Acta Informatica 16, 275–302 (1981)

© Springer-Verlag 1981

The Formal Power of One-Visit Attribute Grammars

Joost Engelfriet and Gilberto Filè

Department of Applied Mathematics, Twente University of Technology,
P.O. Box 217, 7500 AE Enschede, The Netherlands

Contents

Summary. An attribute grammar is one-visit if the attributes can be evaluated by walking through the derivation tree in such a way that each subtree is visited at most once. One-visit (1 V) attribute grammars are compared with one-pass left-to-right (L) attribute grammars and with attribute grammars having only one synthesized attribute (1 S).

Every 1 S attribute grammar can be made one-visit. One-visit attribute grammars are simply permutations of L attribute grammars; thus the classes of output sets of 1 V and L attribute grammars coincide, and similarly for 1 S and L-1 S attribute grammars. In case all attribute values are trees, the translation realized by a 1 V attribute grammar is the composition of the translation realized by a 1 S attribute grammar with a deterministic top-down tree transduction, and vice versa; thus, using a result of Duske e.a., the class of output languages of 1 V (or L) attribute grammars is the image of the class of IO macro tree languages under all deterministic top-down tree transductions.

Introduction

An attribute grammar (AG), as defined in [18], is a context-free grammar with (synthesized and inherited) attributes associated with each of its nonterminals, whose values are computed by means of semantic rules associated with its context-free productions. An AG realizes a translation by giving a meaning to each derivation tree, viz. the value of a (designated) synthesized attribute of its root.

The formal power of attribute grammars (with both synthesized and inherited attributes) as a translation defining mechanism has not yet been investigated very much. This lack of enthousiasm in incorporating attribute grammars into formal language and automata theory may be due partly to the complexity of the model and partly to the (obvious) fact that attribute grammars of a very simple type, having only one synthesized attribute and no inherited attributes, can already realize all possible translations [18] and can e.g. easily simulate type 0 Chomsky grammars (cf. [19]). The latter phenomenon is clearly caused by the power inherent in the involved (semantic) domains of attribute values and in the allowable operations on them (as used in the semantic rules). Since we are not interested in meaning, but rather in the method used by attribute grammars for assigning meaning to derivation trees, we should either abstract from the meaning of the semantic operations (as in program scheme theory) or restrict our attention to a fixed domain (such as the set of strings with the operation of concatenation, the set of trees, or an arbitrary fixed domain). In program scheme theory it can often be shown that both approaches amount to the same thing (in particular when taking the "free" domain of trees). In this paper we will study the formal power of attribute grammars using the second approach. In the following points (1)–(3) we discuss some earlier work done in this direction.

(1) In the "pre-Knuth era" (before [18]) attribute grammars were called syntax-directed translations and had synthesized attributes only (OnlyS-AG, i.e. without inherited attributes). These translations have been formalized and investigated intensively as syntax-directed translation schemes, see e.g. [1], and as top-down (!) tree transducers, see e.g. [26, 27, 11]. Their attribute values are either strings or trees.

(2) Lewis, Rosenkrantz and Stearns [20] show that arbitrary AGs are more powerful than L-AGs (i.e. AGs whose attributes can be evaluated in one left-to-right pass, see also [6]), and L-AGs are more powerful than OnlyS-AGs. Although their model is different from the one of Knuth [18], their results on power are essentially obtained by forcing the values of one synthesized attribute to be strings (of "action symbols"). It is also shown in [20] that the L-AG translations can be realized by a machine: the attributed pushdown transducer.

(3) Duske, Parchmann, Sedello and Specht [10] characterize the class of output languages (i.e. ranges of translations) of L-1S attribute grammars (L-AGs having only one synthesized attribute) over the domain of strings as the class of languages generated by the well-known IO (inside-out) macro grammars of Fischer [15]. For the interested reader we note that this result can be understood intuitively by observing that the arguments of a macro (nonterminal) may be viewed as inherited attributes and the string generated by it as (one) synthesized attribute. (It is also related to the fact that the IO macro languages are the homomorphic image, under the "YIELD" mapping, of certain recognizable tree languages [21, 13]: the YIELD mapping can clearly be realized by such an attribute grammar). We observe that the 1S restriction on attribute grammars is quite natural: the value of the synthesized attribute of the root of a tree is *the*

translation of the tree (and no auxiliary translations are needed). Early papers on syntax-directed translation also assumed the 1S restriction (cf. the discussion in [18]).

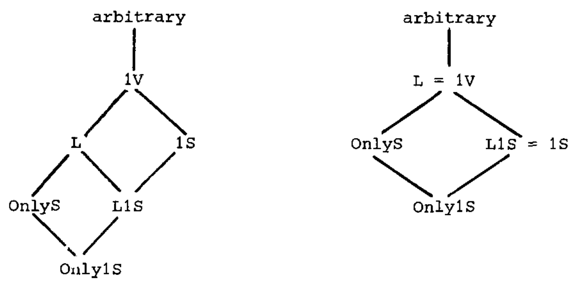
This paper is based on and continues the work of Duske e.a. [10]. Our results answer the question (which we asked ourselves when reading [10]) whether both the L and the 1S restriction are necessary in their characterization result. It turns out that (a) the L condition can be dropped, but (b) the 1S condition is necessary. Moreover, we give a characterization of the class of L-AG output languages similar to the one of Duske e.a.

The essential reason that (a) the L condition can be dropped (i.e., $1S=L\text{-}1S$ for output languages) is that 1S-AGs (over any semantic domain) "by nature" satisfy a property which generalizes the L property: they are "one-visit" (abbreviated by 1V) meaning that attribute evaluation of a derivation tree can proceed by visiting each subtree at most once. This led us to consider the class of one-visit attribute grammars in general. Using a simple static characterization of the 1V property, we show that 1V-AGs are essentially just "permutations" of L-AGs and hence their classes of output sets coincide (and similarly for 1S and L-1S, respectively).

The fact that (b) the 1S condition is necessary means that L is more powerful than L-1S for output languages, or in other words (in view of the preceding remarks on one-visit) that 1V is more powerful than 1S for output languages. In order to show the latter we express (for the domain of trees) the 1V-AG translations in terms of 1S-AG translations and top-down tree transductions (which correspond to OnlyS-AGs). This can be done because the 1V property implies that all synthesized attributes can be computed simultaneously as if they were just one synthesized attribute, whereas the use of many rather than one synthesized attribute corresponds to a deterministic top-down tree transducer. This implies that we can characterize the class of 1V-AG output (tree) languages as the class of images of 1S-AG output tree languages under the deterministic top-down tree transductions (and similarly for L and L-1S, respectively). Since a characterization result corresponding to the one of Duske e.a. also holds for trees, and since the class of IO macro tree languages is not closed under deterministic top-down tree transductions, the result then follows.

Altogether, the above results establish close relationships between the class of 1V-AGs on the one hand and the classes of L-AGs and 1S-AGs on the other hand. These relationships enable us to show that the diagrams of Fig. 1 characterize the power of the investigated classes of attribute grammars with respect to their translations and their output sets. The inclusions shown hold for every fixed semantic domain; moreover there exists a semantic domain (viz. the domain of trees) for which the diagrams do not collapse, i.e. all inclusions are proper and all unconnected classes are incomparable.

The paper is organized as follows. Section 1 contains preliminary notation and a few easy new concepts. Since the one-visit property permeates our whole approach, we start in Sect. 2 by defining it. We show that it can be detected statically, i.e. by an easy property of the dependencies in the semantic rules, just as in the L case [6]. Then we show that each 1S-AG is equivalent to a 1S-AG which is one-visit. Actually we define two, slightly different, 1V properties: one



(a) Translations (b) Output sets

Fig. 1, a, b

can be detected statically and the other is a property of all 1 S-AGs; the difference is that in the one case we require the evaluation of all attributes, in the second case only of those which are needed to compute the translation of the derivation tree, i.e. the designated synthesized attribute of its root. In Sect. 3 we formulate the fact that 1 V attribute grammars are permutations of L attribute grammars (and 1 S-AGs of L-1 S-AGs). In Sect. 4 we prove that (for trees) the 1 V-AG translations can be decomposed into 1 S-AG translations and deterministic top-down tree transductions (and similarly for L and L-1 S). Section 5 contains a discussion of how the diagrams of Fig. 1 can be proved correct.

We finally note that throughout the paper we assume the reader to be familiar with attribute grammars (e.g. [18, 6]). Section 1–3 do not require any knowledge of tree transducers (in Sect. 4–5 this would be helpful).

1. Preliminaries

We denote by $[m, n]$ the set of integers $\{i \mid m \leq i \leq n\}$ and, for functions f and g, by $f \circ g$ the function such that $(f \circ g)(x) = g(f(x))$.

In the rest of this section we recall the definition of attribute grammar [18, 6] and discuss some related concepts which we think are necessary for the understanding of the paper.

We start by defining the notion of semantic domain. As we said in the introduction, the concept of semantic domain is of great importance in our approach to the formal power of attribute grammars.

A *semantic domain* D is a pair (Ω, Φ) where Ω is a set of sets, called *sets of attribute values*, and Φ is a collection of mappings of the form $f\colon V_1 \times V_2 \times \ldots \times V_m \to V_0$ with $m \geq 0$ and $V_i \in \Omega$ for $i \in [0, m]$. The mappings in Φ are called *semantic functions*. Note that in the case that $m = 0$, f is simply an element of V_0.

Assumption 1.1 (on the semantic domain). Throughout the paper we consider only semantic domains $D = (\Omega, \Phi)$ such that Φ contains an element of V, for every $V \in \Omega$. □

Next we give the definition of an attribute grammar (abbreviated by AG). This definition is practically the same as the original one of Knuth [18], the only difference being the stress laid on the semantic domain.

An *attribute grammar G over semantic domain* $D=(\Omega_D, \Phi_D)$ consists of (1)–(4) as follows.

(1) A context-free grammar (CFG) $G_0=(T, N, P, Z)$, called the *underlying context-free grammar* of G, consisting of terminals, nonterminals, productions and initial nonterminal respectively. We will always denote production p of G_0 (or of G) as

$$p\colon\ F_0 \to w_0 F_1 w_1 \ldots w_{n_p-1} F_{n_p} w_{n_p}$$

where $F_i \in N$ and $w_i \in T^*$ for $i \in [0, n_p]$ and $n_p \geqq 0$.

