Acta Informatica 19, 115–150 (1983)

Acta Informatica

Interpretation and Reduction of Attribute Grammars

Gilberto Filè

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

Summary. An attribute grammar (AG) is in reduced form if in all its derivation trees every attribute contributes to the translation. We prove that, eventhough AG are generally not in reduced form, they can be reduced, i.e., put into reduced form, without modifying their translations. This is shown first for noncircular AG and then for arbitrary AG. In both cases the reduction consists of easy (almost syntactic) transformations which do not change the semantic domain of the AG. These easy transformations are formalized by introducing the notion of AG interpretation as an extension to AG of the concept of context-free grammar form. Finally we prove that any general algorithm for reducing even the simple class of L-AG needs exponential time (in the size of the input AG) infinitely often.

Introduction

Just as nonterminals may be useless in context-free grammars, attributes may be useless in attribute grammars (AG). How to eliminate these attributes and what theoretical importance they have, is the topic of the present paper.

Let us first explain, by means of an example, what useless attributes are. The example we use is a simplified version of the "classical" AG of [17] which defines the semantics of binary numbers.

The AG G binary has three nonterminals N, L, and B with the following attributes:

N has the synthesized attribute v (value),

L and B have an inherited attribute s (scale) and also the synthesized attribute v.

The productions of G binary are $N \to L$, $L \to LB$, $L \to B$, $B \to 1$, $B \to 0$. The attribute dependencies and the corresponding semantic rules for the productions are given in Fig. 1.

Figure 2 contains a derivation tree graph of G binary, that is, a derivation tree t of G binary (producing the binary number 0011) in which the de-

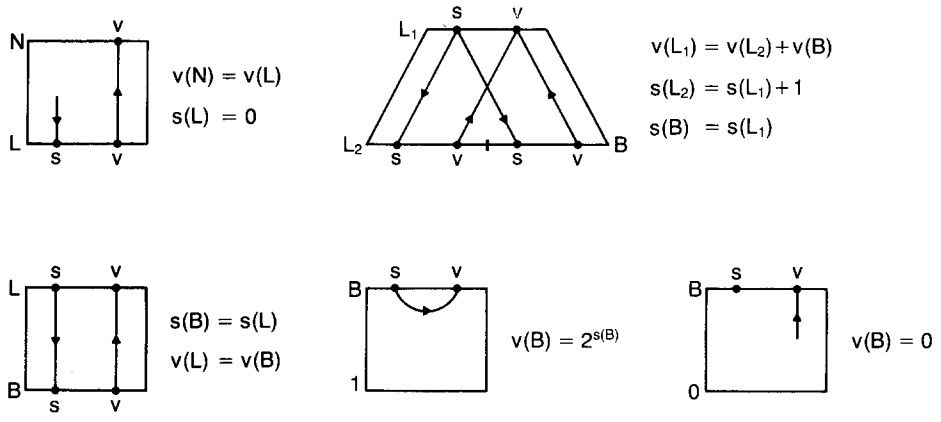

Fig. 1. The AG G binary (subscripts are added to distinguish different occurrences of the same nonterminal in one production)

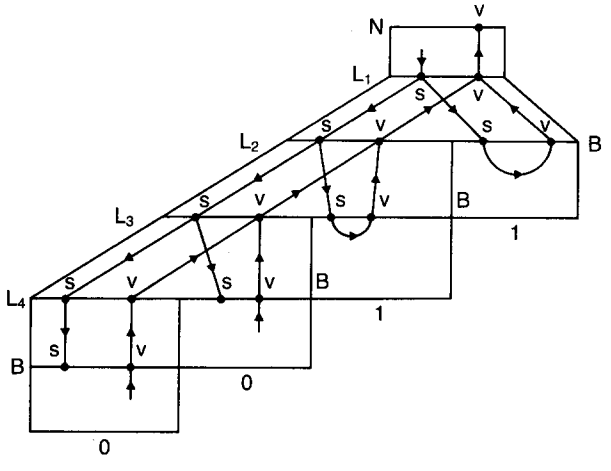

Fig. 2. Derivation tree graph of G binary

pendencies among the attributes in t are also represented. Clearly the attribute v of the root N of t has a special role: we say that its value is the *meaning* of t and of its yield 0011 (for $t, v(N)=3$). In general, we will require that in any AG G the start symbol has a designated synthesized attribute whose value is the meaning of each derivation tree of G (for G binary $v(N)$ is the designated attribute). The translation realized by an AG G is the mapping from each string w in the context-free language of G to the meaning of a derivation tree of w. From Fig. 2 it is easy to see that the attributes $s(L_3)$, $s(L_4)$ and s of the two nonterminals B deriving 0 are not used to compute the meaning of t: they are not connected by any directed (dependency) path to $v(N)$. These attributes are therefore said to be *useless* in t. Note however that the attributes s of the nonterminals B producing 1 and $s(L_1)$ and $s(L_2)$ are *useful* in t. This implies that the attribute s of L and B can in certain instances be useful and in other

useless. Precisely, $s(B)$ is useful if and only if B produces 1 and $s(L)$ is useful if and only if L produces a terminal string containing at least one 1. This ends our example.

Useless attributes have no influence on the translation defined by AG, thus, we can think about eliminating them from AG or simply ignoring them during attribute evaluation. We call these two ways of dealing with useless attributes Line 1 and 2, respectively. In this paper we pursue both lines. The ideas and the results originating from each of them are discussed in what follows.

Line 1. Because useless attributes play no role in the translation of AG, it is natural to consider AG whose derivation trees contain only useful attributes (these AG we call in *reduced form*) and to study how any given AG can be reduced, i.e., put into reduced form, without changing its translation. The main achievement of the paper is, in fact, the construction of an algorithm, called reduction-algorithm, which transforms any AG G into a reduced AG G' with the same translation as G. Such an algorithm is important as a theoretical tool; in fact, in the proofs of Theorems about AG, it is often very convenient to be able to assume that all attributes are actually useful. For example this assumption is useful when considering the complexity of the membership problem for output languages of AG.

The reduction-algorithm, which we describe in this paper, is particularly useful as a theoretical tool because it has the feature of preserving many AG-properties which characterize classes of AG, as, for instance, noncircular AG [17], one pass left-to-right evaluatable AG (L-AG) [2], simple and pure multi-pass AG [2, 1] and one-visit AG [7]. This particular feature of the reduction-algorithm follows from the fact that the algorithm consists of a sequence of "easy" transformations each of which preserves the above mentioned AG-properties. We say "easy" transformations because they are essentially syntactical and, in fact, they fit into a general framework which is an extension to AG of the classical technique for transforming context-free grammars (CFG) which consists roughly in "adding information to the nonterminals of the CFG". Well known examples of the use of this technique are the proofs that CF-languages are closed under intersection with regular languages [13] (pairs of states of the finite automaton are added to the nonterminals) and that for every $LL(k)$ CFG there is an equivalent strong $LL(k)$ CFG [20] (right-contexts are added to the nonterminals).

Because the notion of interpretation in the theory of context-free grammar forms [4, 21, 23] can be viewed as a general description of this technique for transforming CFG, we have extended, in an obvious way, this notion to AG introducing the concept of *AG interpretation.* A similar concept is considered also in [3], where covering AG are introduced.

By using the notion of AG interpretation we will prove general results which apply to any AG-transformation which, from each input AG G, constructs an interpretation of G. It is important to notice that the class of such AG-transformations is very broad: it contains most of the transformations described in earlier works on AG, see [10, 18, 14, 5].

In order to give an idea of what sort of transformations are modelled by interpretations, and how they are used for reducing AG, we transform the AG

G binary of the above example into an AG G' binary which is an interpretation of G binary, is in reduced form, and defines the same translation as G binary. The transformation is mainly syntactical in the sense that the nonterminals of G' binary are obtained from those of G binary by adding extra information to them, viz., information concerning which attributes are useful. Each new nonterminal, then, has the attributes indicated in the information and the productions of G' binary and their semantic rules are obtained from those of G binary by checking that the information in the nonterminals is not contradicted.

At the end of the previous example we have seen that the presence of useless attributes in G binary is due to the fact that the nonterminals L and B have always both attributes s and v eventhough, in case L produces a string of only 0's or B produces 0, the attribute s is not needed to compute v. In order to avoid this, the reduced AG G' binary is constructed as follows. Corresponding to nonterminal L of G binary, G' binary has two nonterminals, L^1, having both attributes s and v of L, and L^0, having only attribute v. Similarly, corresponding to B, G' binary has B^1, having both s and v, and B^0 having only v. Clearly, the idea is that L^1 derives only terminal strings containing at least one 1, whereas L^0 derives only 0's, and that B^1 derives 1, whereas B^0 derives 0. A set of productions for G' binary meeting this condition is constructed from the productions of G binary as follows.

Productions of G binary	*Corresponding productions of G′ binary*
$N \to L$	$N \to L^0$; $N \to L^1$
$L \to LB$	$L^1 \to L^1 B^0$; $L^1 \to L^0 B^1$; $L^1 \to L^1 B^1$
	$L^0 \to L^0 B^0$
$L \to B$	$L^1 \to B^1$; $L^0 \to B^0$
$B \to 1$	$B^1 \to 1$
$B \to 0$	$B^0 \to 0$

In Fig. 3 we give the semantic rules and the attribute dependencies of some of the productions of G' binary in order to show that the semantic rules of a production of G' binary are (apart from the names of the nonterminals) a subset of those of the corresponding production of G binary.

We stated earlier that the importance of the reduction-algorithm as a theoretical tool relies also on the fact that it preserves many AG-properties since it consists of interpretations. In what follows we will see, in fact, that this particular feature of the algorithm is useful for giving answers to questions arising from the second way of dealing with useless attributes which we have called Line 2.

Line 2. Useless attributes do not have influence on the meaning of a derivation tree, therefore, they do not need to be evaluated. We call *translational approach* to attribute evaluation the (theoretically more straightforward) point of view under which "evaluating a derivation t" means to compute the meaning of t (the value of its designated attribute) and, thus, does not require the evaluation of the useless attributes in t. The translational approach differs from the

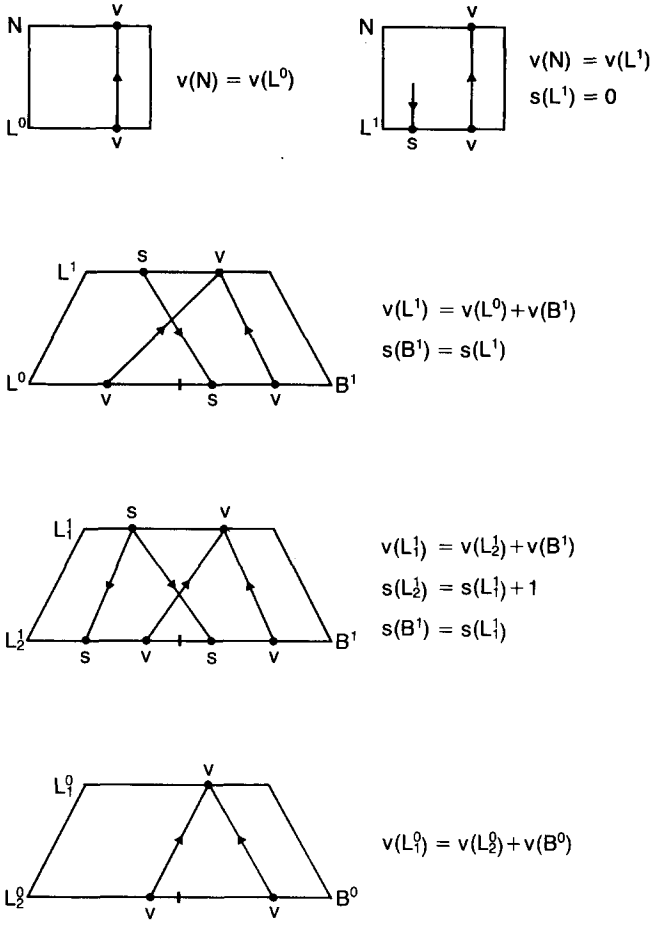

Fig. 3. Some productions of G' binary with the corresponding semantic rules

approach taken in most of the past work on AG [17, 2, 1, 16], which we call *conventional*, and in which "evaluating a tree" means to compute *all* its attributes. In these earlier works, several classes of AG were introduced (adopting the conventional approach); among them we mention the classes (which we have already considered in Line 1) of noncircular AG [17], L-AG [2], and simple [2] and pure [1] multi-pass AG.

