flags (elements of Σ) are partitions of attributes of productions. The transition mapping δ is defined as follows. Let $\pi_p \in \Sigma$ be a partition of

the attributes of a production $p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k$ and let $q \in Q$ be such that $1 \leq q \leq m$ where $\pi_p(0) = A_1 \ldots A_{2m}$. Define $u_1, \ldots, u_m$ by $\pi_p = (0,A_1)u_1(0,A_2) \ldots (0,A_{2m-1})u_m(0,A_{2m})$ and let $\hat{u}_\ell$ be obtained by replacing $(j,A^j_{2i-1})(j,A^j_{2i})$ by (j,i) (here $\pi_p(j) = A^j_1 \ldots A^j_{2m}$ for $1 \leq j \leq k$). Finally, define $\delta_p(q,\pi_p) = (A_{2q-1}, \hat{u}_q, A_{2q})$. The relabelling r to be constructed will fulfill that if (F_0,π_p) is the label of some node n in $r(t)$ where $p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is applied at n in t then the label $(F_j,\pi_{p'})$ of n§j in $r(t)$ will have $\pi_p(j) = \pi_{p'}(0)$. Furthermore π_p will satisfy D(p).

The rest of this section consists of three parts. First we show how to put restrictions on the sets Σ and Q to make them finite. Next we construct the relabelling r using some additional notation and finally we prove the correctness of the evaluator.

In general a production (and a symbol) has an infinite number of partitions of its attributes. We will here restrict ourselves to reduced partitions. A partition $\pi = A_1 \ldots A_{2m}$ of the attributes of the symbol F is a <u>reduced partition</u> if for all i, $1 \leq i \leq m$, $A_{2i-1} = A_{2i} = \emptyset$ implies $m = 1$. A partition π_p of the attributes of a production $p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is reduced if $\pi_p(j)$ is a reduced partition of the attributes of F_j for $0 \leq j \leq k$. It is not a severe restriction just to consider reduced partitions since the transformations on computation sequences given by [11] easily can be modified to construct a reduced partition of the attributes of a production from an unrestricted one. It is easy to see that a reduced partition has a maximal length and thereby that there is a finite number of reduced partitions. We can now give a formal definition of the alphabet Σ of flags:

$$\Sigma = \{\pi_p \mid p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k \text{ is in P and } \pi_p \text{ is a reduced partition of the attributes of p}\}$$

Given Σ we can define the set Q of states. Let

$$mQ = \max\{m \mid \pi_p \in \Sigma \text{ and } \pi_p(0) = A_1 \cdots A_{2m}\}$$

and define

$$Q = \{ q \mid 1 \leq q \leq mQ \}$$

The sequence of initial states $\hat{q}_0$ will be defined in connection with the relabelling r.

In order to specify the relabelling we need some notation and definitions concerning dependency graphs. A dependency graph for a symbol F has one node denoted $[\alpha]$ for each attribute α of $\mathcal{I}(F) \cup \mathcal{S}(F)$ and maybe some arcs. Given a production $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ and dependency graphs $\Gamma_1, \ldots, \Gamma_k$ for $F_1, \ldots, F_k$ we can construct new dependency graphs for both p and F_0. The dependency graph

$$D(p)\,[\![\Gamma_1 \cdots \Gamma_k]\!]$$

for p will have an arc from $[j.\alpha]$ to $[i.\beta]$ if and only if there is an arc from $[j.\alpha]$ to $[i.\beta]$ in $D(p)$ or if $i = j$ and there is an arc from $[\alpha]$ to $[\beta]$ in Γ_j. The dependency graph Γ_0 for F_0 derived from $D(p)[\![\Gamma_1 \cdots \Gamma_k]\!]$ has an arc from $[\alpha]$ to $[\beta]$ if and only if there is a non-empty path from $[0.\alpha]$ to $[0.\beta]$ in $D(p)[\![\Gamma_1 \cdots \Gamma_k]\!]$. For each node n of a derivation tree we define a dependency graph $sym(t_{(n)})$ for the symbol labelling it. Let $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ be the production applied at n. Then $sym(t_{(n)})$ is the dependency graph for F_0 derived from $D(p)[\![sym(t_{(n\S 1)}) \cdots sym(t_{(n\S k)})]\!]$. The set $SYM(F)$ of dependency graphs for F is defined by

$$SYM(F) = \{ sym(t_{(n)}) \mid t \text{ is a derivation tree with a node } n \text{ labelled } F \}$$