Note that, as usual, subscripts put on the nonterminals of the right-hand side will be used as a way of distinguishing among different occurrences of the same nonterminal. Note also that, when considering a derivation tree of G_0, we assume its leaves to be labeled with terminals (or the empty string); a derivation tree is said to be complete if its root is labeled with Z.

Assumption 1.2 (on the underlying CFG). We will consider only AGs G such that the underlying CFG G_0 is reduced in the usual sense, that is, every nonterminal is reachable from the initial nonterminal and can generate a string of terminals. □

(2) Each nonterminal F of G_0 (or of G) has two associated finite sets, denoted $S(F)$ and $I(F)$, $S(F) \cap I(F) = \emptyset$. The set $S(F)$ contains the *synthesized attributes* of F (abbreviated by s-attributes) and $I(F)$ its *inherited attributes* (i-attributes). The initial nonterminal Z does not have any i-attribute and, always, one element of $S(Z)$ is designated to hold the translation of the derivation tree (and we refer to it as the designated s-attribute of Z). We will denote an attribute a of nonterminal F also by $a(F)$.

(3) For each attribute a, Ω_D contains an associated set $V(a)$ of the possible values of a.

(4) With each production p of P is associated a set r_p of *semantic rules* which define all and only the attributes in $S(F_0)$ and in $I(F_j)$, for $j \in [1, n_p]$. A semantic rule of r_p defining attribute $a(F_k)$, $k \in [0, n_p]$, is specified by (i) a mapping $f \in \Phi_D$ of the form $f\colon V_1 \times V_2 \times \ldots \times V_m \to V_0$, where $V_0 = V(a(F_k))$, and (ii) by a sequence of m attributes of nonterminals $a_1(F_{i_1}), a_2(F_{i_2}), \ldots, a_m(F_{i_m})$, $i_j \in [0, n_p]$, such that $V_j = V(a_j(F_{i_j}))$ for all $j \in [1, m]$. We will write such a semantic rule as

$$a(F_k) \leftarrow f(a_1(F_{i_1}), \ldots, a_m(F_{i_m}))$$

or simply $a \leftarrow f(a_1, \ldots, a_m)$ when the identity of the nonterminals is not important. We say that, in p, $a(F_k)$ *depends on* $a_1(F_{i_1}), \ldots, a_m(F_{i_m})$.

Assumption 1.3 (on semantic rules). We assume that, in every semantic rule $a \leftarrow f(a_1, \ldots, a_m)$ of r_p, every a_i, $i \in [1, m]$, is either an attribute of $I(F_0)$ or of some $S(F_j)$, $j \in [1, n_p]$. See e.g. (3) in Sect. 4 of [6]. □

This ends our definition of attribute grammar.

Now we give some short definitions and terminology concerning concepts related to AGs which are going to be used in what follows.

The *translation realized* by AG G, denoted by $T(G)$, is the mapping from complete derivation trees of G to the set of values of the designated s-attribute of G, such that if t is a complete derivation tree, then $T(G)(t)$ is the value of the designated s-attribute of the root of t (after attribute evaluation). The *output set* of G, denoted by $\mathrm{OUT}(G)$, is the range of $T(G)$, that is $\mathrm{OUT}(G)=\{T(G)(t) \mid t$ is a complete derivation tree of $G\}$. In case $\mathrm{OUT}(G)$ is a set of strings or trees, we will call it the output language of G. We define and denote translations and output sets of a class X of AGs over a given semantic domain D as follows: $T(X, D)= \{T(G) \mid G \in X$ and G is over $D\}$ and $\mathrm{OUT}(X, D)=\{\mathrm{OUT}(G) \mid G \in X$ and G is over $D\}$.

Since we will only compare AGs over the same domain, we define equivalence of AGs as follows. Two AGs, G_1 and G_2, are *equivalent* if they are over the same semantic domain and $T(G_1)=T(G_2)$.

Next we recall the concept of dependency graph of a production of an AG, as introduced originally by Knuth [18]. Our terminology will be closer to the one used in [17].

Let G be an AG and consider production p of G. The *dependency graph* of p (denoted by DG_p) is the graph which has as nodes all attributes of all F_j, for $j \in [0, n_p]$, and whose arcs are defined as follows: there is an arc from $a_1(F_{j_1})$ to $a_2(F_{j_2})$ iff $a_2(F_{j_2})$ depends on $a_1(F_{j_1})$. By viewing derivation trees as the connection of single productions, it is easy to extend to them the concept of dependency graph: for a derivation tree t of G, the dependency graph of t, denoted by $DG(t)$, is the graph obtained by connecting, according to t, the dependency graphs of the productions used in t. For a more precise definition see [18]. It is easy to extend to derivation trees also the idea of "one attribute depending on another": for a derivation tree t, if in $DG(t)$ there is an oriented path running from node a_1 to node a_2, then we say that attribute a_1 *depends on* attribute a_2 *in* t. We say that an attribute grammar G is *noncircular* ("well-formed" in [18]) if there is no derivation tree t of G such that $DG(t)$ contains an oriented cycle.

Assumption 1.4 (on noncircularity). We consider only noncircular AGs. □

We now recall the concept of input/output graph (or i/o graph) of a derivation tree and of a nonterminal. Consider a derivation tree t of an AG G and assume its root is labeled by nonterminal F. Then the *input/output graph* of t is the graph whose nodes are the attributes of F and in which there is an arc from a_1 to a_2 iff a_2 depends on a_1 in t (as attributes of the root). From this we immediately define an input/output graph of nonterminal F to be any graph which is the i/o graph of a derivation tree with root F.

It will be important, in the sequel, to consider the way different nonterminals of the right-hand side of a production depend on each other through their attributes. In order to have a clear picture of this dependency situation we introduce the concept of brother-graph of a production as follows (see also [22]). Given an AG G and a production p of G, the *brother-graph* of p (denoted by BG_p) is the graph whose nodes are the nonterminals $F_1, \ldots, F_{n_p}$ of the right-hand side of p and such that there is an arc from F_i to F_j iff some element of $I(F_j)$ depends on some element of $S(F_i)$, $i, j \in [1, n_p]$. Note that in this definition we already assume Assumption 1.3.

Finally we define a few properties of AGs. For a property X we denote the class of AGs which satisfy X as X-AG (and the class of all AGs is denoted as AG). Particular classes of AGs which we will consider in this paper are the following.

L-AG: defined in [20, 6] as the class of AGs for which all attributes of all derivation trees can be evaluated in a single pass from left to right. Equivalently, exploiting already the concept of brother-graph, an AG is L if for every production p, BG_p has no arc from F_i to F_j if $j \leq i$ (see also Theorem 2 of [6]).

1S-AG: the class of AGs whose nonterminals have at most one s-attribute (and an arbitrary number of i-attributes).

OnlyS-AG: the class of AGs whose nonterminals have no i-attributes.

Only1S-AG: clearly, the class of AGs whose nonterminals have no i-attributes and at most one s-attribute.

2. The One-Visit Property

In this section we define "one-visit" attribute evaluation of AGs as a generalization of left-to-right (or right-to-left) one-pass attribute evaluation [6, 20]. We present an easy static condition on the productions of the grammar (also considered in [22]) to decide whether its attributes can be evaluated in one visit (Theorem 2.1). Then we examine the class of 1S-AGs and show that a slightly more general notion of one-visit attribute evaluation naturally applies to them. We prove that for any 1S-AG an equivalent 1S-AG can be constructed satisfying the (strict) one-visit property (Theorem 2.2).

An attribute grammar will be called "one-visit" if the attributes of every derivation tree can be evaluated by walking through the tree without visiting any subtree more than once. Before defining this more formally we observe that in general two different approaches to attribute evaluation may be discerned: the first ("conventional" one) requires all attributes of all nodes to be computed, whereas the second (which we will call the "translational" approach) requires that, for every complete derivation tree t, only those attributes are evaluated which are needed to compute the translation of t, i.e. the value of the designated s-attribute of its root. Although the second approach is more natural, the first has certain theoretical advantages. For "one-visit" we will discuss both concepts but mainly consider the first.

Definition 2.1. Let t be a derivation tree of a given attribute grammar. An *attribute evaluation strategy for* t is a way of walking along the branches of t and computing attributes of its nodes. An attribute evaluation strategy for t is *one-visit* (abbreviated by 1V) if it visits each subtree t' of t at most once, and in such a way that entering t' it computes some i-attributes of the root n of t', and exiting t' it computes some s-attributes of n. $\square$

Thus, for every node, a one-visit attribute evaluation strategy first visits the node computing some of its i-attributes, then visits some of its sons (and their descendants) in some order (computing some of their attributes), and finally visits the node again computing some of its s-attributes.

Now we define one-visit attribute grammars, according to both above-mentioned approaches.

Definition 2.2. (i) An attribute grammar G is *one-visit* (1V) if for every complete derivation tree t of G there exists a one-visit attribute evaluation strategy computing all attributes of all nodes of t.

(ii) An attribute grammar G is *translationally one-visit* if for every complete derivation tree t of G there exists a one-visit attribute evaluation strategy which computes the designated s-attribute of the root of t. □

Note that Definition 2.2(i), requiring the computation of all attributes, implies that the one-visit attribute evaluation strategy enters each subtree computing all i-attributes of its root n and exits it computing all s-attributes of n.

Example 2.1. As an easy example, imagine a programming language in which identifiers may be declared both before and after the statement list of a block. This can be expressed by the context-free production

$$\langle\text{block}\rangle \to \textbf{begin}\ \langle\text{decls}_1\rangle\ \textbf{in}\ \langle\text{statlist}\rangle\ \textbf{where}\ \langle\text{decls}_2\rangle\ \textbf{end}$$

where $\langle\text{decls}_1\rangle$ and $\langle\text{decls}_2\rangle$ are two occurrences of the same nonterminal $\langle\text{decls}\rangle$, distinguished as usual by subscripts. To check whether all identifiers have been declared, there is an i-attribute 'context' (of all nonterminals) which contains a list of all valid declarations, and an s-attribute 'updated" (of $\langle\text{decls}\rangle$ only) which contains context($\langle\text{decls}\rangle$) with the declarations produced by $\langle\text{decls}\rangle$ added to it. The semantic rules corresponding to the above production are

$$\begin{gathered}\text{context}(\langle\text{decls}_1\rangle) \leftarrow \text{context}(\langle\text{block}\rangle),\\ \text{context}(\langle\text{statlist}\rangle) \leftarrow \text{updated}(\langle\text{decls}_2\rangle),\\ \text{context}(\langle\text{decls}_2\rangle) \leftarrow \text{updated}(\langle\text{decls}_1\rangle).\end{gathered}$$

The part of the dependency graph corresponding to these semantic rules is given in Fig. 2 (solid arcs). It should be clear that this (part of an) attribute grammar is one-visit: assuming that one visit to $\langle\text{decls}\rangle$ suffices to compute updated($\langle\text{decls}\rangle$) from context($\langle\text{decls}\rangle$), as indicated by the dotted lines in Fig. 2, the subtrees of $\langle\text{block}\rangle$ can always be visited in the order $\langle\text{decls}_1\rangle$, $\langle\text{decls}_2\rangle$, $\langle\text{statlist}\rangle$. □

In the next definition we present a static property of AGs which we will prove equivalent to the one-visit property of Definition 2.2(i). For the definition of the static one-visit property we use the concept of brother-graph as defined in Sect. 1.