Adopting the translational approach, all definitions of these classes can be easily changed into new definitions which characterize different classes of AG which we call translational. For instance, the conventional definition of L-AG [2] is as follows, "an AG G is L iff all attributes of each derivation tree t of G can be evaluated in one left-to-right pass over t", and the corresponding translational definition is, "an AG, is *translationally* L iff all useful attributes (i.e. the meaning) of each derivation tree t of G can be evaluated in one left-to-right pass over t".

It is easy to see that a translational class, of AG is always larger than the corresponding conventional one: for example, (conventionally) noncircular AG $\subsetneq$ translationally noncircular AG (in which useless cycles are allowed). Note, however, that while for the (conventional) classes of L-AG and simple and pure multi-pass AG there are particularly straightforward attribute evaluation techniques, the same does not hold for the corresponding translational classes.

The existence of these translational classes of AG gives rise to several theoretical questions. First, we may wonder if it is decidable whether any given AG belongs to a particular translational class. Secondly, it is interesting to compare the classes of translations defined by a translational and the corresponding conventional class of AG.

The existence of the reduction-algorithm, we mentioned before, answers both these questions. In fact, we will prove that since it consists of interpretations, it constructs from any AG G an equivalent AG G' in reduced form such that the following is true: G belongs to some translational class iff G' does too, but, because G' is in reduced form, it also belongs to the corresponding conventional class. Thus, because the membership problem for conventional classes of AG is decidable, the reduction-algorithm gives an effective way of testing membership in translational classes. From these observations it also follows that the reduction-algorithm transforms each AG belonging to some translational class into an AG of the corresponding conventional class defining the same translation. Hence, eventhough a translational class is strictly larger than the corresponding conventional one, their classes of translations are the same.

The content of the paper is divided as follows. Sect. 1 consists of preliminary (standard) definitions about AG. Sect. 2 contains the important definitions of reduced AG and of translation and equivalence of AG. In Sect. 3 we introduce the concept of interpretation of an AG which is the abovementioned extension of CFG-forms to AG. General results, of which we will make use throughout the paper, will be proved on what is the relationship between two AG G and G', where G' is an interpretation of G.

In Sect. 4 we consider noncircular AG. We show that it is easy to test whether they are reduced and that there is an easily understandable algorithm which transforms any noncircular AG into a reduced AG defining the same translation.

In Sect. 5 we show that arbitrary (even circular) AG can be reduced without modifying their translation. The same result has been proved independently in [18]. With the purpose of reducing arbitrary AG we introduce an AG-transformation which, from any AG G, constructs an interpretation G' of G, which is equivalent to G and such that the attributes of its nonterminals depend on each other in a fixed way (see also [18]). Apart from its importance in reducing AG, this transformation also proves that for every noncircular AG there is effectively an equivalent AG which is absolutely noncircular [16].

Finally, in Sect. 6 we consider the time-complexity of the task of reducing AG. The reduction-algorithms of Sect. 4 and 5 take exponential time and here we show that there cannot be a polynomial algorithm performing the same task. In fact, we show that any algorithm for reducing AG takes exponential

time (in the size of the given AG) infinitely many times. This result holds even if we allow the algorithm to produce from any AG G, a reduced AG G', no longer with the same translation as G, but just with the same output set (the range of the translation). Moreover the result holds even if G is an L-AG.

1. Preliminaries

This section consists of some terminology and definitions concerning attribute grammars (AG).

With the purpose of studying the formal power of AG we explicitly give, for each AG G, together with the usual specifications, its *semantic domain* D, which consists of an ordered 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: $V_1 \times V_2 \times \ldots \times V_m \to V_0$ with $m \geqq 0$ and $V_i \in \Omega$ for $i \in [0, m]$. The mappings in Φ are called *semantic functions*. We give now the definition of AG [17] and of other useful concepts related to them.

Definition 1.1. An *attribute grammar* G is described in the following five points.

(1) G has a semantic domain $D=(\Omega_D, \Phi_D)$.

(2) G has a context-free grammar (CFG), $G_0=(V_t, V_n, P, Z)$, called the underlying CFG of G. We assume G_0 to be always a reduced CFG in the usual sense. Throughout the paper a production $p \in P$ of G_0 (and of G) is indicated as p: $F_0 \to w_0 F_1 w_1 \ldots w_{n-1} F_n w_n$, where $n \geqq 0$, $F_i \in V_n$ and $w_i \in V_t^*$ for all $i \in [0, n]$. When considering a derivation tree t of G, we assume its leaves to be labeled by terminals (or the empty string); the string resulting by concatenating these terminals from left-to-right is denoted by yield(t). A complete derivation tree is a derivation tree whose root is the start symbol Z.

(3) Each nonterminal F of V_n has two associated disjoint sets, denoted $S(F)$ and $I(F)$, of synthesized and inherited attributes, respectively (shortly s-and i-attributes). The initial nonterminal Z has only s-attributes and one of them, denoted d, is designated to hold the meaning of any complete derivation tree of G. The set of all attributes of F is $A(F)=I(F) \cup S(F)$ and with $b(F)$ we denote an attribute $b \in A(F)$.

(4) For each attribute b of G, Ω_D contains a set $V(b)$ of the possible values of b.

(5) With each production $p \in P$ is associated a finite set r_p of semantic rules which have the following form: $a_0(F_{i_0})=f(a_1(F_{i_1}), \ldots, a_m(F_{i_m}))$, where for $j \in [0, m]$, $i_j \in [0, n]$ and f is a mapping in Φ_D from $V_1 \times \ldots \times V_m$ to V_0, such that $V_j = V(a_j(F_{i_j}))$ for all $j \in [0, m]$. When the identity of the nonterminals is not important, we indicate such a semantic rule simply by $a_0=f(a_1, \ldots, a_m)$. We say that this semantic rule *defines* a_0 *using* $a_1, \ldots, a_m$ *as arguments* or that a_0 *depends on* $a_1, \ldots, a_m$ *in* p. Let $A(p)$ be the set of all attributes of all nonterminals of p and Def(p) the subset of $A(p)$ containing the attributes in $S(F_0)$ and $I(F_j)$ for all $j \in [1, n]$. Throughout the paper we assume every AG to satisfy the following condition, which we call the *consistency-requirement*: for each

production p, the semantic rules in r_p *define all and only* the attributes in Def(p) using as arguments *only* attributes in $A(p) - \text{Def}(p)$ (see also [2]). □

Working with AG, it is useful to represent the dependencies among the attributes of the nonterminals in a production and in a derivation tree by means of graphs as follows.

Given an AG G, for a production p of G of the form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, *the production-graph* of p ($pg(p)$) is the directed graph having as nodes the attributes of all nonterminals F_i of p, $i \in [0, n]$, and in which there is an edge running from attribute a_1 to attribute a_2 iff a_2 depends on a_1 in p (see also [16, 17]). Because derivation trees consist of productions, for each derivation tree t of G we obtain a graph, called *derivation tree graph* of t (dtg(t)), by pasting together appropriately the production-graphs of all the productions used in t (see also [16, 17]).

AG which satisfy the condition that for no complete derivation tree t the graph dtg(t) contains an oriented cycle are called *noncircular AG* [17]. Given an AG G, consider a nonterminal F of G. An *is-graph* of F is a graph which has as nodes the attributes of F and edges running only from i-attributes to s-attributes [16, 17]. An is-graph g of F is *valid* if there exists a derivation tree t of G with root F, such that there is an edge from i-attribute a to s-attribute b in g iff in dtg(t) there is a corresponding oriented path from a to b of the root F. When this condition holds we say also that g is the is-graph of t (denoted $g = is(t)$). In general a nonterminal F of G has more than one valid is-graph. When an AG is such that every nonterminal has exactly one valid is-graph we say that it is a *one-behaviour* AG.

Consider a derivation tree t of an AG G. Let the top-production of t be p, of the usual form, and let $t_1, \dots, t_n$ be the subtrees of t rooted in the nonterminals $F_1, \dots, F_n$ of p. In order to compute the is-graph g_0 of F_0 such that $g_0 = is(t)$ we do not need to consider the whole graph dtg(t). It is sufficient, in fact, to know $pg(p)$ and $g_j = is(t_j)$ for all $j \in [1, n]$. Let us transform $pg(p)$ by adding to it, for each $j \in [1, n]$, the edges which are in g_j; it is easy to see that the obtained graph contains a path running from an i- to an s-attribute of F_0 iff dtg(t) does too, and, therefore, g_0 can be easily computed from it. A graph obtained in this way, that is, by "plugging into" $pg(p)$ one is-graph g_j for every right-hand side nonterminal F_j of p, is called an *augmented production-graph* of p and is indicated with $pg(p)[g_1, \dots, g_n]$. An augmented production-graph of p is *valid* if the is-graphs of the nonterminals F_j are valid for all $j \in [1, n]$.

2. Translation and Equivalence of Attribute Grammars

In this section several important concepts, which we have intuitively described in the Introduction, are explained more precisely. We start from the concept of useful and useless attributes in a derivation tree, to arrive at the definition of meaning of a tree and, finally, at that of translation and equivalence of AG.

Consider a complete derivation tree t of an AG G. We call *useful subgraph of* dtg(t) the subgraph of dtg(t) which contains the designated attribute d of t, all those nodes of dtg(t) connected to d by some directed path in dtg(t), and all

the edges of dtg(t) connecting these nodes. Clearly, the useful subgraph of dtg(t) contains all those attributes which are needed to evaluate the designated attribute d. Thus, we say that an attribute of t is *useful in t* if it belongs to the useful subgraph of dtg(t), otherwise it is *useless in t*. A complete derivation tree whose attributes are all useful is called *reduced* and an AG such that all its complete derivation trees are reduced is said to be in *reduced form* (or, simply, reduced). An important concept in nonreduced trees is that of loose cycle: a *loose cycle* in a complete derivation tree t is a directed cycle in dtg(t) which is not connected to d, i.e., no attribute in the cycle is useful in t.

Let t be a complete derivation tree of an AG G. The *meaning* of t, denoted by $m_G(t)$, is the value of the designated attribute d. Clearly $m_G(t)$ is defined only if the useful subgraph of dtg(t) does not contain cycles: in this case the value of d can be obtained by evaluating in some order the semantic rules defining the useful attributes of t.

We define now the concepts of translation and equivalence of AG.

Definition 2.1. (1) For an arbitrary (i.e., not necessarily noncircular) AG G, the *translation* defined by G is the relation $T_G = \{(\text{yield}(t), m_G(t)) |$ where t is a complete derivation tree of $G\}$.

(2) Two AG G_1 and G_2 over the same semantic domain are *equivalent* if $T_{G_1} = T_{G_2}$. □

For an AG G the range of T_G, i.e., $\{m_G(t) | t$ is a complete derivation tree of $G\}$, is called the *output set* of G.

We point out that the above definition of meaning of a tree t and thus, Definition 2.1(1) and (2) are based on the fact that we only require the evaluation of the designated attribute of t. In other words our definitions are based on the translational approach to attribute evaluation which we have described in the Introduction. Also in the Introduction we stated that it is easy to transform any AG-property based on the conventional approach into one based on the translational approach. In what follows we actually transform in this way several well known AG-properties. In the next section we will use these properties (both, conventional and translational) as examples of AG-properties which are preserved by the particular type of AG-transformation which we will define there.

In Sect. 1 we have given the conventional definition of noncircular AG [17]. The corresponding translational property is easy to find: an AG G is *translationally noncircular* if the meaning of each complete derivation tree t of G is defined, i.e., the useful subgraph of dtg(t) is acyclic. Clearly the translational definition allows cycles in dtg(t) as long as they are loose cycles. From this it is immediate that the class of translationally noncircular AG properly includes that of (conventionally) noncircular AG.

Let us also consider the following conventional definitions.

(1) *Left-to-right one-pass evaluable AG* (*L*-AG) [2]. An AG G is L iff *all attributes* of every complete derivation tree t of G can be evaluated by walking through t once in a depth-first fashion from left-to-right (such a sweep through t is called a *pass*).

(2) *Pure multi-pass AG* [1]. An AG G is pure multi-pass iff for some $k>0$, *all attributes* of every complete derivation tree t of G can be evaluated by performing a sequence of k passes through t.