Finally, a partition $\pi = A_1 \cdots A_{2m}$ of the attributes of F satisfies a dependency graph Γ for F if for $1 \leq \ell$ and $\alpha \in A_\ell$

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

We now turn to the definition of the relabelling r. Consider a derivation tree t and let $p: F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ be the production applied at n.

The relabelling τ that we are going to construct will fulfill:

a) if (F_0, π_p) is the label of n in $\tau(t)$ then π_p satisfies $D(p)[\![sym(t_{(n\S 1)}) \ldots sym(t_{(n\S k)})]\!]$

b) if $(F_j, \pi_{p'})$ is the label of $n\S j$ in $\tau(t)$ then $\pi_{p'}(0) = \pi_p(j)$ $(1 \leq j \leq k)$.

Later we will see that a) and b) are enough to ensure correctness of the evaluator. The relabelling τ will operate in a top-down manner. Given a partition π of the attributes of F_0 satisfying $sym(t_{(n)})$ we will construct the flag π_p at the node n in $\tau(t)$. π_p determines for each j, $1 \leq j \leq k$, a partition $\pi_p(j)$ of the attributes of F_j satisfying $sym(t_{(n\S j)})$. Thus τ can be defined by a set of rules of the form

$$R(\pi, t_{(n)}) = (F_0, \pi_p)[R(\pi_p(1), t_{(n\S 1)}) \ldots R(\pi_p(k), t_{(n\S k)})]$$

The following lemma shows that π_p exists.

Lemma 1

Consider a production $p: F_0 \to v_0 F_1 v_1 \ldots v_{k-1} F_k v_k$ and let $\Gamma_j \in SYM(F_j)$ for $1 \leq j \leq k$. If π is a reduced partition of the attributes of F_0 satisfying the dependency graph derived from $D(p)[\![\Gamma_1 \ldots \Gamma_k]\!]$ then there exists a reduced partition π_p of the attributes of p satisfying $D(p)[\![\Gamma_1 \ldots \Gamma_k]\!]$ and with $\pi_p(0) = \pi$.

Proof Let $\pi = A_1 \ldots A_{2m}$ and $\Gamma = D(p)[\![\Gamma_1 \ldots \Gamma_k]\!]$. Define for $B \subseteq \{[j.\alpha] \mid \alpha \in \mathcal{I}(F_j) \cup \mathcal{S}(F_j),\ 0 \leq j \leq k\}$

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

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

The following algorithm taken from [10] constructs a partition π_p with the wanted properties:

```
B := ∅;                    π_p := λ ;
FOR i = 1 TO m DO
BEGIN B := B ∪ {[0.α] | α ∈ A_{2i-1}} ; π_p := π_p(0, A_{2i-1});
  IF i = 1 THEN
  BEGIN FOR each j with ϑ(F_j) = ∅ or NEW-syn(j,∅,Γ) ≠ ∅, 1≤j≤k DO
    BEGIN B_S := {α | [j.α] ∈ NEW-syn(j,∅,Γ)} ;
      B:= B ∪ NEW-syn(j, ∅,Γ); π_p := π_p(j,∅)(j,B_S)
    END;
  END;
  WHILE there is a j with NEW-inh(j,B,Γ) ≠ ∅, 1 ≤ j ≤ k DO
  BEGIN determine the least j with NEW-inh(j,B,Γ) ≠ ∅, 1≤ j ≤ k
    B_I := {α | [j.α] ∈ NEW-inh(j,B,Γ)} ; B := B ∪ NEW-inh(j,B,Γ);
    B_S := {α | [j.α] ∈ NEW-syn(j,B,Γ)} ; B := B ∪ NEW-syn(j,B,Γ);
    π_p := π_p(j,B_I)(j,B_S);
  END;
  B := B ∪ {[0.α] | α ∈ A_{2i}} ; π_p := π_p(0,A_{2i});
END
```