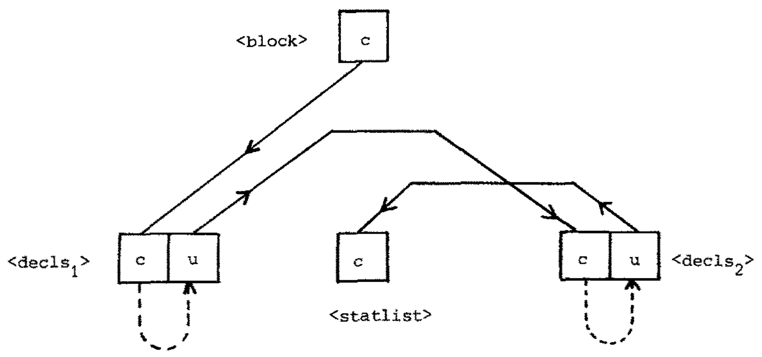


Fig. 2

Definition 2.3. An attribute grammar G is *statically one-visit* (abbreviated by s-1V) if for every production p of G the brother-graph BG_p has no oriented cycles. $\square$

As an example, the brother-graph of the production in Example 2.1 is given in Fig. 2 if one views the c- and u-node of $\langle\text{decls}\rangle$ to be one node; it clearly has no oriented cycles and consequently is statically one-visit.

To show that each s-1V attribute grammar is also 1V, we now construct a one-visit attribute evaluation algorithm as implied by the s-1V property (a similar algorithm is given in [22, 23]). To do this we need the notion of a "visiting sequence" (of the direct descendants of a node), which we will now explain. Consider an attribute grammar G which is s-1V and let production p of G have the (usual) form p: $F_0 \to w_0 F_1 w_1 \ldots w_{n_p-1} F_{n_p} w_{n_p}$. Recall that by Assumption 1.3 of Sect. 1 the semantic rules of p define all and only the attributes of $S(F_0)$ and $I(F_j)$ in terms of those of $I(F_0)$ and $S(F_k)$, for $j, k \in [1, n_p]$. In this respect, we can view BG_p as a picture of how the nonterminals of the right-hand side of p depend on each other. From the fact that there are no cycles in BG_p (Definition 2.3) it is easy to see that there is at least one ordering in which to visit the F_i, $i \in [1, n_p]$, such that when F_j is visited, all the F_k it depends on have already been visited. We will call such an ordering a visiting sequence.

Definition 2.4. Let G be an attribute grammar. A *visiting sequence* v_p of production p of G is a sequence $v_p = \langle i_1, i_2, \ldots, i_{n_p}\rangle$ which is a permutation of the sequence $\langle 1, 2, \ldots, n_p\rangle$ such that, for all $j, k \in [1, n_p]$, if $j \leq k$ then in BG_p there is no arc from F_{i_k} to F_{i_j}. $\square$

As an example, the production of Example 2.1 has visiting sequence $\langle 1, 3, 2\rangle$ and no others (note that a production can have more than one visiting sequence). Observe that an AG is s-1V if and only if there exists a visiting sequence for each of its productions (this version of s-1V was introduced in [22], where it is called "reordered", and in [23] where it is called "grammar with permutation attribute scheme").

The attribute evaluation algorithm is constructed easily from these visiting sequences (and depends on their choice). Since it is obtained from the static 1V property, we will call it the "static algorithm".

Algorithm 2.1 (the static algorithm)

```
begin
procedure evaluate-node($m$);
    begin compute all attributes of $I(m)$;
        {assume production $p$ is applied at node $m$ and let $v_p = \langle i_1, \ldots, i_{n_p}\rangle$ be a
        visiting sequence of $p$}
        for $j := 1$ to $n_p$ do
            evaluate-node ($i_j$-th nonterminal son of $m$)
                    od;
        compute all attributes of $S(m)$
    end;
evaluate-node(root)
end  $\square$
```

Clearly the static algorithm has a fixed behaviour for each production p of G independently from the particular occurrence of p in the same or in a different derivation tree of G. Instead, the 1V definition would allow the attribute evaluation strategy to behave differently at different occurrences of the same production. That this difference between 1V and s-1V is only apparent will be shown in the following theorem.

Theorem 2.1. *An attribute grammar is statically one-visit if and only if it is one-visit.*

Proof. ($\Rightarrow$) Let G be a statically one-visit AG. It is easy to see that the static algorithm (Algorithm 2.1) provides a one-visit strategy for all derivation trees of G. Therefore we have to show only that it computes all attributes, i.e. that it does not block. The proof is by induction on the height of the derivation trees with the following induction hypothesis: if the values needed to evaluate the i-attributes of the root m are available, the static algorithm (i.e. the call evaluate-node(m)) evaluates all the attributes of all nodes of the tree rooted at m. The base of the induction is obvious, and the applicability of the induction hypothesis in the induction step is ensured by the property of the visiting sequence (Definition 2.4).

($\Leftarrow$) Consider a 1V-AG G and assume that G is not s-1V. Then there is a production p of G such that BG_p has an oriented cycle. Let the nodes connected by the cycle be $F_{k_1}, \dots, F_{k_z}$ with $z \geq 1$. The existence of the cycle implies that any attribute evaluation strategy (which computes all attributes) will have to enter at least one of those nodes, say F_{k_i}, when not all elements of $I(F_{k_i})$ are computable! But this contradicts Definition 2.2(i) of 1V (see the observation following this definition). Note that here we use Assumption 1.2 that G has a reduced underlying CFG, implying that there is at least one derivation tree of G in which production p is used. $\square$

We note here that it easily follows from this result that every one-visit attribute grammar is absolutely noncircular as defined in [17].

We have just shown that the 1V property, as defined according to the "conventional" approach (Definition 2.2(i)), can be characterized statically. A similar static characterization of the translational 1V property (Definition 2.2(ii) does not seem to exist (or is much more complicated). This might be the reason that usually, in defining properties of attribute evaluation, the conventional approach has been taken, rather than the translational one. We observe however that an attribute grammar can be "reduced" in the sense that it can be transformed into an equivalent AG with the property that in every dependency graph of a derivation tree each node is connected to the designated s-attribute of the root. For such AGs, since all attributes are needed to compute the translation of the tree, the two approaches coincide.

Let us now turn our attention to the class of 1S-AGs, that is, those AGs whose nonterminal symbols have at most one s-attribute (and any number of i-attributes). As we have suggested in the introduction, 1S-AGs are naturally related to the concept of one-visit. In fact, every 1S-AG is translationally one-visit. Let us intuitively see why. We look at attribute evaluation of a derivation

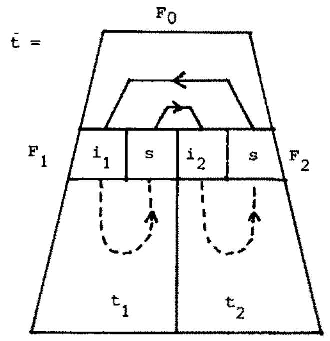


Fig. 3

tree t as a process to which every visit to each subtree t' of t contributes in the following way: a visit to t' is influenced by the parts of t surrounding t' (through some i-attributes of the root of t') and, in its turn, influences the future computation on these surrounding parts (by means of some s-attributes of the root of t'). From this point of view it is clear that, in the case of a 1S-AG, one visit to t' would compute the only s-attribute of its root and, therefore, no second visit would be needed.

Instead of formally proving that every 1S-AG is translationally one-visit, we will show the closely related fact that from every 1S-AG one can construct an equivalent 1S-AG which is 1V. Let us first explain, through an example, why a 1S-AG is not necessarily 1V (i.e. its one-visit attribute evaluation strategy cannot always compute all attributes or, in other words, by Theorem 2.1, some of its productions have oriented cycles in their brother-graphs). Consider a 1S-AG G which is not 1V because its production p: $F_0 \to F_1 F_2$ has a cycle in BG_p: say, i-attribute $i_1(F_1)$ depends on s-attribute $s(F_2)$, and $i_2(F_2)$ depends on $s(F_1)$. Because of Assumption 1.4 of considering only noncircular AG, there cannot exist two derivation trees t_1 and t_2 of G with roots F_1 and F_2 respectively, and such that $s(F_1)$ depends on $i_1(F_1)$ in t_1 and $s(F_2)$ on $i_2(F_2)$ in t_2. In fact, if this would be the case, then we could construct a derivation tree $t = F_0[t_1, t_2]$ of G such that $DG(t)$ has a cycle, as shown in Fig. 3.

In other words, $i_1(F_1)$ or $i_2(F_2)$ is useless in the following sense.

Definition 2.5. Given a 1S-AG G, an i-attribute $i_1(F)$ of a nonterminal F of G is *useless* if there is no derivation tree t in G with root F, such that $DG(t)$ contains an oriented path from $i_1(F)$ to the only s-attribute $s(F)$ of the root of t. □

If, e.g., $i_1(F_1)$ is useless (in our example), then any attribute evaluation strategy would have to visit subtree t_1 twice to compute all its attributes, but only once to compute the s-attribute of its root.

We will now state these observations more formally, but for doing so we need to express a cycle in the brother-graph of a production in terms of the actual attributes whose dependencies give rise to the cycle. Let $p: F_0 \to w_0 F_1 w_1 \ldots w_{n_p-1} F_{n_p} w_{n_p}$ be a production of an AG and let c be a cycle in BG_p consisting of arcs (F_{j_1}, F_{j_2}),

$(F_{j_2}, F_{j_3}), \ldots, (F_{j_m}, F_{j_1})$ where $m \geqq 1$ and $j_k \in [1, n_p]$ for all $k \in [1, m]$. Since each arc of BG_p corresponds to, at least, one arc in DG_p, we can find for each such cycle c in BG_p at least one corresponding sequence of arcs in DG_p: $(s(F_{j_1}), i_2(F_{j_2}))$, $(s(F_{j_2}), i_3(F_{j_3})), \ldots, (s(F_{j_m}), i_1(F_{j_1}))$ where i_k is an i-attribute of F_{j_k} and s is its s-attribute.