(3) *Simple multi-pass AG* [2, 1]. An AG G is simple multi-pass iff for some $k>0$, there is for each attribute a of G a pass number $p(a)\leqq k$, such that, for every complete derivation tree t of G, *all attributes* of t can be evaluated by a sequence of k passes through t, where in pass i *all attributes* a such that $p(a)=i$ are evaluated, $i\in[1,k]$.

The corresponding translational definitions are obtained by substituting in the three definitions above the "*all attributes*" with "all useful attributes". Thus, e.g., for translationally L-AG we do not care whether the useless attributes can be computed in one pass. For one-visit and translationally one-visit AG see [7].

As in the case of noncircularity, it is clear that, if X is any of the three properties above, then, the class of translationally X-AG properly contains that of (conventionally) X-AG. However in Sect. 5 it will be proven that the classes of translations defined by translationally X-AG and X-AG over any semantic domain D are equal. In the coming section we introduce the main tool we use in order to obtain this result, the concept of AG-interpretation.

3. Attribute Grammar Interpretation

In this section we extend in a simple way the concept of context-free grammar form [4, 21, 23] to AG (in [3] the similar concept of covering for CFG is extended to AG). In this way we describe formally a class of AG transformations and we prove precise results which hold for every transformation of the class. Clearly, this extension is very natural and has many applications. In fact, in the first place, all the transformations we describe in this paper for reducing AG, are of this class and, in the second place, most of the AG-transformations, deviced in the past for changing AG of some class into equivalent AG of another class, fit into this framework. As examples of such transformations, we mention that in [10] a transformation of this type was used to prove that for every noncircular AG there exists (effectively) an equivalent simple multi-visit AG. In [18, 5, 22], several transformations of this kind are also used in order to prove, among other things, that for every noncircular AG there exists an equivalent absolutely noncircular AG [16]. This last transformation is also contained in Sect. 5 of this paper.

In order to introduce the definition of AG interpretation we need the following concept [21]. A finite substitution μ, defined on alphabet V, is said to be a disjoint finite letter substitution (shortly: *dfl*-substitution) if, for every $A\in V$, $\mu(A)$ is a finite set of letters, and if A and B belong to V and $A\neq B$, then $\mu(A)\cap\mu(B)=\phi$.

Definition 3.1. Given an AG G, with underlying CFG $G_0=(V_t, V_n, P, Z)$ and a *dfl*-substitution μ on $V_n\cup V_t$, we say that an AG G', with underlying CFG $G'_0=(V_t, V'_n, P', Z')$, is an *interpretation* of G *modulo* μ, denoted by $G'\underset{\mu}{\rhd}G$, if the following five conditions hold.

(1) The semantic domains of G and G' are equal.

(2) $\mu(F) \subseteq V_n'$ for all $F \in V_n$ and if nonterminal F' of G' belongs to $\mu(F)$ then $A(F') \subseteq A(F)$.

(3) $\mu(a) = \{a\}$ for all $a \in V_t$.

(4) For a production p of G of the usual form $F_0 \to w_0 F_1 w_1 \ldots w_{n-1} F_n w_n$, with $\mu(p)$ we indicate the set of productions p' of the form, $F_0' \to w_0 F_1' w_1 \ldots w_{n-1} F_n' w_n$, where $F_i' \in \mu(F_i)$ for all $i \in [0, n]$. Then, $P' \subseteq \mu(P)$, where $\mu(P) = \bigcup \{\mu(p) | p \in G\}$.

Consider two productions $p' \in P'$ and $p \in P$ such that $p' \in \mu(p)$. The set $r_{p'}$ of semantic rules of p' is a subset of the set obtained by substituting in every semantic rule of r_p each nonterminal F_i of p with the corresponding F_i' of p', for $i \in [0, n]$.

(5) $Z' \in \mu(Z)$ and $A(Z')$ in G' contains the designated attribute d of Z which is also the designated attribute of Z'. □

If in point (2) of the above definition, for all $F' \in \mu(F)$ of G', $A(F') = A(F)$, G' is called a *simple interpretation* of G. In this case, corresponding productions p and p', as in point (4), will have the same set of semantic rules, apart from the "priming" of the nonterminals.

Observation 3.1. Consider point (4) of Definition 3.1. It requires that, if p and p' are productions of G and G', respectively, where $p' \in \mu(p)$, then $r_{p'} \subseteq r_p$ (apart from the "priming" of the nonterminals). This requirement has implications which need some more explanations. From point (2) of Definition 3.1, it follows that $A(p) \supseteq A(p')$ and, in particular, $\mathrm{Def}(p) \supseteq \mathrm{Def}(p')$. The consistency-requirement we assumed on all AG (cf. Definition 1.1), implies that the semantic rules in r_p and $r_{p'}$ must define all and only the attributes in $\mathrm{Def}(p)$ and $\mathrm{Def}(p')$, respectively. Hence, $r_{p'}$ consists of those semantic rules of r_p which define the attributes in $\mathrm{Def}(p')$. From this it follows that all the attributes which are used as arguments in these semantic rules must also be present in $A(p')$. In other words, because of the consistency-requirement and point (4) of Definition 3.1, the attributes in $\mathrm{Def}(p')$ force the presence in $A(p')$ of the attributes which are needed as arguments in the semantic rules of $r_{p'}$.

Example 3.1. Consider an AG G which has a production p: $A \to B$, where A and B are nonterminals of G, both with attribute set $\{i_1, i_2, s_1, s_2, s_3\}$, where i_1 and i_2 are i-attributes and the others are s-attributes. Figure 4(1) shows the

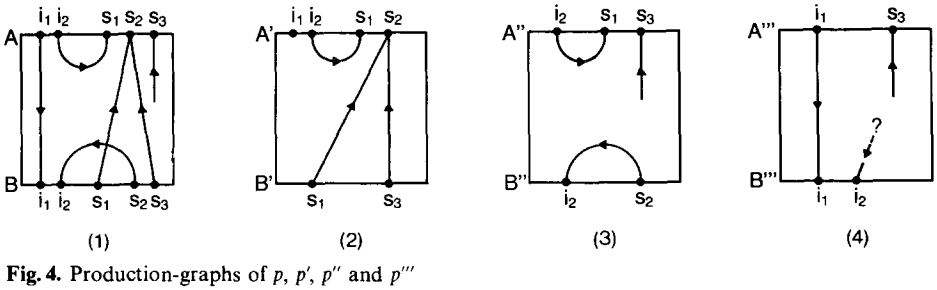

Fig. 4. Production-graphs of p, p', p'' and p'''

production-graph of production p. Clearly the two edges $(s_1(B), s_2(A))$ and $(s_3(B), s_2(A))$, in Fig. 4(1), imply that the semantic rule of p defining $s_2(A)$ has the form, $s_2(A)=f(s_1(B), s_3(B))$. The arrow entering $s_3(A)$ is used to represent the fact that this attribute is defined in p by a constant-assignment-semantic rule i.e. a rule of the form, $s_3(A)=$constant.

In the other parts of Fig. 4 we give examples of which productions may or may not correspond to p in an interpretation G' of G.

Assume that $G' \underset{\mu}{\rhd} G$, where $\mu(A)=\{A', A'', A'''\}$ and $\mu(B)=\{B', B'', B'''\}$. As indicated in Fig. 4, each of these nonterminals of G' has only a subset of the attributes of the corresponding nonterminal of G (point(2) of Definition 3.1 is satisfied).

In Definition 3.1(4) we required that the semantic rules of each production of G' are a subset of those of the corresponding production of G (apart form the priming of the nonterminals). Assume, then, that this is true for the three productions p', p'' and p''' of Fig. 4, that is, assume that the semantic rule which, for example, in p' defines $s_2(A')$, using $s_1(B')$ and $s_3(B')$ as arguments, is "the same" as the semantic rule which defines $s_2(A)$ in p, i.e. it is $s_2(A')=f(s_1(B'), s_3(B'))$. At this point, it is easy to see from Fig. 4 that p' and p'' may be productions of G', whereas p''' cannot. This is because, while the semantic rules of p which are passed to p' and p'' satisfy the consistency-requirement of AG (see Definition 1.1), those of p''' do not: $i_2(B''')$ which belongs to $\mathrm{Def}(p''')$ is not defined by any semantic rule of p'''. Note also that it would not be possible to add to p''' the semantic rule of p defining $i_2(B)$ because B''' does not have the attribute s_2 which this rule needs as argument (cf. Observation 3.1). □

Intuitively, if $G' \underset{\mu}{\rhd} G$, then G' is very similar to G; let us see better how. From Definition 3.1 we know that for every F' of V_n' there is a unique $F \in V_n$ such that $F' \in \mu(F)$; we denote this F by $\mu^{-1}(F')$. Thus, μ^{-1} is a mapping from V_n' to V_n. Similarly, for each production p' of G', there is a unique production p of G such that $p' \in \mu(p)$ and again we can write $p=\mu^{-1}(p')$. Extending in an obvious way μ and μ^{-1} to derivation trees of G and G', it is clear that, for each derivation tree t' of G', there is a unique derivation tree t of G such that $t' \in \mu(t)$ or $t=\mu^{-1}(t')$. Note that obviously t and t' differ simply for the labels of the internal nodes whereas $\mathrm{yield}(t)=\mathrm{yield}(t')$. In case the mapping μ^{-1} on complete derivation trees is onto, i.e., for every complete derivation tree t of G there is a complete derivation tree t' of G' such that $t=\mu^{-1}(t')$, we say that G' is a *full interpretation* of G.

These observations have clarified the syntactic similarity between G and G'. In what follows we will consider the semantic similarity between G and G' in terms of attribute dependencies and translation.

Let t and t' be derivation trees of G and G', respectively, such that $t=\mu^{-1}(t')$. As noted before, t and t' have the same structure and differ only in the labeling of internal nodes, i.e., if F and F' are (labels of) corresponding nodes of t and t', then $\mu^{-1}(F')=F$. Consider now $\mathrm{dtg}(t)$ and $\mathrm{dtg}(t')$. Since for corresponding nodes F and F', $A(F) \supseteq A(F')$, (cf. Definition 3.1(2)), each node of $\mathrm{dtg}(t')$ has a corresponding node in $\mathrm{dtg}(t)$.

Moreover, since the semantic rules in G' are a subset of those of G (cf. Definition 3.1(4)), each edge in dtg(t') has a corresponding edge in dtg(t). In this sense, we can identify corresponding nodes and edges of dtg(t) and dtg(t'). Thus, in general, we can view dtg(t') as a subgraph of dtg(t), which we denote by dtg(t)$\supseteq$dtg(t'); in particular cases dtg(t) and dtg(t') may even be equal.

With these remarks in mind, let us compare the semantics of two AG G and G' such that $G' \underset{\mu}{\rhd} G$.

Lemma 3.1. *For any two AG G and G', such that $G' \underset{\mu}{\rhd} G$, the following statements hold.*

(1) *If t and t' are complete derivation trees of G and G', respectively, such that $t = \mu^{-1}(t')$, then* dtg(t)$\supseteq$dtg(t')$\supseteq$*useful subgraph of* dtg(t)$=$*useful subgraph of* dtg(t'). *If, moreover, G' is a simple interpretation of G, then* dtg(t)$=$dtg(t').

(2) $T_G \supseteq T_{G'}$

(3) *If G' is a full interpretation of G, then $T_G = T_{G'}$ (i.e., G and G' are equivalent).*

Proof. The fact that dtg(t)$\supseteq$dtg(t'), (dtg(t)$=$dtg(t') in case G' is a simple interpretation of G) is immediate from Definition 3.1: each node and edge of dtg(t') has a corresponding node and edge in dtg(t) (cf. remarks before the Lemma). From this and the fact that t and t' have the same designated attribute d (Definition 3.1(5)), it follows that useful subgraph of dtg(t)$\supseteq$useful subgraph of dtg(t'). By induction on the length of paths in dtg(t), which run from useful attributes to d, it is possible to show that useful subgraph of dtg(t') $\supseteq$useful subgraph of dtg(t), using the fact that any attribute present in both trees is defined by the "same" semantic rule (see Observation 3.1).