□

The relabelling r will be defined by a mapping R: $\{\pi \mid \pi \in \Pi_F$ for some $F\}$ $\times$ DT(G) $\rightarrow$ $DT_\Sigma(G)$ as follows: if the production p: $F_0 \rightarrow v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is applied at the root of a derivation tree t and π is a reduced partition of the attributes of F_0 satisfying sym(t) then

$$R(\pi, t) = (F_0, \pi_p)[R(\pi_p(1), t_{(1)}) \cdots R(\pi_p(k), t_{(k)})]$$

where π_p is the partition for p given by lemma 1. We choose the initial partition of the attributes of the start symbol as simple as possible: $\emptyset\, \vartheta(S)$. If t is a derivation tree with root labelled S then

$$r(t) = R(\emptyset\, \vartheta(S), t)$$

We now see that the root of t will be visited exactly once so we define the initial sequence of states as

$$\hat{q}_0 = 1$$

From the definition above and lemma 1 we immediately have

Lemma 2

The relabelling r constructed above fulfill that for any derivation tree t where the production p: $F_0 \to v_0 F_1 v_1 \cdots v_{k-1} F_k v_k$ is applied at a node n we have

a) if (F_0, π_p) is the label of n in $r(t)$ then π_p satisfies $D(p)[\![sym(t_{(n\S1)}) \cdots sym(t_{(n\S k)})]\!]$

b) if $(F_j, \pi_{p'})$ is the label of $n\S j$ in $r(t)$ then $\pi_{p'}(0) = \pi_p(j)$ $(1 \leq j \leq k)$.

□

This completes the construction of r and thereby the evaluator M. The construction is summarized below:

Construction of the evaluator $M = (Q, \Sigma, \delta, \hat{q}_0, r)$

$Q = \{j \mid 1 \leq j \leq mQ\}$

where $mQ = \max\{m \mid \pi_p \in \Sigma \text{ and } \pi_p(0) = A_1 \ldots A_{2m}\}$

$\Sigma = \{\pi_p \mid p: F_0 \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$ is a production and π_p is a reduced partition of the attributes of $p\}$

δ : for $p: F_0 \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$, $\pi_p \in \Sigma$ a partition of the attributes of p with $\pi_p(0) = A_1 \ldots A_{2m}$ and $1 \leq q \leq m$ let

$$\delta_p(q, \pi_p) = (A_{2q-1}, \hat{u}_q, A_{2q})$$

where $\hat{u}_q$ is defined from u_q where

$$\pi_p = (0, A_1)u_1(0, A_2)\ldots(0, A_{2m-1})u_m(0, A_{2m})$$

by replacing pairs $(j, A^j_{2i-1})(j, A^j_{2i})$ by

$$(j, i)\ (\pi_p(j) = A^j_1 \ldots A^j_{2m_j}).$$

For $m < q \leq mQ$ $\delta_p(q, \pi_p)$ is undefined

$\hat{q}_0 = 1$

r : $r(t) = R(\emptyset\mathbb{S}(S), t)$ where

$R(\pi, t) = (F_0, \pi_p)[R(\pi_p(1), t_{(1)}) \ldots R(\pi_p(k), t_{(k)})]$

and π_p is given by (the algorithm of) lemma 1 applied to π and $D(p)[\![\text{sym}(t_{(1)} \ldots \text{sym}(t_{(k)})]\!]$

We now turn to a proof of the correctness of M:

Theorem 1

For each well-defined AG there exists a correct evaluator.

Proof We will prove that the evaluator M constructed above is correct, i.e. each derivation tree t with root labelled S has a computation sequence s such that $(t, s) \in \mathfrak{T}(M)$. Let $p: F_0 \to v_0F_1v_1 \ldots v_{k-1}F_kv_k$ be the production applied at a node n in t and let (F_0, π_p) be the label of n in $r(t)$ with $\pi_p(j) = A^j_1 \ldots A^j_{2m_j}$ for $0 \leq j \leq k$. By structural induction we will show

(*) for $0 \leq q \leq m_0$ there exists a walk s'_q through $t_{(n)}$ such that $s = \hat{\delta}(q, t_{(n)}) \ldots \hat{\delta}(q, t_{(n)})s'_q$ is a computation sequence for $t_{(n)}$ with $s(n, p) = \pi_p$.