Lemma 2.1. *Let G be a* 1*S-AG which is not* 1*V. For any cycle in* BG_p *of a production p of G and any corresponding sequence of arcs in* DG_p, *at least one of the i-attributes occurring in the sequence is useless.*

Proof. Consider without loss of generality a cycle c in BG_p which does not traverse a node more than once (apart from the first). Assume that a sequence of arcs in DG_p corresponding to c is $(s(F_{j_1}), i_2(F_{j_2})), (s(F_{j_2}), i_3(F_{j_3})), \ldots, (s(F_{j_m}), i_1(F_{j_1}))$ where $1 \leqq m \leqq n_p$ and for all $k, n \in [1, m]$ if $k \neq n$ then $j_k \neq j_n$. The fact that at least one i-attribute $i_k(F_{j_k})$ of the sequence must be useless can now easily be proved using noncircularity of G as we did before in the example preceding Definition 2.5. □

We will now take advantage of this result in order to "break" all the cycles in the brother-graphs of a 1S-AG making it 1V. First we observe that there is an effective way of detecting, given a 1S-AG, which are its useless i-attributes. To do so we use the circularity algorithm of Knuth [18] for computing all the i/o graphs of all nonterminals of the given AG. We then detect all useless i-attributes by the following criterion: i-attribute i_1 of nonterminal F is useless iff in no i/o graph of F there is an arc from i_1 to the only s-attribute of F. We note that in the case of 1S-AGs the computation of the i/o graphs need not take exponential time [16]. Actually, it can be shown that Knuth's original (wrong) circularity algorithm [18], which works in polynomial time, is correct for 1S-AG!

Lemma 2.2. *For every* 1*S-AG there exists (effectively) an equivalent* 1*V-*1*S-AG.*

Proof. Given a 1S-AG G over semantic domain $D = (\Omega, \Phi)$, we change it into a 1S-AG G' by making the following changes to its semantic rules: every semantic rule $a_0 \leftarrow f(a_1, \ldots, a_m)$ where a_0 is a useless i-attribute, is changed into the semantic rule $a_0 \leftarrow c$, where c is an element of $V(a_0)$ such that $c \in \Phi$ (the existence of which is guaranteed by Assumption 1.1).

By the construction it is clear that G and G' are defined over the same domain D. To see that G and G' are equivalent it suffices to observe that the value of the (designated) s-attribute of the root of any derivation tree of G never depends on a useless i-attribute, and that G' is obtained from G simply changing those semantic rules of G that define useless i-attributes.

Finally, to show that G' is 1V, consider two corresponding productions p and p' of G and G' respectively. By the construction we know that every arc of $DG_{p'}$ is also in DG_p and that no arc of $DG_{p'}$ enters an i-attribute which is useless in G. This means that every arc of $BG_{p'}$ is also in BG_p and (by Lemma 2.1) that no cycle of BG_p carries over to $BG_{p'}$. □

From the previous lemma we now obtain the following (effective) result on translations of 1S-AG.

Theorem 2.2. *For any semantic domain D*, $T(1\text{S-AG}, D) = T(1\text{V-}1\text{S-AG}, D)$.

We end this section with a few observations on the translational approach (Definition 2.2(ii)).

For 1 S-AGs (which are translationally 1 V) we have shown that an equivalent 1 S-AG can be found which is 1 V, simply redefining the useless i-attributes. About general AGs which are translationally 1 V we note that a similar, though more complicated, construction exists which finds an equivalent 1 V-AG (it "reduces" the AG, as discussed after Theorem 2.1).

In Lemma 2.1 we have made essential use of the noncircularity of the 1 S-AG. Observe however that the definition of noncircularity (see Sect. 1) is based on the conventional approach to attribute evaluation which requires all attributes to be computed. Another definition of noncircularity is possible, based on the translational approach to attribute evaluation. In this approach an AG would be circular only if the computation of the translation of a derivation tree depends on the elements of a cycle. We observe that Theorem 2.2 would stay true assuming just the translational version of noncircularity (by a slight change of proof).

3. Syntactic Permutations

In this section we will compare translations of 1 V-AGs with those of L-AGs. We will show (Theorem 3.1) that for any semantic domain D the class of translations defined by 1 V-AGs over D is equal to the class of translations defined by "syntactic permutations" of L-AGs over D. As a consequence these classes of AGs have the same class of output sets (Corollary 3.2). As a particular case, the same results hold for translations and output sets of 1 S-AGs and L-1 S-AGs, respectively.

We start by giving an intuitive feeling of the difference between the 1 V and the L properties. It is immediate that the L property is a special case of the 1 V property: for any production p of an L-AG G, BG_p is not only acyclic, as required by the 1 V property (Definition 2.4), but all its arcs run from left to right. It is easy to see, then, that if we permute the order of the nonterminals of the right-hand side of p, without modifying the dependencies among their attributes, the BG of the obtained production is isomorphic to BG_p and, thus, satisfies the 1 V property, but it does not necessarily satisfy the L property: there may be arcs running from right to left. It is also easy to see that if we permute the nonterminals of the right-hand side of a production p of a 1 V-AG according to any visiting sequence v_p for p (Definition 2.4), the obtained production satisfies the L property. As an example, permuting the production p of Example 2.1 according to the visiting sequence $\langle 1, 3, 2\rangle$ produces the production

$$p'\colon \langle\text{block}\rangle \rightarrow \textbf{begin}\ \langle\text{decls}_1\rangle\ \textbf{in}\ \langle\text{decls}_2\rangle\ \textbf{where}\ \langle\text{statlist}\rangle\ \textbf{end}$$

which satisfies the L property. Note that for readability, it would also be wise to change the terminals; in what follows we will not bother about terminals. Therefore, each 1 V-AG G_1 can be viewed as an L-AG G_2 whose productions have been permuted leaving the semantics unchanged. The relation between G_1 and G_2 can also be expressed by saying that their underlying CFGs are the input and output grammars of a syntax-directed translation scheme (sdts) [1]. The relation between 1 V-AGs and L-AGs via sdts was observed in [23].

As we have seen, the idea of syntactic permutation of an AG G is very simple. Therefore we will quickly introduce it, leaving a formal definition to the reader.

An AG G' is a *syntactic permutation* of an AG G if it can be obtained from G by permuting in some way the nonterminals of the right-hand side of each production of G without changing anything else. This means that every nonterminal F has in G' the same set of attributes as in G and every permuted production of G' has the same set of semantic rules as the corresponding production of G. Observe that, in particular, G' is over the same semantic domain as G.

A way of permuting the right-hand sides of the productions of an AG G can be specified by giving, for each production p: $F_0 \to w_0 F_1 w_1 \dots w_{n_p-1} F_{n_p} w_{n_p}$ of G, a sequence $\pi_p = \langle i_1, \dots, i_{n_p} \rangle$ which is a permutation of the sequence $1, \dots, n_p$. Permuting p according to π_p we obtain the production $F_0 \to w_0 F_{i_1} w_1 \dots w_{n_p-1} F_{i_{n_p}} w_{n_p}$. Let π be a set containing one such sequence π_p for each production p of G. The syntactic permutation G' of G obtained by permuting the productions of G according to the sequences in π is denoted by $\mathrm{perm}(G, \pi)$. It is easy to see that there is a one-to-one correspondence between derivation trees of G and $G' = \mathrm{perm}(G, \pi)$. If t is a derivation tree of G, we will indicate the corresponding permuted tree of G' by $t' = \mathrm{perm}(t, \pi)$. It should be clear that there is also a one-to-one correspondence between the nodes of t and t'. Furthermore, from the fact that the semantics of G' is the same as that of G it follows immediately that $DG(t)$ and $DG(t')$ are isomorphic. From this the following result is easy to show.

Lemma 3.1. *Let G and G' be two AGs such that $G' = \mathrm{perm}(G, \pi)$ for some π. For every complete derivation tree t of G, $T(G)(t) = T(G')(t')$, where $t' = \mathrm{perm}(t, \pi)$.*

From the previous lemma we obtain immediately that a syntactic permutation does not change the output set.

Corollary 3.1. *For every two AGs G and G' such that one is a syntactic permutation of the other,* $\mathrm{OUT}(G) = \mathrm{OUT}(G')$.

Proof. By Lemma 3.1 the ranges of $T(G)$ and $T(G')$ are equal. □

Let us now apply these results to 1V-AGs and L-AGs. We will denote the class of AGs consisting of all possible syntactic permutations of X-AG by PERM(X-AG).

Theorem 3.1. (*a*) *The class of* 1 *V-AGs is equal to the class of syntactic permutations of L-AGs, i.e.,* 1 *V-AG* = PERM(*L-AG*).

(*b*) *For every semantic domain D*, T(1 V-AG, D) = T(PERM(L-AG), D).

(*c*) *For every semantic domain D*, T(1 S-AG, D) = T(PERM(L-1 S-AG), D).

Proof. (a) (i) 1 V-AG ⊆ PERM(L-AG). It is sufficient to observe that, for any 1 V-AG G, if π contains a visiting sequence for each production of G, then $\mathrm{perm}(G, \pi)$ is an L-AG.

(ii) PERM(L-AG) ⊆ 1 V-AG. Observe that a syntactic permutation of a 1 V-AG is still a 1 V-AG.

(b) Immediate from (a).

(c) By (a) and the obvious observation that a syntactic permutation of a 1 S-AG is still a 1 S-AG we have that 1 V-1 S-AG = PERM(L-1 S-AG). By Theorem 2.2 we know that, for any semantic domain D, $T(1\text{V-}1\text{S-AG}, D) = T(1\text{S-AG}, D)$. □

It follows directly from Theorem 3.1 and Lemma 3.1 that the 1V-AG translations can be expressed as the composition of syntax-directed translations (on derivation trees) and L-AG translations (over the same semantic domain), and similarly for 1S-AG and L-1S-AG respectively. A formal statement of this relationship is left to the reader, cf. [23]. We only state the consequences for the corresponding classes of output sets.