Point (2) of the Lemma is shown as follows. Since dtg(t) and dtg(t') have the same useful subgraph and since the semantic rules defining attributes in both trees are "equal", $m_G(t) = m_{G'}(t')$. Thus, from the fact that yield (t)$=$yield (t'), we have that $T_G \supseteq T_{G'}$. Point (3) is immediate from the preceding ones. □

We will now look at some interesting properties of AG interpretations. First of all, it is easy to see that, as in the CF-case, AG-interpretations are transitive, i.e. if $G_2 \underset{\mu_2}{\rhd} G_1$ and $G_1 \underset{\mu_1}{\rhd} G$, then $G_2 \underset{\mu}{\rhd} G$, where μ is the *dfl*-substitution obtained by composing, in the obvious way, μ_1 and μ_2.

A second interesting aspect of AG interpretations, which we already mentioned in the Introduction, is that they preserve many AG-properties, i.e., if $G' \underset{\mu}{\rhd} G$ and G has some particular property, then also G' does. First, it is obvious that if G has, for instance, only s-attributes, or only i-attributes, so does G'. Secondly, we will show that interpretations preserve the more interesting AG-properties which we have described in Sect. 2 under both, the conventional and the translational approach.

The fact that noncircularity and the L-property (under both approaches) are preserved under interpretations is immediate from Lemma 3.1. In what follows we give an informal argument showing that also the pure and the simple multi-pass properties (conventional and translational) are preserved. The same

result can be proved in a similar way also for other AG properties as, for instance, the one-visit, the pure and the simple multi-visit [19, 10] and multi-sweep [11] properties.

Let $G' \underset{\mu}{\rhd} G$ and t and t' be complete derivation trees of G and G' such that $\mu^{-1}(t')=t$. If there is a way of evaluating all (or all useful) attributes in t, by performing k left-to-right passes ($k>0$) over t, then, since, by Lemma 3.1, $\mathrm{dtg}(t) \supseteq \mathrm{dtg}(t') \supseteq$ useful subgraph of $\mathrm{dtg}(t)=$ useful subgraph of $\mathrm{dtg}(t')$, in the same way all (or all useful) attributes of t' can also be computed. Therefore, if G is pure multi-pass so is G'. Let us now consider the simple multi-pass property. Assume that G is simple multi-pass, that is, for some $k>0$ there is a pass number $p(a) \in [1, k]$ for each attribute a of G, such that every occurrence of a is evaluated during pass $p(a)$. Consider again t and t' of G and G' such that $t=\mu^{-1}(t')$. From Lemma 3.1 it is immediate to see that if all occurrences of an attribute a in t can be evaluated during pass $p(a)$, then, those occurrences of a, which are present also in t', can be evaluated during pass $p(a)$ too. Hence, the pass-assignment to attributes carries over from G to G' and, thus, G' is also simple multi-pass.

Observe that, in the case that G' is a full interpretation of G, for translational AG-properties, a stronger result holds: G has a translational property iff also G' does. This follows from Lemma 3.1(1) which states that two corresponding complete derivation trees of G and G' have the same useful subgraph. This stronger result will be useful later for showing that translational properties of AG are decidable.

4. Reducing Noncircular Attribute Grammars

In this section we consider the problem of reducing noncircular AG. We do this first of all because the class of noncircular AG is the largest class of AG of practical importance, and secondly because it is possible to characterize noncircular AG in reduced form by means of easily testable conditions. This characterization will then be the basic intuition behind the proof that for every noncircular AG there is an equivalent one in reduced form. This proof is based on an algorithm, called the NC-reduction-algorithm, which for every noncircular AG G constructs a full interpretation G' of G in reduced form. The NC-reduction-algorithm will play an important role also in the next section, where we will face the general problem of reducing arbitrary AG. We can already say that, in the general case, the process of reducing an AG will be more complicated and that the NC-reduction-algorithm will perform the first of a chain of three transformations needed for the reduction.

Let us see how to test whether a noncircular AG is in reduced form.

Lemma 4.1. *A noncircular AG G is in reduced form iff the following two conditions hold,*

(1) *the start symbol Z of G has only the designated attribute d,*

(2) *for each production p of G of the usual form $F_0 \to w_0 F_1 w_1 \ldots w_{n-1} F_n w_n$, all attributes in $A(p)-\mathrm{Def}(p)$, i.e., in $I(F_0)$ and $S(F_j)$, $j \in [1, n]$, are used as arguments in the semantic rules of p.*

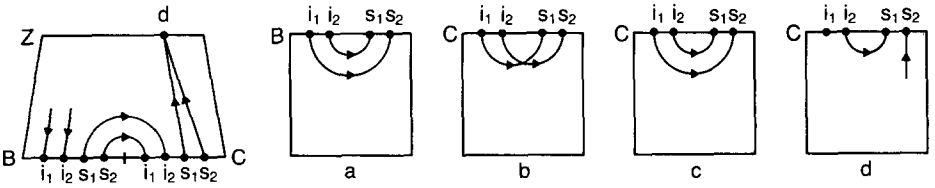

Fig. 5. The production-graphs of G

Proof. $\Rightarrow$ Condition (2) requires that in every production p of G all the attributes which are usable as arguments in the semantic rules of p, (recall the consistency-requirement of Definition 1.1), are actually used.

The obvious observation that, if an attribute of a derivation tree is useful, then it must be used to define another useful attribute or d, is sufficient to show that if G is in reduced form, then it meets condition (2). Condition (1) is trivial because no s-attribute of the root other than d can be useful.

$\Leftarrow$ Assume that G meets conditions (1) and (2). This implies that for any complete derivation tree t of G, all attributes of t, apart from d, are used as arguments of some semantic rule in t. This means that in the graph dtg(t) there is only one node with outdegree equal to zero which is d, and all other nodes have at least one edge exiting them. From this, it is easy to understand that either all nodes of dtg(t) are connected to d or there is a loose cycle in dtg(t). But, because we assumed G to be noncircular, dtg(t) cannot contain cycles of any sort. Hence, every attribute of t is useful in t and therefore G is in reduced form. □

We illustrate the result we just stated by means of the following example.

Example 4.1. Consider the noncircular AG G with nonterminals $\{Z, B, C\}$, terminals $\{a, b, c, d\}$ and productions $Z \to BC$, $B \to a$, $C \to b$, $C \to c$, $C \to d$.

The nonterminals have the following attributes: $S(Z) = \{d\}$, $I(B) = I(C) = \{i_1, i_2\}$, $S(B) = S(C) = \{s_1, s_2\}$. The production-graphs of these productions are given in Fig. 5; the corresponding semantic rules are omitted because they have no relevance in our discussion.

Clearly G can produce only three complete derivation trees, t_1, t_2 and t_3, corresponding to the derivations $Z \Rightarrow BC \overset{*}{\Rightarrow} ab$, $Z \Rightarrow BC \overset{*}{\Rightarrow} ac$ and $Z \Rightarrow BC \overset{*}{\Rightarrow} ad$, respectively. From Fig. 6, which represents the derivation tree graphs of t_1, t_2 and t_3, it is easy to see that while t_1 and t_2 contain only useful attributes (connected with d), t_3 does not: $i_1(B)$, $s_2(B)$, $i_1(C)$ are useless in t_3.

Therefore G is *not* in reduced form. Clearly, this fact is due to production $C \to d$ which, in fact, does not satisfy the second condition of Lemma 4.1 ($i_1(C)$ is not used). Observe, however, that the AG obtained from G by eliminating production $C \to d$ satisfies both conditions of Lemma 4.1 and is, in fact, in reduced form: it produces only trees t_1 and t_2.

After the following theorem we will see how G can be transformed into an equivalent reduced AG. □

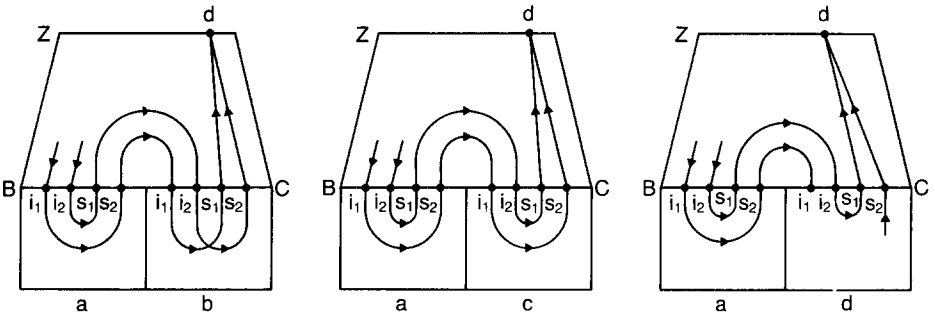

Fig. 6. The derivation tree graphs of t_1, t_2 and t_3

The idea of the preceding Lemma, that a noncircular AG is in reduced form whenever all attributes are used, is the basis of the transformation which will follow and by means of which we show how noncircular AG can be reduced. Roughly, the transformation works as follows: each production p of the given AG G, which has the usual form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, is mapped into as many productions p' as there are of the form $F_0' \to w_0 F_1' w_1 \dots w_{n-1} F_n' w_n$, where $A(F_i) \supseteq A(F_i')$ for all $i \in [0, n]$, and such that the subset of semantic rules of r_p which define the attributes in $\mathrm{Def}(p')$ satisfies condition (2) of Lemma 4.1, i.e. each attribute in $A(p') - \mathrm{Def}(p')$ is an argument of some of these semantic rules. Even from this quick idea it should be easy to see that this transformation defines an interpretation of G. The formal construction is contained in the following theorem.

Theorem 4.1. *For every noncircular* AG G, *there is, effectively, an equivalent* AG G' *such that,*

(1) G' *is a full interpretation of* G,

(2) G' *is in reduced form.*

Proof. We denote the underlying CFG's of G and G' with (V_t, V_n, P, Z) and (V_t, V_n', P', Z'), respectively.

First we give an algorithm which constructs, for every noncircular AG G, a full interpretation G' of G in reduced form; later we prove its correctness.

NC-reduction-algorithm

a) We define a *dfl*-substitution μ over $V_t \cup V_n$ as follows. For every nonterminal F of G, $\mu(F)$ contains a symbol (F, A') for each subset A' of the set $A(F)$ of attributes of F in G. The disjoint subsets of i- and s-attributes of A' are indicated with IA' and SA', respectively. As in Definition 3.1, μ leaves terminal symbols unchanged.

b) The AG G' is as follows.

(i) The set of terminal symbols of G' is V_t as in G.

(ii) The set of nonterminals is $V_n' = \bigcup \{\mu(F) | F \in V_n\}$. The start symbol is $Z' = (Z, \{d\})$, where Z is the start symbol of G and d its designated attribute.

(iii) Every nonterminal (F, A') of V_n' has attribute set A'; d is the designated attribute of Z'.

(iv) The productions of P' are constructed from those of P as follows. For every production p of G of the form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, P' contains *a production* $p' \in \mu(p)$, i.e., of the form $(F_0, A'_0) \to w_0 (F_1, A'_1) w_1 \dots w_{n-1} (F_n, A'_n) w_n$, *for each choice* of the subsets A'_i, $i \in [0, n]$, that satisfies the following condition (\$).

Condition (\$): the semantic rules $r_1, \dots, r_m$ or r_p which define the attributes in $\mathrm{Def}(p')$, i.e., those in SA'_0 and $IA'_j, j \in [1, n]$, must use as arguments *all* (and only) the attributes in $A(p') - \mathrm{Def}(p')$, i.e., in IA'_0 and $SA'_j, j \in [1, n]$.

(v) The semantic rules $r_1, \dots, r_m$ of point (iv), with the appropriate renaming of the nonterminals, are the semantic rules of $r_{p'}$ in G'.

This ends the NC-reduction-algorithm.