At the basis we have $p: F_0 \to v_0$ and $\pi_p = (0, A^0_1) \ldots (0, A^0_{2m_0})$. Since $\pi_p \in \Sigma$ we have $\delta_p(q, \pi_p) = (A^0_{2q-1}, \lambda, A^0_{2q})$ for $1 \leq q \leq m_0$ and thereby

$$\hat{\delta}(1, t_{(n)}) \ldots \hat{\delta}(q, t_{(n)}) = (n, A^0_1) \ldots (n, A^0_{2q})$$

By letting $s'_q = (n, A^0_{2q+1}) \ldots (n, A^0_{2m_0})$ it is easy to see that (*) holds.

For the induction step let $\pi_p = (0, A^0_1)u_1(0, A^0_2) \ldots (0, A^0_{2m_0-1})u_{m_0}(0, A^0_{2m_0})$. As before let $\hat{u}_\ell$ be obtained from u_ℓ by replacing each pair $(j, A^j_{2i-1})(j, A^j_{2i})$ by (j, i). Let $\hat{u}_\ell = (i_{1\ell}, h_{1\ell}) \ldots (i_{x_\ell \ell}, h_{x_\ell \ell})$. From the definition of δ and $\hat{\delta}$ we get for $1 \leq \ell \leq m_0$:

$$\hat{\delta}(\ell, t_{(n)}) = (n, A^0_{2\ell-1})\, \hat{\delta}(h_{1\ell}, t_{(n\S i_{1\ell})}) \ldots \hat{\delta}(h_{x_\ell \ell}, t_{(n\S i_{x_\ell \ell})})(n, A^0_{2\ell})$$

Define $s_q = \hat{\delta}(1, t_{(n)}) \ldots \hat{\delta}(q, t_{(n)})$ and

$$y_{qj} = \max\{ i \mid (j.i) \text{ occurs in } \hat{u}_1 \ldots \hat{u}_q \text{ or } i = 0\}$$

We then have

$$s_q(t_{(n\S j)}) = \hat{\delta}(1, t_{(n\S j)}) \ldots \hat{\delta}(y_{qj}, t_{(n\S j)})$$

We can now apply the induction hypothesis to $t_{(n\S j)}$. Since $0 \leq y_{qj} \leq m_j$ there is a walk, call it s'_{qj}, through $t_{(n\S j)}$ such that $s_{qj} = s'_q(t_{(n\S j)})s'_{qj}$ is a computation sequence for $t_{(n\S j)}$ and $(F_j, s_{qj}(n\S j, p_j))$ is the label of $n\S j$ in $r(t)$ (p_j applied at $n\S j$ in t). Because of the property b) of the relabelling r (lemma 2) $s_{qj}(n\S j) = \pi_p(j)$. Let $s_{qj} = s^1_{qj} \ldots s^{m_j}_{qj}$ be such that $s^i_{qj}(n\S j) = A^j_{2i-1}A^j_{2i}$. Given the computation sequences $s_{q1} \ldots s_{qk}$ for $t_{(n\S 1)}, \ldots, t_{(n\S k)}$ and given π_p we can construct a computation sequence s for $t_{(n)}$ with $s(n, p) = \pi_p$ and $s(t_{(n\S j)}) = s_{qj}$. This is done by replacing each $(0, A^0_i)$ in π_p by (n, A^0_i) for $1 \leq i \leq 2m_0$ and by replacing each pair $(j, A^j_{2i-1})(j, A^j_{2i})$ by s^i_{qj} for $1 \leq i \leq m_j$, $1 \leq j \leq k$. Since π_p is a partition of the attributes of p

satisfying D(p) (lemma 2) it is easy to see that s is a computation sequence for $t_{(n)}$. Let $s = s''_1 \ldots s''_{m_0}$ be such that $s''_i(n) = A^0_{2i-1}A^0_{2i}$. For $1 \leq \ell \leq q$ we have $s''_\ell = s_\ell$ so $s_q = s''_1 \ldots s''_q$. Then $s'_q = s''_{q+1} \ldots s''_{m_0}$ is a walk through $t_{(n)}$ such that $s = \hat{\delta}(1, t_{(n)}) \ldots \hat{\delta}(q, t_{(n)})s'_q$ is a computation sequence for $t_{(n)}$ with $s(n,p) = \pi_p$. This proves (*).