Corollary 3.2. *For any semantic domain D,*
(*a*) $\text{OUT}(1\text{V-AG}, D) = \text{OUT}(\text{L-AG}, D)$
(*b*) $\text{OUT}(1\text{S-AG}, D) = \text{OUT}(\text{L-}1\text{S-AG}, D)$.

Proof. By Theorem 3.1 and Corollary 3.1. □

We finally observe that from Theorem 3.1(a) one might conclude that 1V-AGs are just "syntactically sugared" L-AGs, which never arise in practice except when one insists on having such unnecessary facilities as putting declarations after statements (Example 2.1). That this is not completely true is shown by the following example [Swierstra, private communication]. Suppose that in ⟨decls⟩ one may define procedures which recursively call each other; the corresponding production is p: ⟨decls⟩→⟨heading⟩⟨body⟩⟨decls⟩. The semantic rules needed to check that all procedure names used in the bodies are declared in the headings, will be one-visit with visiting sequence $v_p = \langle 1, 3, 2 \rangle$: first all headings are listed (from left to right) and then all bodies are checked (from right to left). It would be awkward to replace p by the permuted production ⟨decls⟩→⟨heading⟩⟨decls⟩⟨body⟩ which would result in a sequence of headings followed by a sequence of bodies in the wrong order!

4. Trees and Strings

In this section we will restrict attention to two particular semantic domains for AGs: the domains TREES and STRINGS. The reason we want to consider these two particular domains is that for those it is possible to show the existence of a relationship between translations (and output languages) of 1V-AGs and 1S-AGs, and similarly between L-AGs and L-1S-AGs. Actually, we will show that the class of translations defined by 1V-AGs over TREES is equal to the class of translations which are the composition of a translation defined by a 1S-AG over TREES and a deterministic top-down tree transduction (Theorem 4.1). This relationship, with the addition of already known results, allow us to express the class of output languages of 1V-AGs (and L-AGs) in terms of known classes of languages and translations (viz. the IO macro languages and the top-down tree transductions).

In what follows we will consider the usual kind of labeled ordered trees (as e.g. in [26, 11]). A tree with root labeled σ and direct subtrees $t_1, \ldots, t_n$ will be denoted by $\sigma[t_1, \ldots, t_n]$.

We informally define TREES as the semantic domain containing all possible labeled ordered trees and whose only operation is "top-concatenation" of trees. This means that the values of the attributes of any AG over TREES are trees and that its semantic rules will have the form $a \leftarrow t(a_1, \ldots, a_m)$, where $t = t(a_1, \ldots, a_m)$ is a labeled tree in which some of its leaves carry names of attributes $a_1, \ldots, a_m$ in that order (where possibly $a_i = a_j$ for $i \neq j$). Through this semantic rule the value of a is computed by "concatenating" the values of $a_1, \ldots, a_m$ (which are trees) with "top" tree t, i.e. if a_i has value t_i, then a will become the tree $t(t_1, \ldots, t_m)$ obtained from t by substituting t_i for every occurrence of a_i, $i \in [1, m]$.

Formally, TREES is the semantic domain (Ω, Φ) where $\Omega = \{V\}$ is the singleton containing the set V of all labeled ordered trees (with labels taken from some fixed denumerable set of symbols) and Φ contains all "derived functions", i.e. for every tree $t = t(a_1, \ldots, a_m)$, $m \geq 0$, Φ contains the function f_t: $V^m \rightarrow V$ such that $f_t(t_1, \ldots, t_m) = t(t_1, \ldots, t_m)$. For a still more formal description see [7].

The domain STRINGS is similarly defined to contain all possible strings with string concatenation as only operation. This means that, for any AG over STRINGS the attributes will take strings as values and the semantic rules will have the form $a \leftarrow \phi = w_1 a_1 w_2 \ldots w_m a_m w_{m+1}$, where $a_1, \ldots, a_m$ are attribute names and w_i is a string, $i \in [1, m+1]$. The value of a is computed by substituting in ϕ the corresponding values (strings) for the attribute names $a_1, \ldots, a_m$.

Formally, as for TREES, STRINGS is the semantic domain (Ω, Φ) where Ω consists only of the set of all strings (over some denumerable alphabet) and Φ consists of all functions $f(v_1, \ldots, v_m) = w_1 v_1 w_2 \ldots w_m v_m w_{m+1}$. The elements of Φ are called "simple word functions" in [10].

From the way TREES and STRINGS are defined it should be clear that, for every class X-AG of attribute grammars discussed in this paper,

$$T(X\text{-AG, STRINGS}) = T(X\text{-AG, TREES}) \circ \text{yield}$$

and

$$\text{OUT}(X\text{-AG, STRINGS}) = \text{yield}(\text{OUT}(X\text{-AG, TREES})),$$

where yield is the usual mapping which associates to each tree its sequence of leaf labels.

In the following we will consider the class of IO macro languages (denoted by IO) and the class of IO macro tree languages (denoted by IOT). Precise definitions of the first can be found in [15] and of the second in [13]. Note that yield(IOT) = IO.

A relationship between IO and output languages of AGs was recently established in [10]. This relationship can be expressed as follows.

(*) $$\text{OUT(L-1S-AG, STRINGS)} = \text{IO}.$$

Using Corollary 3.2 we can extend this result immediately to OUT(1S-AG, STRINGS) = IO. Thus, in (*), the L condition can be dropped. It is easy to see, slightly modifying the proof in [10], that an analogous result can be shown for TREES instead of STRINGS, that is, OUT(1S-AG, TREES) = OUT(L-1S-AG, TREES) = IOT. What we will do in the remaining part of this section is to show that OUT(1V-AG, TREES) is a larger class than OUT(1S-AG, TREES), and thus the same for L and L-1S respectively (Corollary 3.2). In fact we will prove that the class OUT(1V-AG, TREES) can be obtained by applying deterministic top-down tree transductions (*DT*, see [26, 27, 11]) to IOT. That is OUT(1V-AG, TREES) $= DT$(OUT(1S-AG, TREES)) $= DT$(IOT). Consequently, for the domain STRINGS, we will have OUT(1V-AG, STRINGS) $=$ yield(DT(IOT)) which shows that in (*) the 1S condition cannot be replaced by the 1V condition. We will prove these results by showing an even stronger one (Theorem 4.1) on translations, viz. T(1V-AG, TREES) $= T$(1S-AG, TREES) $\circ DT$. Before proving it we need to give a short definition of *DT*.

Definition 4.1. A *total deterministic top-down tree transducer* (denoted by *DT*) is a 5-tuple $(\Sigma, \Delta, Q, q_0, R)$ where Σ and Δ are the input and output alphabets, Q is the set of states, q_0 is the initial state and R is a set of rules of the form

$$(**) \qquad q(\sigma[x_1, \dots, x_k]) \to t$$

where $\sigma \in \Sigma$, $q \in Q$, k is the number of sons of $\sigma (k \geq 0)$, and t is a tree of the form $t(q_1(x_{i_1}), \dots, q_m(x_{i_m}))$, $m \geq 0$, i.e. some of its leaves are labeled by couples of the form $q_j(x_{i_j})$, where $q_j \in Q$ and $i_j \in [1, k]$, and all other nodes of t are labeled by elements of Δ. We require that, for each $\sigma \in \Sigma$ and $q \in Q$, R contains one rule of the form (**). □

Instead of formally defining the way a *DT* processes an input tree to produce the corresponding output tree, we only give an example.

Example. Assume that a *DT M* has a rule $q(\sigma[x_1, x_2, x_3]) \to t$, where

$$t = Z_0[Z_1[Z_3, q_2(X_3)], Z_2[q_1(X_2), q_2(X_3)]]$$

as shown in Fig. 4, $Z_i \in \Delta$. The effect of the application of this rule on an input tree, starting in state q, is shown in Fig. 5 (where σ_i is the root of t_i). □

Thus, for every input tree t (over Σ) and state $q \in Q$, a *DT M* produces an output tree (over Δ) as a result of processing t in a top-down fashion starting at the root of t in state q. This output tree will be called the *q-transduction* of t (by M). It should be clear that the rule (**) in Definition 4.1 (recursively) defines the q-transduction of any tree $\sigma[t_1, \dots, t_k]$ to be the result of substituting the q_j-transduction of t_{i_j} for $q_j(x_{i_j})$ in t. From this it can be seen that a total deterministic top-down tree transducer is very close to an OnlyS-AG over the domain TREES (each state q being an attribute of each node with the q-transduction of the subtree rooted at that node as its value). In fact the following result can easily be proved.

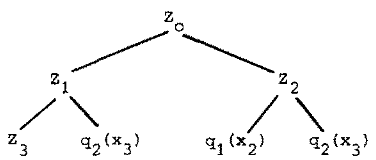


Fig. 4

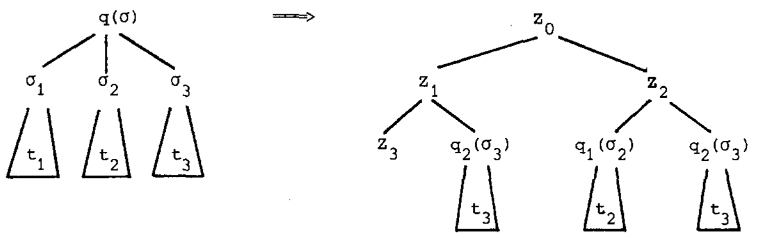


Fig. 5

Proposition 4.1. *The class of tree languages obtained by applying DT-transductions to the class RECOG of recognizable tree languages is equal to the class of output languages defined by OnlyS-AGs over the domain TREES, that is*

$$DT(\text{RECOG}) = \text{OUT}(\text{OnlyS-AG}, \text{TREES}). \quad \square$$

We point out that the classes of translations defined by DT and OnlyS-AG over TREES are also closely related.

Finally we will show the relationship between translations of 1V-AGs and 1S-AGs. The intuitive idea on which this relationship is based can be expressed as follows. The 1V-property implies that, during attribute evaluation on any derivation tree t, the s-attributes of any node n of t can be computed "simultaneously" (note that the same holds for i-attributes). Thus, a 1S-AG could simulate many s-attributes of a 1V-AG by computing them "simultaneously" and then tupling them. But then, clearly, a selection operation should be available to select the s-attributes from the tuple when needed. Therefore, for any semantic domain D, we could say that $T(\text{1V-AG}, D) \subseteq T(\text{1S-AG}, D')$, where D' is obtained by augmenting D with the operations of tupling and selection. Since in our case D is the domain of TREES, we clearly have the tupling operation available, but not the selecting operation. For this reason we are able to decompose the translation of a 1V-AG into the translation of a 1S-AG, which performs the tupling but does the selection only symbolically, followed by the translation of a DT, which performs the selections. Equality of the two classes of translations comes from the fact that $T(\text{1V-AG}, \text{TREES})$ is closed under composition with DT-transductions.