It is immediate from Lemma 4.1 that, since all productions of G' satisfy condition (\$), G' is in reduced form. It is also straightforward to see that $G' \underset{\mu}{\triangleright} G$. In order to show that G' is also equivalent to G, it is sufficient, by Lemma 3.1(3), to show that G' is a full interpretation of G, i.e., for every complete derivation tree t of G, there exists a complete derivation tree t' of G', such that $t = \mu^{-1}(t')$. We show that this is true with the following argument. Let t be a complete derivation tree of G and relabel each node F of t with the symbol (F, A'), where A' contains those attributes of F which are useful in t. Let t' be the resulting tree; clearly $t' \in \mu(t)$. We will show that t' is a complete derivation tree of G'. Let production p of the usual form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, be used in t and assume that each nonterminal F_i of p is relabeled in t' by (F_i, A'_i), $i \in [0, n]$. As in point(a) of the NC-reduction-algorithm we indicate the subsets of the i- and s-attributes of A'_i with IA'_i and SA'_i, respectively. Consider the semantic rules $r_1, \dots, r_m$ of r_p which define the attributes in SA'_0 and $IA'_j, j \in [1, n]$. With an argument similar to that of the first part of Lemma 4.1, it is easy to prove that the attributes in IA'_0 and SA'_j, $j \in [1, n]$, since they are useful in t, must be all used as arguments by the rules $r_1, \dots, r_m$: they must be used to define the other useful attributes of p. Moreover, no other attribute in $A(p)$ can be an argument of these semantic rules because, in this case, it would also be useful. From this, it follows that production p': $(F_0, A'_0) \to w_0 (F_1, A'_1) w_1 \dots w_{n-1} (F_n, A'_n) w_n$, where each (F_i, A'_i) has the attributes in A'_i and which has the semantic rules $r_1, \dots, r_m$ that we have specified above (with the appropriate renaming of the nonterminals), satisfies condition (\$) and is, therefore, a production of G'. This obviously holds for all productions of t, hence t' is a complete derivation tree of G' such that $t = \mu^{-1}(t')$. Thus, by Lemma 3.1(3) G and G' are equivalent ($T_G = T_{G'}$).

Observe that the mapping μ^{-1} is one-to-one. Let, in fact, t' and t'' be complete derivation trees of G' such that $\mu^{-1}(t') = \mu^{-1}(t'') = t$, where t is clearly a complete derivation tree of G. By Lemma 3.1, it follows that useful subgraph of $\mathrm{dtg}(t')$ = useful subgraph of $\mathrm{dtg}(t'')$ = useful subgraph of $\mathrm{dtg}(t)$ and, because G' is reduced, useful subgraph of $\mathrm{dtg}(t') = \mathrm{dtg}(t')$ = useful subgraph of $\mathrm{dtg}(t'') = \mathrm{dtg}(t'')$. Thus, corresponding nodes of t' and t'' have the same attributes and, therefore, the same nonterminal label (see NC-reduction-algorithm). Thus, $t' = t''$. □

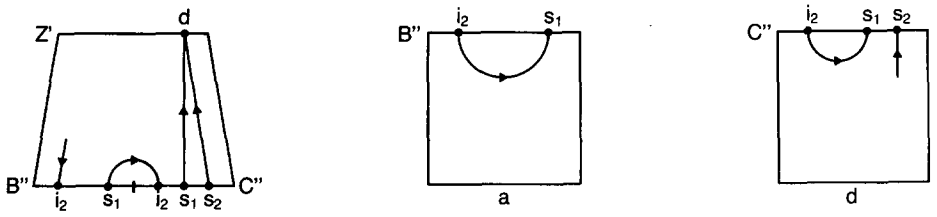

Fig. 7. The productions of G' which generate a tree corresponding to t_3 of G

In the following example we will apply the NC-reduction algorithm of Theorem 4.1 to the AG G of Example 4.1.

Example 4.2. Consider again the AG G of Example 4.1; as was shown there, G is not in reduced form. We will now use the NC-reduction-algorithm in order to obtain a reduced AG G' equivalent to G and such that $G' \underset{\mu}{\triangleright} G$. For questions of space and simplicity we will not give all the nonterminals and the productions which would be actually assigned to G' by the NC-reduction algorithm, but only those which can be used in a complete derivation tree of G' (the useful nonterminals and productions of G', in the context-free sense). Recall that G produces only the three complete derivation trees t_1, t_2 and t_3 which are shown in Fig. 6 and that in t_1 and t_2 all attributes are useful, whereas in t_3, $i_1(B)$, $s_2(B)$ and $i_1(C)$ are useless.

From the observation we made in Example 4.1 about the fact that the attribute dependencies in the four productions of G, $Z \to BC$, $B \to a$, $C \to b$ and $C \to c$, satisfy the conditions of Lemma 4.1, it is immediate that G' will have nonterminals Z', B' and C' in $\mu(Z)$, $\mu(B)$ and $\mu(C)$ with the same attribute sets as Z, B and C, respectively, and productions $Z' \to B'C'$, $B' \to a$, $C' \to b$ and $C' \to c$, with the same set of semantic rules as the corresponding productions of G. These productions of G' obviously generate two complete derivation trees t'_1 and t'_2 such that $t_1 = \mu^{-1}(t'_1)$ and $t_2 = \mu^{-1}(t'_2)$ and also $\mathrm{dtg}(t_1) = \mathrm{dtg}(t'_1)$ and $\mathrm{dtg}(t_2) = \mathrm{dtg}(t'_2)$.

For producing a tree corresponding to t_3 of G and containing only its useful attributes, G has also the productions of Fig. 7, where nonterminals B'' and C'' are in $\mu(B)$ and $\mu(C)$ and have the following attributes sets: $A(B'') = \{i_2, s_1\}$, $A(C'') = \{i_2, s_1, s_2\}$.

It is easy to see that these three productions are actually produced by the NC-reduction-algorithm (they satisfy condition (\$)).

For showing that no other production generated for G' by the NC-reduction-algorithm can be used in a complete derivation tree of G' we could use the fact, showed at the end of Theorem 4.1, that μ^{-1} over derivation trees is one-to-one: with the productions described so far G' produces three complete derivation trees which correspond to the three of G. However we prefer to give the following intuitive argument. Consider the productions of Fig. 8, where $\bar{B} \in \mu(B)$, $\bar{C} \in \mu(C)$, $A(\bar{B}) = \{i_1, s_2\}$ and $A(\bar{C}) = \{i_1, s_1, s_2\}$.

Clearly the first two productions of Fig. 8 satisfy condition (\$) of the NC-reduction-algorithm. Hence, they will be productions of G'. However they will be useless. Note, in fact, that $\bar{C}$ cannot derive a terminal symbol, because, as

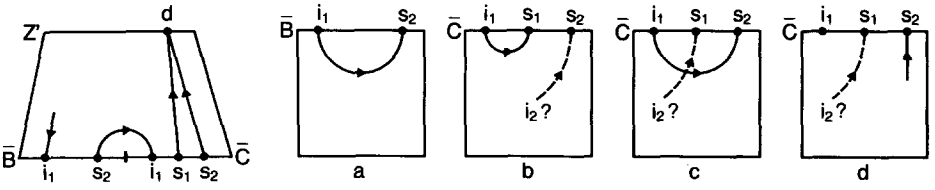

Fig. 8. Productions which are useless in G'

shown in Fig. 8, in $\bar{C} \to b$ the presence of s_2 requires that of i_2 and in $\bar{C} \to c$ and $\bar{C} \to d$ the presence of s_1 requires that of i_2 (cf. Definition 3.1 (4)). □

We observe that Theorem 4.1 is very similar and can be viewed as an extension of a result of [12] which shows that every IO macro grammar can be transformed into an equivalent IO macro grammar having only argument preserving productions; where a production is argument preserving if all the arguments of the left-hand side function symbol appear also in the right-hand side.

A precise connection between IO macro grammars and AG was shown in [6], see also [7]. Consider the class of (conventionally) noncircular AG having at most one *s*-attribute for each nonterminal (1*S*-AG) and which are defined over the semantic domain STRINGS of all strings over some finite alphabet Σ and with string concatenation as only operation. In [6, 7] it is shown that the class of output languages (range of the translation, see Sect. 2) defined by 1*S*-AG over STRINGS is equal to the class of IO macro languages over Σ.

The similarity between IO macro grammars and AG can be explained intuitively as follows: a function symbol F, of an IO macrogrammar G, can be viewed as having one *s*-attribute whose value is the terminal string w produced by F, and the list of arguments of F can be viewed as *i*-attributes which may or may not be used in the generation of w. Clearly, if all productions of G are argument preserving, then, in any derivation of a terminal string w from F all arguments of F are used in generating w. From this it should be easy to see the similarity between the result of [12] and our Theorem 4.1.

It is also interesting to note that the proof of the result of [12] uses a construction very similar to the NC-reduction-algorithm of Theorem 4.1. From an IO macro grammar G, an equivalent IO macro grammar G' with only argument preserving productions, is constructed as follows. For each function symbol F of G with m arguments, G' has one function symbol of the form F^I for each subset I of the set $\{1, \ldots, m\}$. Clearly, F^I has only the arguments of F which are indicated in I and derives in G' those terminal strings, derivable from F in G, for which only the arguments in I are needed. For each production p of G, G' has all the productions which can be obtained by substituting in p the new nonterminals of G' for the corresponding ones of G, and which are argument preserving.

From this it is clear that we can view the produced IO macro grammar G' as an interpretation of G.

The analogy between the argument preserving condition, met by the productions of G', and condition (\$) of the NC-reduction-algorithm is also straightforward.

With Theorem 4.1 we have accomplished our first main goal of showing that noncircular AG can be reduced. In the next section we will solve also the general problem of reducing arbitrary AG. Many of the proofs and observations of this section will turn out to be useful in the next one.

5. Reducing Arbitrary Attribute Grammars

In this section we extend the result of Theorem 4.1, showing that also arbitrary AG, i.e., not necessarily noncircular or even translationally noncircular AG, can be reduced without modifying their translations. The NC-reduction-algorithm of Theorem 4.1 will be useful also in this extension, but further transformations of the resulting AG are needed for obtaining a reduced AG.

In order to understand what sort of new transformations must be added to the NC-reduction-algorithm, we start by stating in the following result (whose proof will be given later) that the interpretation G_1, produced by the NC-reduction-algorithm from an arbitrary AG G, even if not reduced, still has some interesting properties.

Result 5.1. If G_1 is the interpretation produced by the NC-reduction-algorithm of Theorem 4.1 from an arbitrary AG G, then the following two points hold.

Point 1) For any complete derivation tree t of G, there is a *reduced* complete derivation tree t_1 of G_1, such that $t=\mu^{-1}(t_1)$. (Note that this implies that G_1 is a full interpretation of G and, by Lemma 3.1, that $\mathrm{dtg}(t_1)=$useful subgraph of $\mathrm{dtg}(t)$).

Point 2) Let t_1 be a complete derivation tree of G_1. If t_1 is not reduced, then $\mathrm{dtg}(t_1)$ contains a loose cycle. □

Thus, G_1 is not necessarily reduced: for each complete derivation tree t of G, G_1 has a reduced tree t_1 such that $t=\mu^{-1}(t_1)$, but it may also have other trees t_1' such that $t=\mu^{-1}(t_1')$ and which contain loose cycles and hence useless attributes. Therefore, what we will do is to filter from G_1 the trees with loose cycles and keep only the reduced ones. The main difficulty in doing this is due to the fact that the same attribute may be, in certain instances, useful and, in certain others, in a loose cycle. We overcome this difficulty by transforming G_1 into a simple full interpretation G_2 in which the dependency behaviour of the attributes is somehow fixed. The idea for doing this is the same as that of the circularity test of [17]. From this test, we know that the presence of cycles in a derivation tree t can be tested as follows: compute the is-graph of each node of t and check, for each production p of t, whether the corresponding valid augmented production-graph of p, contains a cycle (see Preliminaries). Among the same lines, we build G_2 from G_1 by adding is-graphs to the nonterminals of G_1 and by constructing the productions of G_2 from those of G_1 in such a way that the i- to s-attribute dependencies, specified in the is-graph of each

nonterminal, are respected. More precisely, the AG G_2 will be a one-behaviour simple full interpretation of G_1.

The fact that G_2 is one-behaviour implies that for each production p of G_2 there is only one valid augmented production-graph which represents the dependencies among the attributes of p in every derivation tree containing p. Therefore, if this graph contains a cycle, then, for every derivation tree t in which p occurs, dtg(t) contains a corresponding cycle and, vice versa, if dtg(t) contains a cycle then a production of t has a cyclic valid augmented production-graph. Hence, it is easy to detect the productions of G_2 giving rise to cycles and, in particular, since by Lemma 3.1, Result 5.1 holds also for G_2, those giving rise to loose cycles. Deleting these productions from G_2 is the last step needed in order to obtain from it a reduced AG G'.