By letting $q = m_0$ in (*) we get that $\hat{\delta}(1, t_{(n)}) \ldots \hat{\delta}(m_0, t_{(n)})s'_{m_0}$ is a computation sequence for $t_{(n)}$. Since for $1 \leq \ell \leq m_0$ $\hat{\delta}(\ell, t_{(n)})(n) = A^0_{2\ell-1}A^0_{2\ell}$ it follows that $s'_{m_0} = \lambda$. So $\hat{\delta}(1, t_{(n)}) \ldots \hat{\delta}(m_0, t_{(n)})$ is a computation sequence for $t_{(n)}$. This proves that each derivation tree t with root labelled S has a computation sequence s (= $\hat{\delta}(1, t)$) and $(t, s) \in \mathfrak{I}(M)$.

□

5. EVALUATORS FOR ABSOLUTELY WELL-DEFINED AGs

The constructions given in the previous section apply to any well-defined AG as those given by [3] and [12]. Kennedy and Warren's evaluator ([8]) is obtained as a special case of that in [12] and it only applies to the absolutely well-defined AGs. In this section we will show how to simplify the construction of section 4 for absolutely well-defined AGs.

In [10] the absolutely well-defined AGs are characterized by properties of computation sequences. We will first modify the evaluator of section 4 to construct computation sequences with these properties and next we will show that the relabelling of the evaluator is not needed for this class of AGs.

From [10] we have the following definitions: An assignment of partitions to the productions P of an AG is any (partial) mapping

$$\mathcal{G}: P \times \{\pi \mid \pi \in \Pi_F,\ F \in V_N\} \to \{\pi_p \mid \pi_p \in \Pi_{\pi,p} \text{ for some } \pi, p \in P\}$$

satisfying that $\mathcal{G}(p,\pi)$ is a partition of the attributes of p with $\mathcal{G}(p,\pi)(0) = \pi$. A computation sequence s for a subtree $t_{(n)}$ of t is uniform with respect to $\mathcal{G}$ if

- $s(n,p) = \mathcal{G}(p, s(n))$ where $p: F_0 \to v_0F_1v_1 \cdots v_{k-1}F_kv_k$ is applied at n
- $s(t_{(n\S 1)})$ is a computation sequence for $t_{(n\S j)}$ that is uniform with respect to $\mathcal{G}$ $(1 \leq j \leq k)$

And we have the following characterization stated and proved in [10]:

An AG is absolutely well-defined if and only if there is an assignment $\mathcal{G}$ of partitions to productions and a partition π_0 of the attributes of S such that each derivation tree with root labelled S has a computation sequence s which is uniform with respect to $\mathcal{G}$ and has $s(\lambda) = \pi_0$.

It is easy to change the evaluator of section 4 to construct these special computation sequences. In fact, we only need change the relabelling as it fully determines the computation sequences - remember that $s(n,p)$ is the flag at the node n. Consider now a derivation tree t where $p: F_0 \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ is applied at the root of t and define for a partition π of the attributes of F_0:

$$R(\pi, t) = (F_0, \mathcal{A}(p,\pi))\ [R(\mathcal{A}(p,\pi)(1), t_{(1)})\ldots R(\mathcal{A}(p,\pi)(k), t_{(k)}]$$

The relabelling constructed in this way can be proven to satisfy lemma 1, and theorem 1 gives that the evaluator is correct. From the proof of theorem 1 we directly get that the computation sequences have the right form. So we state

Theorem 2

Consider an absolutely well-defined AG with assignment $\mathcal{A}$ of partitions to productions and initial partition π_0. Then there is an evaluator M for the AG such that if $(t,s) \in \mathcal{J}(M)$ then s is uniform with respect to $\mathcal{A}$ and has $s(\lambda) = \pi_0$.