Theorem 4.1. $T(\text{1V-AG}, \text{TREES}) = T(\text{1S-AG}, \text{TREES}) \circ DT$

Proof. (a) $T(\text{1V-AG}, \text{TREES}) \subseteq T(\text{1S-AG}, \text{TREES}) \circ DT$. We will show that for every 1V-AG G over TREES we can construct a 1S-AG G_1 over TREES and a DT M such that $T(G) = T(G_1) \circ M$.

Construction 1 (of G_1)

We assume that, for each nonterminal of G, its s-attributes have a fixed (but arbitrary) order.

(1) G_1 has the same underlying CFG as G.

(2) To every nonterminal F, G_1 associates the same set of i-attributes as G and only one s-attribute, which we will denote $s(F)$.

(3) For each production p: $F_0 \to w_0 F_1 w_1 \dots F_{n_p} w_{n_p}$ of G, consider the set r_p of semantic rules associated to p by G. From r_p we will first define a new (auxiliary) set of rules r'_p and then, from this set, we will construct the set r_{1p} of semantic rules associated to p by G_1.

The construction of r'_p is as follows. For every semantic rule $a \leftarrow t(a_1, \dots, a_m)$ in r_p the corresponding rule of r'_p is $a \leftarrow t'$, where t' is obtained from t by substituting for a_i the tree $k[s(F_j)]$, if a_i is (in G) the k-th s-attribute of F_j, $j \in [1, n_p]$ (thus if a_i is in $I(F_0)$, it stays as it is).

The integer k is used as a (new) symbol (with only one son) and represents the operation of selecting the k-th element of a tuple. As we have already said, G_1 cannot perform selection itself, but leaves messages (selection symbols k) about where the selection has to be performed. Interpreting these messages will be the only task of the DTM.

The construction of r_{1p} from r'_p simply consists of replacing all the rules in r'_p defining the s-attributes of F_0 (in G) by a single semantic rule defining the only s-attribute of F_0 in G_1, as follows. If F_0 has in G the d s-attributes $s_1, \dots, s_d$ and the semantic rules defining them are $s_1 \leftarrow t_1, \dots, s_d \leftarrow t_d$, then the semantic rule for $s(F_0)$ in r_{1p} is $s(F_0) \leftarrow c_d[t_1, \dots, t_d]$. That is, the rule is $s(F_0) \leftarrow t$, where t is produced by tupling trees $t_1, \dots, t_d$ using symbol c_d, which is a (new) symbol representing the operation of tupling d elements.

Construction 2 (of M)

The input alphabet Σ of M is the output alphabet of G_1 and its output alphabet Δ is the output alphabet of G. Note that, if we denote by K the set of all selection symbols, and by C the set of all tupling symbols, as used in Construction 1, then $\Sigma = \Delta \cup K \cup C$.

We will now give the set of rules and the states of M. By Definition 4.1 of DT we should describe M as total, but looking at Construction 1, we see that all trees produced by G_1 satisfy the following condition:

each node labeled by selection symbol k has only one direct descendant and this descendant is labeled by a tupling symbol c_d such that $k \leqq d$ (and d is the number of direct descendants of c_d).

Taking this fact into account we give only those rules of M which may actually be applied, and the remaining rules can be taken arbitrarily.

M has a set Q of states $q_0, q_1, \dots, q_r$, where r is the maximal element of the set K. The set R of rules of M is defined as follows.

(i) For every $\sigma \in \Sigma - K - C$, R contains the rule

$$q_0(\sigma[x_1, \dots, x_m]) \to \sigma[q_0(x_1), \dots, q_0(x_m)]$$

where m is the number of sons of σ.

(ii) For every $k \in K$, R contains the rule

$$q_0(k[x_1]) \to q_k(x_1).$$

(iii) For every $c_d \in C$, R contains the rule

$$q_k(c_d[x_1, \ldots, x_d]) \rightarrow q_0(x_k)$$

for all states q_k such that $k \in [1, d]$.

Rules of type (i) clearly do not do anything to the input tree. The operation of selection which is, as we said before, all M does, is performed by the successive application of a rule of type (ii) and a rule of type (iii): in the first step M stores the integer k in its finite control and in the next step it selects the k-th subtree.

Finally, if the designated s-attribute of the initial nonterminal of G is its j-th s-attribute, then q_j is the initial state of M.

This ends the two constructions. We will not give a complete proof of their correctness, but we will indicate how the proof may be organized. Before the statement of this theorem we have mentioned that at the basis of the simulation of a 1 V-AG by a 1 S-AG is the fact that in a 1 V attribute evaluation all s-attributes of a node are computed "simultaneously". Now we can make this statement more clear. By Construction 1 we know that G and G_1 have the same underlying CFG. Considering any production p of this CFG, it is easy to see that the DG_p of G_1 can be obtained from the DG_p of G through the obvious transformation of considering the set of s-attributes of each nonterminal in p as one s-attribute (clearly, in the transformation, some arcs may overlap). Consequently the brother-graphs BG_p are exactly the same in G and G_1, and so attribute evaluation can be carried out according to exactly the same static algorithm (Algorithm 2.1), i.e., with the same visiting sequences. Therefore, for a given derivation tree t, the static algorithm gives rise to two corresponding computation sequences $c = \langle st_1, \ldots, st_k \rangle$ and $c_1 = \langle st'_1, \ldots, st'_k \rangle$, of G and G_1 respectively, where the computation steps st_j and st'_j consist of the evaluation of all i-attributes (or of all s-attributes) of the same node of t. The concept of corresponding computation sequences of G and G_1 is at the basis of the proof we want to sketch. For a complete derivation tree t we clearly have a complete computation sequence $c = \langle st_1, \ldots, st_k \rangle$ on t, which computes the translation of t, but for the purpose of the proof we also consider every prefix c' of c (i.e. $c' = \langle st_1, \ldots, st_m \rangle$, $m < k$) to be a computation sequence. The "output" of c' will be the set of values of the attributes computed in step st_m.

We want to use induction on the length of corresponding computation sequences c and c_1 of G and G_1 respectively, and show, roughly speaking, that the transduction of M on the output of c_1 is equal to the output of c. We now make some observations which will enable us to formulate this induction hypothesis in a precise way. In comparing c and c_1 (on some derivation tree t) we distinguish two cases:

(i) The last steps of c and c_1 compute the s-attribute(s) of a node n of t.

(ii) The last steps of c and c_1 compute the i-attributes of a node n of t.

In both cases something has to be said about how M works on the output of c_1, because we know only how M should work on the output of a complete computation sequence of G_1, but not yet on its prefixes.

Case (i). From Construction 1 we know that the output of c_1 will be a tree whose root is labeled by a tupling symbol c_d having d subtrees, each of which represents one s-attribute of node n in G. However, the output of c consists directly of the values of the d s-attributes of n. This means that we must run M d times on the output of c_1 with initial states $q_1, \dots, q_d$, respectively, to obtain the desired values of the d s-attributes of n in G.

Case (ii). From Construction 1 we know that each nonterminal has the same number of i-attributes in G and G_1. Assume that node n has m i-attributes. Then we will run M on each of them, but what should the starting state be? Again from Construction 1 we can immediately see that the values of any i-attribute of G_1 can only be trees whose roots are not a c_d (tupling) symbol. Furthermore, from Construction 2 we know that states different from q_0 are needed only when M is scanning a node labeled by a c_d symbol. Thus, for each of the m runs of M the start state should be q_0.

As we said previously, the q_i-transduction of a tree t by M is the result of "running" M on t with starting state q_i.

In order to prove this part of the theorem it would suffice to prove the following (by induction on the length of the computation sequences):

for every complete derivation tree t of G and G_1, and for every couple of corresponding computation sequences c and c_1 on t the following holds:

(i) if the last step of c computes the d s-attributes of node n of t, with values $t_1, \dots, t_d$, then t_i is equal to the q_i-transduction by M of the output of c_1, for every $i \in [1, d]$;

(ii) assume that the last steps of c and c_1 compute the m i-attributes of node n of t, and let their values in c be $t_1, \dots, t_m$ and in c_1 be $t'_1, \dots, t'_m$; then the q_0-transduction by M of t'_i is equal to t_i, $i \in [1, m]$.

The induction proof simply has to look closely how right-hand sides of semantic rules of G_1 are built and how M works on them.

(b) $T(1\,\text{S-AG}, \text{TREES}) \circ DT \subseteq T(1\,\text{V-AG}, \text{TREES})$.
Given a 1 S-AG G (defined over TREES) and a DT M, we will give a construction of a 1 V-AG G_1 (defined over TREES) such that $T(G_1) = T(G) \circ M$. We assume G to be 1 V, without loss of generality by Theorem 2.2.

Construction of G_1

(1) G_1 has the same underlying CFG as G.

(2) Let M have states $q_1, \dots, q_n$, then each attribute a of G becomes n attributes $a^1, \dots, a^n$ in G_1. Thus, if nonterminal F has m i-attributes and one s-attribute in G, then, in G_1, F has nm i-attributes and n s-attributes. Intuitively, attribute a^i of G_1 will hold the q_i-transduction by M of the value of the attribute a of G.

(3) For each production p in G, consider the set r_p of semantic rules associated to p by G. We construct the set r_{1p} of semantic rules associated to p by G_1 by deriving, from each rule in r_p, n rules in the following way. Let rule sr: $a_0 \leftarrow t$ be in r_p, where $t = t(a_1, \dots, a_m)$ for $m \geqq 0$. From rule sr we build rules $sr_1, \dots, sr_n$ of G_1 which define the n attributes $a_0^1, \dots, a_0^n$ respectively, as follows. For every $i \in [1, n]$ let t'_i be the tree resulting from "running" M on t with starting state q_i. In general, some of the leaves of t'_i will be couples $q_k(a_x)$, where q_k is a state of M and a_x is an attribute in $\{a_1, \dots, a_m\}$. From t'_i we derive the tree t_i by replacing

each leaf $q_k(a_x)$ by the new attribute name a_x^k. Thus, rule sr_i of r_{1p} is $a_0^i \leftarrow t_i$. Note that here we use the fact that M is total.