Summarizing, we reduce arbitrary AG in three steps, each consisting of an interpretation. The first step consists of the NC-reduction-algorithm of Theorem 4.1; from the AG G_1 produced by it, we build a one-behaviour simple full interpretation G_2 and, finally, eliminating some productions from G_2, we obtain a reduced AG G' equivalent to the AG G from which we had started.

After this intuitive idea of the transformations involved in the process of reducing arbitrary AG, we start their formal description by giving the proof of Result 5.1.

Proof of Result 5.1. (Point 1) Note that the proof in Theorem 4.1, of the fact that the interpretation defined by the NC-reduction-algorithm is a full interpretation, does not depend on the assumption that the input AG is noncircular. Thus, the same argument proves that also in the case of an arbitrary AG G, the following holds: for every complete derivation tree t of G, there is a complete derivation tree t_1 of G_1, such that $t=\mu^{-1}(t_1)$ and which contains exactly those attributes which are useful in t. Hence, from Lemma 3.1, t_1 is reduced.

(Point 2) Point 2 is immediate from the fact that each production of G_1 satisfies condition (\$) of the NC-reduction-algorithm (cf. 2nd part of the proof of Lemma 4.1). □

Observation 5.1. In Theorem 4.1 we have shown that, if G is a noncircular AG, then the interpretation G_1, produced by the NC-reduction-algorithm from G, is such that the mapping μ^{-1} on complete derivation trees is one-to-one. From Result 5.1 it follows that this is, in general, not true in case G is an arbitrary AG.

In order to explain what trees correspond in G_1 to one complete derivation tree t of G, we introduce the following concept.

A *closed subgraph* of dtg(t) is a subgraph h of dtg(t) satisfying the following conditions: (i) the designated attribute d is in h; (ii) if an attribute a_0 of dtg(t) is in h, then all the attributes $a_1, \ldots, a_m$ from which, in dtg(t), there are edges entering a_0 (i.e., all the attributes which are arguments of the semantic rule defining a_0), and all these edges are in h; (iii) all attributes in h, apart from d, have at least one edge exiting them.

From the definition, it is not difficult to see that a closed subgraph of dtg(t) contains the useful subgraph of dtg(t) and if it contains attributes which are

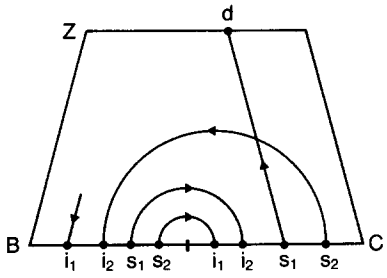

Fig. 9. The new production-graph of production $Z \rightarrow BC$

useless in t, they are, by condition (iii), connected to, or directly participating in a loose cycle (see Lemma 4.1). We claim that for each complete derivation tree t of G, G_1 has one corresponding complete derivation tree t_1 for each closed subgraph h of t and such that $\mathrm{dtg}(t_1)=h$.

The tree t_1 is obtained from t by relabeling each node F of t with (F, A'), where $A' \subseteq A(F)$ contains those attributes of F which are in h. The fact that t_1 is a tree of G_1 can be proven with a similar argument as that used in Theorem 4.1: a closed subgraph h of $\mathrm{dtg}(t)$, as the useful subgraph of which is a generalization, has the important property that all attributes in h are used to define other attributes in h, and this is the property which, by condition (\$) of the NC-reduction-algorithm, is satisfied by each production of G_1.

From this fact, it is also easy to show that for every complete derivation tree t_1 of G_1, if t is the tree of G such that $t=\mu^{-1}(t_1)$, then $\mathrm{dtg}(t_1)$ is a closed subgraph of $\mathrm{dtg}(t)$. □

As we stated above, for reducing an arbitrary AG we will need three successive transformations. In order to describe each of these transformations, we consider a circular AG G and follow its modifications step by step, until finally it will be in reduced form. The following example illustrates the first step consisting of the NC-reduction-algorithm.

Example 5.1. Consider the AG G which is the same as the AG of Example 4.1, apart from the dependencies of the attributes in production $Z \rightarrow BC$ which are now as in Fig. 9. Note that again the actual semantic rules associated with the productions of G are omitted because they have no relevance in our discussion.

As the AG of Example 4.1, G generates only three complete derivation trees t_1, t_2 and t_3 whose derivation tree graphs are represented in Fig. 10.

It is easy to see that $\mathrm{dtg}(t_1)$ contains a loose cycle which involves the attributes $i_2(B)$, $s_1(B)$, $i_2(C)$ and $s_2(C)$, while the other attributes are useful, t_2 is a reduced tree, and t_3 has some useful and some useless attributes. Because of the cycle in $\mathrm{dtg}(t_1)$, G is (conventionally) circular, but, because the cycle is loose (not connected to d), G is translationally noncircular. We will now describe the AG G_1 produced from G by the NC-reduction-algorithm. As in Example 4.1 we will only give the useful productions of G_1 and, in order to find them, we will make use of Observation 5.1. From this observation we

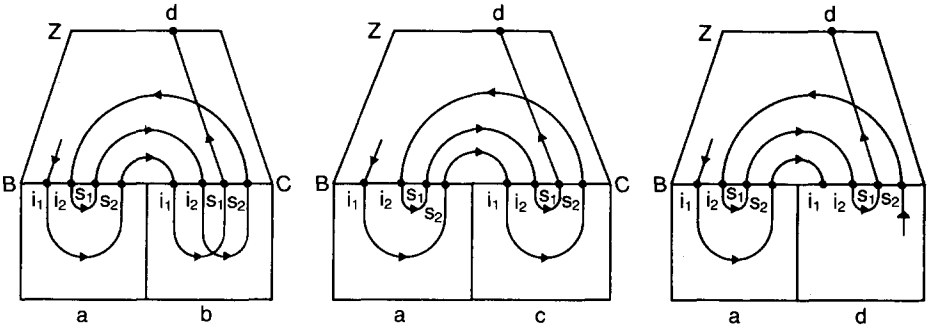

Fig. 10. The derivation tree graphs of t_1, t_2 and t_3

know that for each t_i of G, $i \in [1,3]$, G_1 has as many corresponding trees as there are closed subgraphs in t_i.

It is easy to see from Fig. 10 that, (i) t_1 has two closed subgraphs: the whole dtg(t_1) and the useful subgraph of dtg(t_1), (ii) t_2 and t_3 have only one closed subgraph each: the whole dtg(t_2), because t_2 is reduced, and the useful subgraph of dtg(t_3), respectively. Hence, G_1 has two trees t_1^c and t_1^r, corresponding to t_1 and two trees, t_2^r and t_3^r, corresponding to t_2 and t_3, respectively, (the superscript c stands for cyclic and r for reduced). The productions and nonterminals of G_1 generating t_1^c and t_2^r are easy to find. It is sufficient, in fact, to observe that the production of G used in t_1 and t_2 satisfy condition (\$) of the NC-reduction-algorithm (cf. Fig. 10). Thus, G_1 has nonterminals Z', B', and C', in $\mu(Z)$, $\mu(B)$, and $\mu(C)$, respectively, with the same attribute sets as Z, B, and C in G. Moreover, G_1 has productions $Z' \to B'C'$, $B' \to a$, $C' \to b$, $C' \to c$ with the same semantic rules as the corresponding productions of G (cf. the NC-reduction-algorithm). Clearly, these productions of G_1 generate t_1^c and t_2^r. For the two remaining trees t_1^r and t_3^r of G_1 we simply "take away", from dtg(t_1) and dtg(t_3) of Fig. 10, the useless attributes, keeping only their useful subgraphs. In this way it is easy to obtain the nonterminals and the productions of G_1 generating t_1^r and t_3^r. They are represented in Fig. 11. The first three productions generate t_1^r and the others t_3^r. These productions satisfy condition (\$) and, therefore, they are constructed by the NC-reduction-algorithm. It is also easy to see that the trees t_1^r and t_3^r that they produce, indeed are as we want, i.e., dtg(t_1^r) = useful subgraph of dtg(t_1) and dtg(t_3^r) = useful subgraph of dtg(t_3). □

As we said in the introductory part of this section, our second step in reducing arbitrary AG will be to show that for every AG G there is a simple full interpretation of G which is one-behaviour. A similar construction can be found also in [14, 5, 18, 22].

Theorem 5.1. *For every AG G, there is effectively an equivalent AG G' such that:*

(1) *G' is a simple full interpretation of G,*

(2) *G' is one-behaviour.*

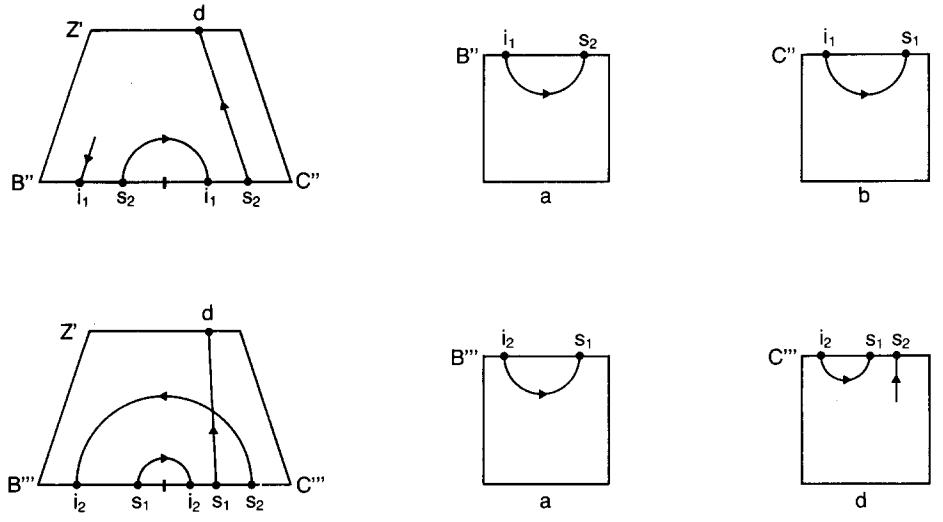

Fig. 11. The remaining useful productions of G_1

Proof. Let the underlying CFG's of G and G' be (V_t, V_n, P, Z) and (V_t, V_n', P', Z'), respectively.

We describe the interpretation G' of G by means of the following algorithm.

One-behaviour algorithm

(a) We define a *dfl*-substitution μ over $V_t \cup V_n$ as follows.

For every nonterminal F of G, $\mu(F)$ contains one symbol (F, g), for each possible is-graph g of F. Because Z has by definition only s-attributes, it has only one possible is-graph which we indicate with ϕ because it has no edges. Thus, $\mu(Z)=\{(Z, \phi)\}$. As usual $\mu(a)=\{a\}$ for every $a \in V_t$.

(b) The AG G' is defined as follows.

(i) G' has the same terminal alphabet V_t as G.

(ii) $V_n' = \bigcup \{\mu(F) | F \in G\}$, $(F, g) \in \mu(F)$ has the same attribute set as F, i.e., $A((F, g)) = A(F)$.

(iii) The start symbol of G' is $Z'=(Z, \phi)$ and it has the same designated attribute d as G.

(iv) The set P' of productions of G' is built as follows. For every production p of G of the usual form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, P' contains a production p' of the form $(F_0, g_0) \to w_0 (F_1, g_1) w_1 \dots w_{n-1} (F_n, g_n) w_n$, for every combination of is-graphs $g_0, \dots, g_n$ of $F_0, \dots, F_n$, respectively, which satisfies the following condition (*).

Condition (*): in g_0 there is an edge running from an i-attribute a to an s-attribute b of F_0 iff in the augmented production-graph $\mathrm{pg}(p)[g_1, \dots, g_n]$, there is a path running from a to b.

(v) The semantic rules in $r_{p'}$ are the same as those in r_p apart from the usual renaming of the nonterminals.
This ends the One-behaviour algorithm.