For the ordered AGs considered by [5, 7] we can simplify the construction of the evaluator further and we get (almost) the same evaluators as [5, 7]. Again it is only necessary to change the relabelling in order to obtain a theorem corresponding to that above (in fact it can be omitted). For the pass-oriented subclasses of AGs considered in e.g. [2] and [6] we can give characterizations by computation sequences ([10, 5]). It seems to be possible to construct evaluators which give computation sequences reflecting the pass properties by only modifying the relabelling of our evaluator.

In the rest of this section we will show that the relabelling of the evaluator of theorem 2 is in fact not needed. The reason is that it is deterministic top-down ([4]) and since the evaluator itself also operates in a top-down fashion, it is possible to remember the relabelling r in the states of the evaluator. In general the transition mapping δ of the evaluator is of the form

$$\delta_p(q,\pi_p) = (A_1,(j_1,q_1)\ldots(j_m,q_m),A_S) \qquad (*)$$

where $p: F_0 \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ and π_p is a partition of the attributes of p. In our case π_p is fully determined from p and $\pi_p(0)$ (via the assignment $\mathbb{G}$) so (*) can be rewritten as

$$\delta'_p(<q,\pi_0>,-) = (A_1,(j_1,<q_1,\mathbb{G}(p,\pi_0)(j_1)>)\ldots$$
$$\ldots(j_m,<q_m,\mathbb{G}(p,\pi_0)(j_m)>),A_S) \qquad (**)$$

Thus the evaluator $M = (Q,\Sigma,\delta,\hat{q}_0,r)$ of theorem 2 can be simulated by an evaluator $M' = (Q',\Sigma',\delta',\hat{q}_0',r')$ where

- $Q' = Q \times \{\pi \mid \pi \in \Pi_F,\ F \in V_N\}$
- $\Sigma' = \{-\}$
- δ' is defined from δ as indicated by (*) and (**)
- $\hat{q}_0' = <\hat{q}_0,\pi_0>$ (since $\hat{q}_0 \in Q$)
- r': if $p: F_0 \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ is applied at the root of t then
  $$r(t) = <F_0,->[r(t_{(1)})\ldots r(t_{(k)})]$$
  (i.e. r is "almost" the identity)

This leads to the following definition: A <u>direct evaluator</u> for an AG is an evaluator $M = (Q,\Sigma,\delta,\hat{q}_0,r)$ where

- Σ $(= \{-\}$ is a singleton
- r is the relabelling defined by
  $$r(t) = <F_0,->[r(t_{(1)})\ldots r(t_{(k)})]$$
  where $p: F_0 \to v_0F_1v_1\ldots v_{k-1}F_kv_k$ is applied at the root of the derivation tree t.

With this definition we easily see that the first part of the following theorem holds.

Theorem 3

An AG is absolutely well-defined if and only if it has a direct evaluator that is correct.

□

To prove the second part of the theorem we note that the computation sequences constructed by a direct evaluator do not have to be uniform with respect to an assignment $\mathcal{A}$. So we will use another characterization of the absolutely well-defined AGs, also given in [10]. First we need define that a property $\mathcal{P}$ of computation sequences is preserved under substitution if for any derivation trees t and t' with roots labelled S and for 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 furthermore $s'(t'^{(n)}) = s(t^{(n)})$.

The characterization of the absolutely well-defined AGs we are looking for is as follows: An AG is absolutely well-defined if and only if there is a property $\mathcal{P}$ of computation sequences such that

1) each derivation tree with root labelled S has a computation sequence with property $\mathcal{P}$
2) property $\mathcal{P}$ is preserved under substitution.

Now let $M = (Q, \Sigma, \delta, \hat{q}_0, r)$ be a direct evaluator for an AG which is correct. We will show that the AG is absolutely well-defined. Define that a computation sequence s has property $\mathcal{P}$ if and only if $(t, s) \in \mathcal{T}(M)$ for some derivation tree t. Since M is correct condition 1) above is clearly satisfied. Thus it suffices to prove that property $\mathcal{P}$ is preserved under substitution.