(4) The designated s-attribute of the initial nonterminal Z of G_1 is a^i, where a is the (only) s-attribute of Z in G and q_i is the initial state of M.

This ends the construction of G_1. From the construction we can see that for every production p the BG_p of G_1 will be a subgraph of the BG_p of G (it is not the same graph in general because M may be deleting certain branches). Thus, from the fact that G is 1V it follows that G_1 is also 1V. From this we also see that the concept of corresponding computation sequences of G and G_1 applies as we described it in the first half of this proof.

Intuitively, if we consider two corresponding computation sequences c and c_1, the output of c_1 can be viewed as the n transductions that M can perform on the output of c with starting states $q_1, \ldots, q_n$ respectively. More formally the following should be proved:

for any couple of corresponding computation sequences c and c_1 on a complete derivation tree t, if the output of c consists of m values $t_1, \ldots, t_m$, then correspondingly the output of c_1 consists of nm values t_i^j with $i \in [1, m]$ and $j \in [1, n]$, such that t_i^j is the q_j-transduction of t_i by M.

Clearly this can be proved again using induction on the length of corresponding computation sequences. The formal proof is left to the reader. □

It should be clear that in the construction given in part (b) of the preceding proof, there was no need of restricting G to be 1S. This means that in the same way it can be shown that the class of 1V-AG translations (over TREES) is closed under composition with DT.

We can immediately extend the result of Theorem 4.1 to the domain STRINGS by applying the yield operation.

Corollary 4.1. T(1V-AG, STRINGS) = T(1S-AG, TREES) ∘ DT ∘ yield.

Obviously, analogous results hold for output languages. We summarize all results on output languages in the next theorem (including the one of Duske e.a. [10]).

Theorem 4.2. (*a*) *For the domain STRINGS,*

(*i*) OUT(1S-AG, STRINGS) = OUT(L-1S-AG, STRINGS) = IO,

(*ii*) OUT(1V-AG, STRINGS) = OUT(L-AG, STRINGS) =
= yield(DT(OUT(1S-AG, TREES))) = yield(DT(IOT)).

(*b*) *For the domain TREES,*

(*i*) OUT(1S-AG, TREES) = OUT(L-1S-AG, TREES) = IOT,

(*ii*) OUT(1V-AG, TREES) = OUT(L-AG, TREES) =
= DT(OUT(1S-AG, TREES)) = DT(IOT).

Proof. Points $a(i)$ and $b(i)$, as already said before, follow from the results of [10] and Corollary 3.2. Points $a(ii)$ and $b(ii)$ follow from Theorem 4.1, Corollary 4.1 and point $b(i)$. □

Note that, in Theorem 4.2, we may also assume the DT to be partial because the domain of a partial DT is recognizable [26] and IOT is closed under intersection with recognizable tree languages [13]. On the other hand the DT could also be restricted to be linear (i.e. noncopying) because the DT constructed in the first part of the proof of Theorem 4.1 is linear.

We end this section with two more general observations on the result OUT(L-AG, TREES) = DT(IOT).

In the literature, the study of DT was motivated from syntax-directed translation, i.e. AGs with s-attributes only (cf. Proposition 4.1), whereas macro grammars were motivated by the possibility to handle certain context-sensitive restrictions on the context-free syntax, which can be handled by AGs with i-attributes only. It is more or less to be expected, but still surprising, that the combination of s- and i-attributes in L-AGs (or 1V-AGs) leads to such a simple combination of DT and IOT.

Fischer [15] suggests the investigation of "multiple-value macro grammars" in which a nonterminal generates a finite sequence of strings rather than just one. Since, in the proof of OUT(L-1S-AG, STRINGS) = IO in [10], the string generated by a nonterminal corresponds to the value of the one s-attribute, one would expect multiple-value IO macro grammars to correspond to L-AGs with any number of s-attributes. In fact, an obvious generalization of the above-mentioned proof shows that OUT(L-AG, STRINGS) = multiple-value IO and similarly OUT(L-AG, TREES) = multiple-value IOT. Thus, by Theorem 4.2, multiple-value IO = yield(DT(IOT)). Multiple-value macro grammars have been investigated in an algebraic framework in [2].

5. Comparison of Power

In this section we want to compare the formal power of the restrictions on attribute grammars which we have considered in this paper. To do so we show the correctness of the diagrams of Fig. 6 (which are the same as in Fig. 1).

In Fig. 6(a) every label X should be replaced by $T(X\text{-AG}, D)$ and in Fig. 6(b) by OUT(X-AG, D). For any fixed domain D, the inclusions shown in Fig. 6 are either obvious or follow from Theorem 2.2 (1S $\subseteq$ 1V in Fig. 8(a)) and Corollary 3.2 (L = 1V and L1S = 1S in Fig. 6(b)). In the rest of the section we want to show that there is a semantic domain D, viz. TREES, such that also the proper inclusions

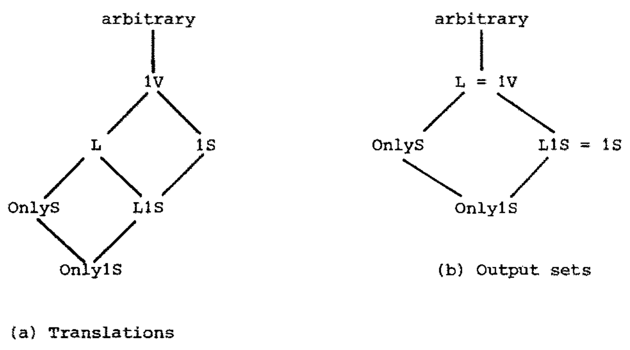


Fig. 6, a, b

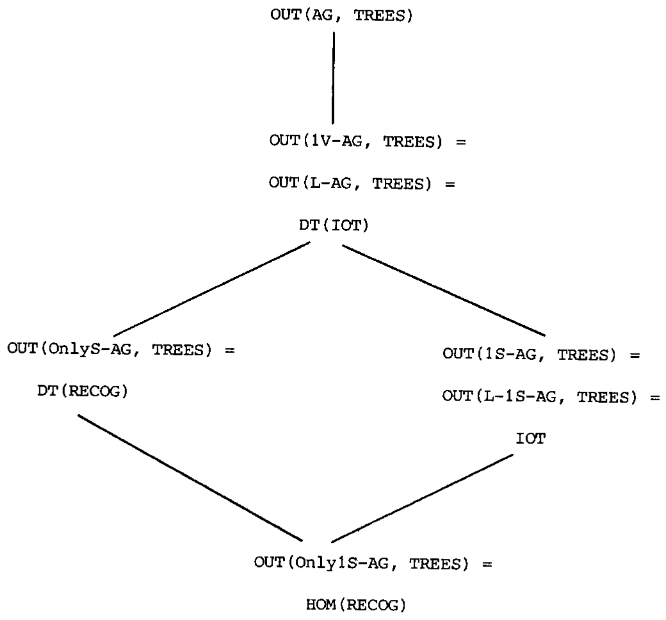


Fig. 7

and incomparabilities hold as shown in Fig. 6. This implies that an upward line in the diagrams represents "more power", whereas unconnected classes have "incomparable power". It suffices to prove the following.

(1) OUT(OnlyS-AG, TREES) − OUT(1S-AG, TREES) $\neq \emptyset$.

(2) OUT(L-1S-AG, TREES) − OUT(OnlyS-AG, TREES) $\neq \emptyset$.

(3) OUT(AG, TREES) − OUT(1V-AG, TREES) $\neq \emptyset$.

(4) T(1S-AG, TREES) − T(L-AG, TREES) $\neq \emptyset$.

In the proof of (1)–(3) we will use the characterization of the classes of output languages over TREES in terms of tree transducers and tree grammars as stated in Theorem 4.2 and Proposition 4.1, see Fig. 7 which corresponds to Fig. 6(b). For completeness we added OUT(Only1S-AG, TREES) = HOM(RECOG), where HOM is the class of tree homomorphisms, i.e. DT with one state only (special case of Proposition 4.1).

To show (1). Let T_π be the set of all binary trees with all nodes labeled π, and consider the tree language $L_0 = \{\sigma[t, \bar{t}] \mid t \in T_\pi\}$ where σ is a symbol and $\bar{t}$ is equal to t except that it is labeled by $\bar{\pi}$. Clearly L_0 is the output language of the following OnlyS-AG with two s-attributes s_1 and s_2 (one to hold t and the other to hold $\bar{t}$).

$$\begin{array}{ll} Z \to A & s_1(Z) \leftarrow \sigma[s_1(A), s_2(A)] \\ A \to A_1 A_2 & \begin{cases} s_1(A) \leftarrow \pi[s_1(A_1), s_1(A_2)] \\ s_2(A) \leftarrow \bar{\pi}[s_2(A_1), s_2(A_2)] \end{cases} \\ A \to a & \begin{cases} s_1(A) \leftarrow \pi \\ s_2(A) \leftarrow \bar{\pi}. \end{cases} \end{array}$$

We now argue that L_0 is not in OUT(1S-AG, TREES) = IOT. In general, assuming that an IO macro tree grammar G exists for a language of the form $\{\sigma[t, \bar{t}] \mid t \in L\}$, where L is some tree language, it is quite easy to change G slightly such that it is linear (i.e., in each right-hand side of a production at most one nonterminal occurs) and "iterative" (i.e. if a nonterminal occurs in a right-hand side, then it occurs at its root). Such grammars are called ILBT grammars [9] or coregular tree grammars [3]. By erasing all barred symbols in (the changed) G, it then follows that L is also an ILBT language. Thus, if $\{\sigma[t, \bar{t}] \mid t \in L\} \in \mathrm{IOT}$ then $L \in \mathrm{ILBT}$. A similar result (without bar on the second t) for OI macro tree languages is shown in [4]. Since ILBT grammars are very similar to register programs [9, 5] and one register program cannot compute all expressions [24], it follows that T_π is not in ILBT [3]. Hence, by the above general result, L_0 is not in IOT and cannot be produced by a 1S-AG over TREES.

To show (2) and (3) we use the "path-approach", i.e. we determine the classes of path-languages corresponding to the involved classes of tree languages. For a tree t, let $\pi(t)$ be the set of paths through t which lead from the root to some leaf. A path is coded by a string of symbols σ_k meaning that "at node labeled σ the k-th branch is taken" (k is omitted for a leaf). For a tree language L and a class of tree languages X, $\pi(L) = \bigcup \{\pi(t) \mid t \in L\}$ and $\pi(X) = \{\pi(L) \mid L \in X\}$. By Figs. 8(b) and 9, (2) is equivalent to $\mathrm{IOT} - DT(\mathrm{RECOG}) \neq \emptyset$. Since $\pi(\mathrm{IOT})$ is equal to the class CF of context-free languages [14] and $\pi(DT(\mathrm{RECOG}))$ is the class of regular languages [26], (2) immediately follows.