The fact that G' is a simple interpretation of G is clear from the One-behaviour algorithm. It is easy to prove, using induction on the height of derivation trees t' of G', that, if (F, g) is the root of t', then $\mathrm{is}(t')=g$. (and also $g=\mathrm{is}(t)$, where $t=\mu^{-1}(t')$). This clearly shows that G' is one-behaviour. From this it should also be easy to see that, corresponding to each nonterminal F of G, G' has as many useful (in the context-free sense) nonterminals as there are valid is-graphs of F in G. By Lemma 3.1(3), in order to show that G and G' are equivalent, it is sufficient to show that G' is a full interpretation of G. In proving this we use a technique similar to that of Theorem 4.1. Consider a complete derivation tree t of G and relabel every node F of t with the symbol (F, g), where g is the is-graph of the subtree of t rooted in F; let t' be the resulting tree. We show that t' is a complete derivation tree of G' with the following argument. Assume that production p: $F_0 \to w_0 F_1 w_1 \ldots w_{n-1} F_n w_n$ is used in t and that every F_i of p, through the relabeling, becomes (F_i, g_i) in t', $i \in [0, n]$. Because of the way the relabeling is done, it is clear that the is-graphs g_i respect condition (*) of the One-behaviour algorithm (see Preliminaries). Hence, $(F_0, g_0) \to w_0 (F_1, g_1) w_1 \ldots w_{n-1} (F_n, g_n) w_n$ is a production of G'. Since this is true for all the productions of t, t' is a tree of G'. This shows that G and G' are equivalent. □

Apart from the use we will soon make of it, the preceding theorem has an important consequence which we explain shortly as follows. In [16] the class of *absolutely noncircular* AG was introduced together with a particularly interesting attribute evaluation technique for them. It is easy to prove that a noncircular one-behaviour AG is also absolutely noncircular. Therefore, because of the fact that noncircularity is preserved by interpretations (cf. Sect. 3), Theorem 5.1 shows that for every noncircular AG there is effectively an equivalent absolutely noncircular AG. In the following example we continue our process of reducing the circular AG G of Example 5.1.

Example 5.2. Consider the AG G_1 which, in Example 5.1, was obtained from G by means of the NC-reduction-algorithm. We will now transform it into an equivalent one-behaviour AG G_2, using the One-behaviour algorithm of Theorem 5.1.

Observe that in G_1, apart from Z', whose only valid is-graph is anyhow ϕ, only nonterminal C' can derive two terminal strings (through productions $C' \to b$ and $C' \to c$), whereas all other nonterminals generate exactly one terminal string: B', B'' and B''' generate a, C'' generates b, and C''' generates d. This obviously implies that in G_1 C' has two valid is-graphs g_1 and g_2 which are represented in Fig. 12, whereas each of the other nonterminals of G_1 has only one valid is-graph. From these observations it is easy to see that the one-behaviour AG G_2, produced from G_1 by the One-behaviour algorithm, has two useful nonterminals, $C'_1=(C', g_1)$ and $C'_2=(C', g_2)$, corresponding to C', and only one corresponding to each of the other nonterminals of G_1. We keep

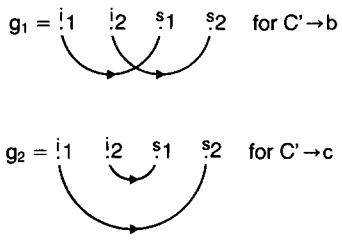

Fig. 12. The valid is-graphs of C' in G_1

for these last nonterminals of G_2 the same names as the corresponding nonterminals of G_1, so, if $g_{B''}$ is the only valid is-graph of B'', we will indicate the nonterminal of G_2 corresponding to B'' still with B'' rather than with $(B'', g_{B''})$.

From what we have said so far, it should be easy to understand that G_2 has two productions $Z' \to B' C'_1$ and $Z' \to B' C'_2$, corresponding to $Z' \to B' C'$ of G_1 and that, corresponding to $C' \to b$ and $C' \to c$ of G_1, G_2 has the productions $C'_1 \to b$ and $C'_2 \to c$, respectively. This roughly means that C', which has two valid is-graphs in G_1, gets split into two nonterminals of G_2, each with one of the is-graphs. Clearly, all the other productions of G_1 are passed unchanged to G_2: they involve nonterminals already having only one valid is-graph. Recall that G_2 is a simple interpretation of G_1. Hence, each nonterminal in G_2 has the same attributes as the corresponding nonterminal of G_1, and, therefore, every production of G_2 has the same semantic rules (thus, the same production-graph) as those of the corresponding production of G_1. Therefore, Fig. 10 and 11, which give the production-graphs for G_1 represent also those for G_2. However, we prefer, for questions of understanding, to repeat the production-graphs in Fig. 13. Note that in Fig. 13 the production-graphs of $Z' \to B' C'_1$ and $Z' \to B' C'_2$ are given together: they are both equal to that of $Z' \to B' C'$ of G_1.

The observations made in the proof of Theorem 5.1, should be sufficient for understanding that those represented in Fig. 13 are all the useful productions of G_2. □

We are finally at the last step of our transformation. Using Result 5.1 and Theorem 5.1 we can prove that every AG can be reduced.

Theorem 5.2. *For every AG G, there is effectively an equivalent AG G' such that,*
(1) *G' is a full interpretation of G,*
(2) *G' is in reduced form.*

Proof. G is transformed into G' in three steps, using all the previous results of the section.

Step 1. G is transformed into G_1 using the NC-reduction-algorithm of Theorem 4.1. $(G_1 \underset{\mu 1}{\trianglerighteq} G)$.

Step 2. G_1 is transformed into G_2 using the One-behaviour algorithm $(G_2 \underset{\mu 2}{\trianglerighteq} G_1)$.

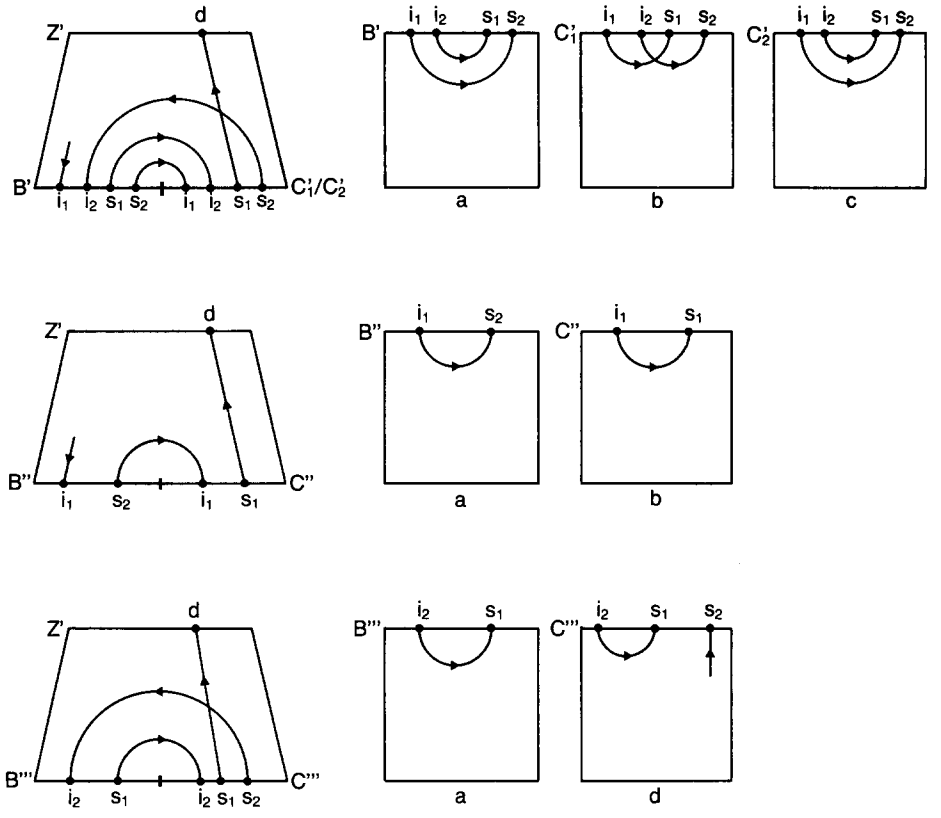

Fig. 13. All the production-graphs of the one-behaviour AG G_2

Step 3. Finally, from G_2 we obtain G' by eliminating from G_2 those productions in whose valid augmented production-graphs there are cycles which are not connected with any *s*-attribute of the left-hand side nonterminal. Recall that G_2 is one-behaviour and, therefore, each production of G_2 has only one valid augmented production-graph. Note that G' is an interpretation of G_2, i.e., $G' \underset{\mu 3}{\rhd} G_2$, where μ_3 is the identity *dfl*-substitution, but it is not a full interpretation of G_2.

The fact that $G' \underset{\mu}{\rhd} G$ follows immediately from the fact that the composition of interpretations is also an interpretation (cf. Sect. 3). Thus, $G' \underset{\mu 3}{\rhd} G_2 \underset{\mu 2}{\rhd} G_1 \underset{\mu 1}{\rhd} G$ implies $G' \underset{\mu}{\rhd} G$, where μ is the *dfl*-substitution obtained by composing $\mu 1$, $\mu 2$ and $\mu 3$.

Point (2) of the theorem is proved by the following argument. In any complete derivation tree t_1 of G_1, if an attribute is useless, then it is connected to, or directly participates in a loose cycle of $\mathrm{dtg}(t_1)$ (cf. Point 2 of Result 5.1 and Observation 5.1). By Lemma 3.1, this also holds for the trees of G_2 since G_2 is a simple full interpretation of G_1. Moreover G_2 is one-behaviour, there-

fore, each production p of G_2 has only one valid augmented production-graph which gives the dependencies among the attributes of p in whatever derivation tree p occurs. From this the following is easy to prove: let t_2 be a complete derivation tree of G_2, $\mathrm{dtg}(t_2)$ contains a loose cycle iff a production p of t_2 has in its valid augmented production-graph a cycle which is not connected with any s-attribute of the left-hand side nonterminal of p. (Hint: consider the top most loose cycle in $\mathrm{dtg}(t_2)$). Hence, since in Step 3 all these productions are eliminated from G_2 in order to obtain G', no complete derivation tree of G' contains a loose cycle. Thus, by Result 5.1(2), G' is in reduced form.

It remains to show that G and G' are equivalent. Again, by Lemma 3.1, it suffices to show that G' is a full interpretation of G. We do this by following the transformation of G into G' step by step.

Step 1. By Result 5.1(1), for every complete derivation tree t of G there is a reduced complete derivation tree t_1 of G_1 such that $t=\mu 1^{-1}(t_1)$ and $\mathrm{dtg}(t_1)$ = useful subgraph of $\mathrm{dtg}(t)$.

Step 2. Since, by Theorem 5.1, G_2 is a simple full interpretation of G_1, by Lemma 3.1, for every reduced complete derivation tree t_1 of G_1, there is a corresponding tree t_2 of G_2 such that $\mathrm{dtg}(t_1)=\mathrm{dtg}(t_2)$. Hence, t_2 is still reduced.

Step 3. Clearly a reduced tree does not contain loose cycle. Thus, from the above proof of point (2) of this theorem, none of the productions used in the reduced tree t_2 of G_2 is eliminated in Step 3. Hence, t_2 is also a derivation tree of G'.

Altogether this proves that G' is a full interpretation of G. Hence G and G' are equivalent. □

In the previous examples of this section we have transformed a given AG G by means of the NC-reduction and the One-behaviour algorithms into an equivalent AG G_2 which is one-behaviour and which satisfies Point 1 and 2 of Result 5.1. In the following Example we use the information embedded in the nonterminals of G_2 in order to obtain from it a reduced AG G' equivalent to the AG G from which we have started.

Example 5.3. Consider the one-behaviour AG G_2 constructed in Example 5.2 and whose production-graphs are in Fig. 13. It is not difficult to see that, among all the productions of G_2, only $Z'\to B'C_1'$, as shown in Fig. 14 (cf. with t_1 of Fig. 10), has a loose cycle in its valid augmented production-graph.

Thus, by Theorem 5.2, it is sufficient to eliminate $Z'\to B'C_1'$ from G_2 in order to obtain an AG G' such that $G' \underset{\mu}{\trianglerighteq} G$, where G is the AG from which we started in Example 5.1, and such that G' is in reduced form and equivalent to G.