Consider a derivation tree t with root labelled S. To each node n in t we can associate an element $q_1 \ldots q_m$ of Q^* where q_i is the state in which M visited n for the i'th time, and m is the total number of visits to n. Let $st(t,n) = q_1 \ldots q_m$ be the state sequence associated with n in t.

Now let t' be a derivation tree and assume that for some n, $t^{(n)} = t'^{(n)}$. by induction on the length of the path from the root of t (t') to a node n' in $t^{(n)}$ ($t'^{(n)}$) we will show that $st(t, n') = st(t', n')$. If the length of the path is zero (i.e. $n' = \lambda$) we have $st(t, n') = \hat{q}_0 = st(t', n')$. So let n' be any node in $t^{(n)}$ ($t'^{(n)}$) and let n'' be the father of n'. The induction hypothesis gives $st(t, n'') = st(t', n'')$. For each state q in the sequence $st(t, n'')$ the transition mapping δ gives a subsequence of the sequence $st(t, n')$. Similarly in t' we can determine $st(t', n')$ from $st(t', n'')$, δ and the production applied at n''. So it is easy to see that $st(t, n') = st(t', n')$ and the induction step is completed. Now let s and s' be the computation sequences for t and t', respectively, constructed by M. Since M is direct the state sequence $st(t, n')$ associated with n' in t determines $s(n', p)$ uniquely (p is the production applied at n'). Similarly, $st(t', n')$ determines $s'(n', p')$ uniquely. When n' is in $t^{(n)}$ ($= t'^{(n)}$) and $n' \neq n$ we have $s(n', p) = s'(n', p)$ and it follows that $s(t^{(n)}) = s'(t'^{(n)})$. This means exactly that property $\mathcal{P}$ is preserved under substitution and the second part of theorem 3 has been proved.

□

6. CONCLUSION

In this paper we have shown how to construct computation sequences for derivation trees by an evaluator. We have formally defined evaluators and have shown how to construct a correct evaluator for any well-defined AG. Using the characterization of the absolutely well-defined AGs given in [10] we showed how our construction easily could be modified such that the evaluator produces computation sequences with some special properties. In fact only a preprocessing stage of the evaluator needed be changed. For other subclasses of AGs characterized by computation sequences similar modifications seem to be possible. Because only a preprocessing stage of the evaluator has to be changed in order to obtain evaluators reflecting special properties of AGs we claim that our evaluator is simpler than the general ones of e.g. [3] and [12].

Acknowledgement

I wish to thank Joost Engelfriet for his helpful comments, especially his suggestion of theorem 3. Also thanks to Brian Mayoh and Flemming Nielson.

REFERENCES

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

[2] G.V. Bochmann, Semantic evaluation from left to right, Comm. ACM, 19 (1976), pp. 55-62.

[3] R. Cohen and E. Harry, Automatic generation of near-optimal linear-time translators for non-circular attribute grammars, Conf. Record of the Sixth ACM Symp. on Principles of Programming Languages (1979), pp. 121-134.

[4] J. Engelfriet, Bottom-up and top-down tree transformations - a comparison, Math. Systems Theory 9, pp. 198-231.

[5] J. Engelfriet and G. File, Simple multi-visit attribute grammars, Mem. 314, Twente University of Technology, The Netherlands, 1980.

[6] M. Jazayeri and K.G. Walter, Alternating semantic evaluator, Proc. ACM 1975 Annual Conference (1975), pp. 230-234.

[7] U. Kastens, Ordered attribute grammars, Acta Informatica 13 (1980), pp. 229-256.

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

[9] D.E. Knuth, Semantics of context free languages, Math. Systems Theory 2, (1968), pp. 127-145.

[10] H.R. Nielson, Computation sequences: A way to characterize subclasses of attribute grammars, Aarhus University, Denmark, 1981.

[11] H. Riis and S. Skyum, k-visit attribute grammars, to appear in Math. Systems Theory (1982).

[12] S.K. Warren, The efficient evaluation of attribute grammars, Master's Thesis, Rice University, Texas, 1975.