To show (3) we use the equality OUT(1V-AG, TREES) = DT(IOT) and observe that, on paths, a (total deterministic) top-down tree transducer works like a gsm (generalized sequential machine): to a rule $q(\sigma[x_1, \ldots, x_n]) \to t$ of the DT correspond gsm rules $(q, \sigma_k) \to (q', v)$ and $(q, \sigma_k) \to (q_f, w)$, where v is any path through t from the root down to $q'(x_k)$, w is any path through t from the root to a leaf labeled by an output symbol, and q_f is a final state which consumes the rest of the input without producing more output. In this way it can be proved that, for any class X of tree languages, $\pi(DT(X)) \subseteq GSM(\pi(X))$. Hence, since $\pi(\mathrm{IOT}) = \mathrm{CF}$ and CF is closed under gsm mappings, we obtain that $\pi(DT(\mathrm{IOT})) = \mathrm{CF}$. It now remains to give an example of an AG over TREES whose output language has a non context-free path language. It is easy to see that the following AG produces the path language $\{ww \mid w \in \{A_1, B_1\}^* \cdot \{A, B\}\}$. It uses s-attribute s_1 to hold the (monadic) derivation tree, uses i-attribute i to pass this value to the leaf and then puts the derivation tree on top of itself using s-attribute s_2. Let $X, Y \in \{A, B\}$.

$$\begin{array}{lll}
Z \to X & s_2(Z) \leftarrow s_2(X); & i(X) \leftarrow s_1(X) \\
X \to Y & s_j(X) \leftarrow X[s_j(Y)],\ j = 1, 2; & i(Y) \leftarrow i(X) \\
X \to e & s_1(X) \leftarrow X; & s_2(X) \leftarrow X[i(X)].
\end{array}$$

Correctness of the diagram of Fig. 6(b) has now been proved. For readers interested in STRINGS we note that yield(DT(RECOG)) and IO are incomparable; the proper inclusion of yield(DT(IOT)) in OUT(AG, STRINGS) is open, but is very probably true.

Finally we prove (4) by providing a counter-example, with a combinatorial proof. In what follows we will use the string $\sigma_1\sigma_2\ldots\sigma_{n-1}\sigma_n$ to denote the (monadic) tree $\sigma_1[\sigma_2[\ldots\sigma_{n-1}[\sigma_n]\ldots]]$.

Let the (underlying) CFG G_0 have productions $Z\to AB$, $A\to A|B|e$ and $B\to A|B|e$. Obviously the derivation trees of G_0 are of the form $Z[Aw_1e, Bw_2e]$ with $w_1, w_2 \in \{A, B\}^*$. Our counter-example is the translation which transforms $Z[Aw_1e, Bw_2e]$ into the monadic tree Aw_1Bw_2. It is easy to see that a 1S-AG can define this translation as follows: it would first compute (the right branch) Bw_2 through the only allowed s-attribute, then pass this value down the left branch through an i-attribute, and finally compute Aw_1x coming up the left branch (where x is the value of the i-attribute, i.e. Bw_2) using the s-attribute, cf. the similar example in the proof of (3). Assume now that there is an L-AG G which defines this translation. Then G has to have underlying CFG G_0. Consider the production $Z\to AB$ and the semantic rule defining the designated s-attribute $s(Z)$. Consider also a derivation tree $t=Z[Aw_1e, Bw_2e]$ with nonempty w_1 and w_2. Because the output is monadic, we can distinguish the following two cases.

(a) The semantic rule is $s(Z)\leftarrow cs_1(A)$ where $c\in\{A,B\}^*$ and $s_1\in S(A)$. This case has an easy solution. In fact, because of the L condition we know that $s_1(A)$ cannot depend on any attribute of the right-branch Bw_2e. Therefore, from t we can build a derivation tree t_1 for which G will fail, by simply changing w_2.

(b) The semantic rule $s(Z)\leftarrow cs_2(B)$ where $c\in\{A,B\}^*$ and $s_2\in S(B)$. Actually also here we would have two cases: $s_2(B)$ depends on attributes in Aw_1e or it does not. Since the latter case can be handled as in (a), we consider the first. Clearly, $s_2(B)$ can only depend on an attribute of Aw_1e through some i-attribute of B, call it $i(B)$, which depends on some s-attribute of A. Thus the (monadic!) value of $s_2(B)$ can be represented as $s_2(B)=d_2d_1$, where $d_2\in\{A,B\}^*$ is the result of attribute evaluation on Bw_2e only, and $d_1=i(B)\in\{A,B\}^+$, which depends on attribute evaluation on Aw_1e only. Thus, if we change Bw_2e, only part d_2 of $s_2(B)$ can change. The translation of derivation tree t by G is $s(Z)=cd_2d_1=Aw_1Bw_2$. Therefore, we can find a derivation tree t_1 for which G fails simply changing the last symbol of w_2 in t. This proves (4).

Conclusion

We consider the results of [10] and this paper as the start of a more thorough investigation into the power of attribute grammars viewed as tree transducers. We think that the large amount of knowledge now available concerning top-down tree transducers and macro grammars allows one to attack all the usual formal language theoretic problems concerning this new type of tree transducer. A few questions which came to our mind are the following.

Is it possible to identify the class of output languages of arbitrary attribute grammars in terms of known concepts?

Do the classes of translations and output languages of arbitrary attribute grammars form a proper hierarchy with respect to the maximal number of visits to subtrees during attribute evaluation?

What kind of sequential machines can realize the AG translations or accept their output languages? It is rather easy to find an automaton recognizing the L-AG output languages, i.e. yield(DT(IOT)), taking a variant of the attributed pushdown transducer of [20].

After submission of this paper new results on the formal power of AG appeared in [25, 8, 12]. In particular, in [25] the second question above was answered in the affirmative.

References

1. Aho, A.V., Ullman, J.D.: The theory of parsing, translation and compiling, Vols. 1 and 2; Englewood Cliffs, N.J., Prentice-Hall 1972
2. Arnold, A.: Systèmes d'equations dans le magmoide; ensembles rationels et algébriques d'arbres. Thèse, Université de Lille 1977
3. Arnold, A., Dauchet, M.: Transductions de forêts reconnaissables monadiques; forêts corégulières. RAIRO IT **10**, 5-28 (1976)
4. Arnold, A., Dauchet, M.: Un théorème de duplication pour les forêts algébriques. JCSS **13**, 223-244 (1976)
5. Asveld, P.R.J., Engelfriet, J.: Extended linear macro grammars, iteration grammars, and register programs. Acta Informat. **11**, 259-285 (1979)
6. Bochmann, G.V.: Semantic evaluation from left to right. CACM **19**, 55-62 (1976)
7. Chirica, L.M., Martin, D.F.: An algebraic formulation of Knuthian semantics. Proc. 17th Ann. Symp. on Foundations of Computer Science, pp. 127-136, IEEE, Houston, Texas, 1976
8. Courcelle, B., Franchi-Zannettacci, P.: On the expressive power of attribute grammars. Proc. of the 21st FOCS, IEEE, pp. 161-172, 1980
9. Downey, P.J.: Formal languages and recursion schemes. Ph. D. Thesis, Report TR-16-74, Cambridge, Mass.: Harvard University 1974
10. Duske, J., Parchmann, R., Sedello, M., Specht, J.: IO-macrolanguages and attributed translations. Information and Control **35**, 87-105 (1977)
11. Engelfriet, J.: Bottom-up and top-down tree transformations - a comparison. Math. Systems Theory **9**, 198-231 (1975)
12. Engelfriet, J.: Some open questions and recent results on tree transducers and tree languages. In: Formal language theory; perspectives and open problems. (R.V. Book, ed.), New York: Academic Press 1980
13. Engelfriet, J., Meineche Schmidt, E.: IO and OI; JCSS **15**, 328-353 (1977); JCSS **16**, 67-99 (1978)
14. Engelfriet, J., Slutzki, G.: Bounded nesting in macro grammars. Information and Control **42**, 157-193 (1979)
15. Fischer, M.J.: Grammars with macro-like productions. Ph. D. Thesis, Harvard University 1968 (also: Proc. 9-th Symp. on SWAT, pp. 131-142, 1968)
16. Jazayeri, M., Ogden, W.F., Rounds, W.C.: The intrinsically exponential complexity of the circularity problem for attribute grammars. CACM **18**, 697-706 (1975)
17. Kennedy, K., Warren, S.K.: Automatic generation of efficient evaluators for attribute grammars. Conf. Record of the Third Symposium on Principles of Programming Languages, pp. 32-49 (1976)
18. Knuth, D.E.: Semantics of context-free languages. Math. Syst. Theory **2**, 127-145 (1968). Correction: Math. Systems Theory **5**, 95-96 (1971)
19. Koster, C.H.A.: Affix grammars. Proc. IFIP Working Conf. on Algol 68 implementation; Amsterdam: North-Holland 1971
20. Lewis, P.M., Rosenkrantz, P.J., Stearns, R.E.: Attributed translations. JCSS **9**, 279-307 (1974)
21. Maibaum, T.S.E.: A generalized approach to formal languages. JCSS **8**, 409-439 (1974)
22. Mayoh, B.H.: Attribute grammars and mathematical semantics. Report DAIMI PB-80, Aarhus University 1978

23. Parchmann, R.: Grammatiken mit Attributschema und Zweistufige Auswertung attributierter Grammatiken; Schriften zur Informatik und angewandte Mathematik, Bericht No. 46, Technische Hochschule Aachen 1978
24. Paterson, M.S., Hewitt, C.E.: Comparative schematology. Record of Project MAC Conf. on concurrent systems and parallel computation, pp. 119–128, ACM, New York 1970
25. Riis, H., Skyum, S.: *K*-visit attribute grammars; DAIMI PB-121, Aarhus University, 1980; Math. Systems Theory (in press, 1981)
26. Rounds, W.C.: Mappings and grammars on trees. Math. Systems Theory **4**, 257–287 (1970)
27. Thatcher, J.W.: Generalized^2 sequential machine maps; JCSS **4**, 339–367 (1970)

Received May 5, 1980 / June 21, 1981