In order to see that this is actually true, note that production $C_1'\to b$ is now useless in G' and that, with the remaining productions, G' can generate three reduced complete derivation trees t_1', t_2' and t_3' (whose derivation tree graphs are in Fig. 15). Recall now that G can also generate only three complete

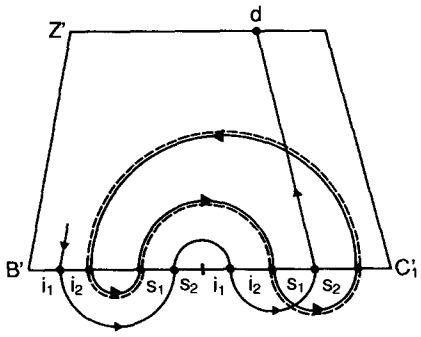

Fig. 14. The cyclic augmented production-graph of $Z' \to B' C'_1$ of G_2

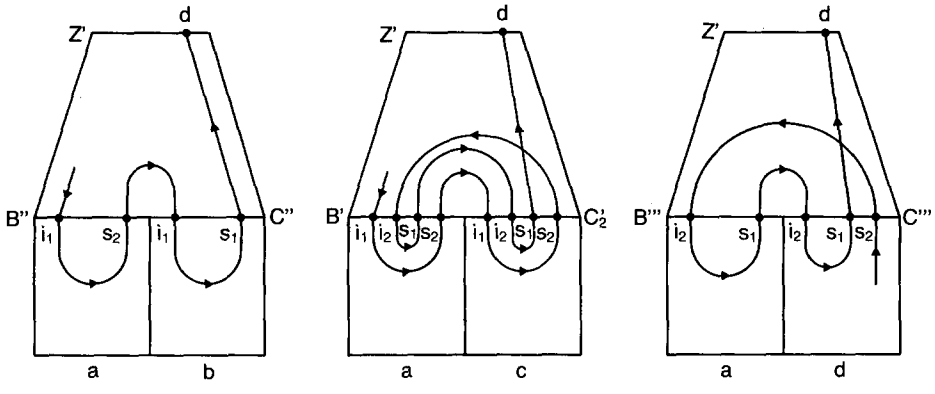

Fig. 15. The derivation tree graphs of the three complete derivation trees t'_1, t'_2 and t'_3 of G'

derivation trees t_1, t_2 and t_3 (see Fig. 10). At this point, it is sufficient to compare Fig. 10 and 15 to see that $t_i = \mu^{-1}(t'_i)$ and $\mathrm{dtg}(t'_i)$ is equal to the useful subgraph of $\mathrm{dtg}(t_i)$, for each $i \in [1, 3]$. □

We want to point out the fact that the AG G' in Example 5.3 is (conventionally) noncircular (see Fig. 15). This is particularly interesting because the AG G from which we started in Example 5.1 was (conventionally) circular and only translationally noncircular (see Fig. 10: the cycle in $\mathrm{dtg}(t_1)$ consists only of attributes useless in t_1). From this it can be seen that, reducing an AG having some translational property, means also transforming it into an equivalent AG which has the corresponding conventional property. Then, because conventional properties of AG are decidable, reduction can be used also for deciding translational properties. This intuition is made precise in the following results.

Theorem 5.3. *Let* $X \in \{$*noncircular, L, simple multi-pass, pure multi-pass*$\}$.

(i) *The translational X-property is decidable.*

(ii) *For any semantic domain D, the classes of translations over D defined by translationally X-AG, reduced X-AG, and conventionally X-AG, are equal.*

Proof. The proof of both points is based on the following facts.

(1) An AG in reduced form is translationally X iff it is conventionally X.

(2) If G' is a full interpretation of G, then, G is translationally X iff G' is also translationally X (see the end of Sect. 3).

Thus, in order to test whether an AG G is translationally X, it is sufficient to construct its equivalent reduced, full interpretation G', as in Theorem 5.2, and test whether G' is conventionally X which is decidable [17, 2, 9]. For showing point (ii), observe that every conventionally X-AG is obviously also translationally X and that, by Theorem 5.2 and (2) above, for every translationally X-AG there is an equivalent reduced translationally X-AG, which is, by (1) above, also conventionally X. □

Analogous results for other translational properties of AG can be proven in the same way: for example, for translational one-visit [7] and translational pure and simple multi-visit [19, 10].

We will now consider the time complexity of the process of reducing arbitrary AG which is described in Theorem 5.2. For doing this we need the notion of size of an AG G, denoted with $|G|$, which is the length of some coding of the nonterminals of G with their attributes, and of the productions of G with their semantic rules. It is easy to see that the first two steps of Theorem 5.2 are the most time consuming ones, whereas the third step is surely polynomial in the size of G_2. Let us start, then, by considering the first step which consists of the NC-reduction-algorithm.

Observe that, since the NC-reduction-algorithm suffices for reducing noncircular AG, from the analysis of its time complexity we will obtain an upper bound for this task.

Let G be the AG we want to reduce. For any production p of G of the usual form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$, we call n_p the number of attributes in Def(p), i.e., the attributes in $S(F_0)$ and $I(F_i)$, $i \in [1, n]$. Clearly, for each production p of G, $n_p < |G|$, since, by the consistency-requirement for AG, r_p must contain n_p semantic rules defining all attributes in Def(p). The NC-reduction-algorithm generates for G_1 2^{n_p} productions corresponding to p of G: each subset A' of Def(p) uniquely determines a production p' of G_1 corresponding to p, for which $r_{p'}$ contains those semantic rules of r_p which define the attributes in A' and $A(p')$ contains A' and the attributes used as arguments in these semantic rules. It is not difficult to see that in this way we construct all the productions p' of G_1, corresponding to p of G, and which satisfy condition (\$) of the NC-reduction-algorithm. It is clear that the size of each production p' is polynomial in $|G|$ and that it can be constructed in time polynomial in its size. From this it follows that the time taken by the NC-reduction-algorithm is $O(2^{d_1|G|})$. This result implies that any noncircular AG G can be reduced in time $O(2^{d_1|G|})$.

Observe that, because G_1 is an interpretation of G, for each production p of G_1 the number n_p of attributes in Def(p) is still smaller than $|G|$. For the same reason, if m is the maximum number of attributes associated to a nonterminal of G, then m is also an upper bound on the number of attributes of each nonterminal of G_1. Clearly $m < |G|$. Let us now consider the 2nd step of the

reduction process. From each production p of G_1, the One-behaviour algorithm constructs as many productions as there are combinations of is-graphs of the nonterminals of p.

Let p be of the usual form $F_0 \to w_0 F_1 w_1 \dots w_{n-1} F_n w_n$. Assigning an is-graph g_0 to F_0 can be viewed as assigning a set of i-attributes of F_0 to each s-attribute a of F_0: those i-attributes from which, in g_0, there is an edge entering a. Similarly, assigning an is-graph g_i to each F_i, $i \in [1, n]$, can be viewed as assigning a set of s-attributes of F_i to each i-attribute a of F_i: those s-attributes to which, in g_i, there are edges from a. Thus, since Def(p) consists of $S(F_0)$ and $I(F_i)$, $i \in [1, n]$, this means to associate a set of at most m attributes to each of the n_p attributes in Def(p). Therefore, the One-behaviour algorithm constructs for G_2 at most $2^{n_p \cdot m}$ productions corresponding to production p of G_1. Constructing each of these productions and testing whether it satisfies condition (*) of the One-behaviour algorithm can be done in time polynomial to $|G|$. Thus, the time taken by the One-behaviour algorithm is $O(2^{d_2|G|^2})$ which, since the third step is clearly polynomial in the size of G_2, is also an upper bound for the time taken by the whole reduction process. Therefore, the following upper bounds hold for reducing noncircular and arbitrary AG.

Result 5.2. (1) Every noncircular AG G can be reduced in time $O(2^{d_1|G|})$, for some $d_1 > 0$.

(2) Every AG G can be reduced in time $O(2^{d|G|^2})$, for some $d > 0$. □

As last remark of the section, we note that, if in Step 3 of Theorem 5.2, instead of eliminating from G_2 only the productions having a loose cycle in their valid augmented production-graph, we would eliminate the productions having any cycle, then the obtained AG G'' would obviously be reduced as G' is, and, moreover, would also be translationally and conventionally noncircular. It is easy to see that doing this means to eliminate from G_2 not only the non reduced trees, but also those reduced trees whose meaning is undefined (i.e., with cyclic useful subgraphs). Hence, G'' is equivalent to G' and, by Theorem 5.2, to G. This shows that for any AG G there exists effectively an equivalent AG G'' which is reduced and conventionally (and translationally) noncircular.

We note that a similar result was also shown in [5], but using a conventional definition of meaning of a tree and, hence, of translation and equivalence. This result was, in fact, the reason for introducing in [5] a construction similar to the One-behaviour algorithm of Theorem 5.1.

Consider, in fact, the following conventional definition of meaning of a complete derivation tree t of an AG G: the meaning of t is the value of its designated attribute if dtg(t) does not contain any cycle, otherwise it is undefined. Assume now that this concept of meaning of a tree is used in Definition 2.1(1) and (2) of translation and equivalence of AG. From the fact that the One-behaviour algorithm defines a simple full interpretation of the input AG, Theorem 5.1 holds also with this definition of equivalence. Thus, for every AG G there is an equivalent one-behaviour AG G_1. If we eliminate from G_1 all the productions whose valid augmented production-graphs contain cycles, we obtain an AG G' which is conventionally noncircular and equivalent to G_1 (all

and only the trees with cycles have been eliminated) and, hence, is also equivalent to G. This shows [5] that for every AG G there exists effectively a (recall conventionally!) equivalent AG G' which is conventionally noncircular.

6. Reducing is Hard

From Result 5.2 we know that any noncircular (arbitrary) AG G can be reduced in time $O(2^{d_1|G|})$, ($O(2^{d|G|^2})$, respectively). In this section we will show that reducing even the simple class of L-AG is hard. We will prove, in fact, that there are infinitely many L-AG G such that any algorithm for reducing AG takes for them more than $O(2^{c|G|/\log|G|})$ steps, for some constant $c>0$. This lower bound proves that no substantial improvement can be made on the upper bound for reducing noncircular AG, whereas some improvement could be possible on the upper bound for reducing arbitrary AG. However we conjecture that a sharper lower bound could be found for the latter case.

In order to prove that reducing AG is hard, we actually prove a stronger result which concerns a more general notion of reduction than the one used so far. We will, in fact, consider output preserving reductions instead of translation preserving ones as we have done in the previous sections. Recall from Sect. 2 that the output set of an AG G is the range of the translation of G. Thus, from now on, a reduction algorithm is an algorithm which from an AG G over some semantic domain D, produces a reduced AG G', also on D, and with the same output set as G. It is clear that, if we prove a particular lower bound to hold for the time complexity of every reduction algorithm of this new type, this bound also holds for the translation preserving reduction algorithms.

In what follows we will show that there is a semantic domain AND such that any reduction-algorithm, which, given an AG G on AND, produces a reduced AG G' still on AND and with the same output set as G, takes more than $2^{cn/\log n}$ steps infinitely often, where n is the size of G and c is a constant.

In order to prove this result, we use a construction of [14] which shows that the problem of recognizing an exponential time language can be reduced in polynomial time to that of deciding whether an AG is (conventionally) circular. Obviously, by means of this construction, it was proven in [14] that the circularity problem for AG is exponential hard. A slightly different construction is used in [9] for proving that some other problems concerning AG are also exponential time hard, as, for example the problem of deciding whether any given AG is pure multi-pass. We will recall the construction of [9] in some detail and, later, use it for showing the exponential hardness of reducing AG.

Theorem 6.1 [14, 9]. *Let K be an exponential time language, i.e. $K \in \mathrm{DTIME}(2^{d^n})$ for some d. For every word w of length n there is an AG G_w such that:*

(1) *G_w is an L-AG;*

(2) *G_w can be constructed in deterministic time $O(n \log n)$ where the constant depends on K only;*

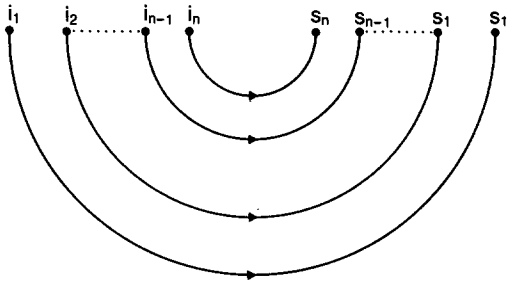

Fig. 16

(3) *a designated nonterminal B of* G_w *has i-attributes* $i_1, \ldots, i_n$ *and s-attributes* $s_1, \ldots, s_n$;

(4) $w \in K$ *iff there is a derivation tree t of* G_w, *with root B, such that its is-graph* $is(t)$ *has edges running from* i_k *to* s_k, *for all* $k \in [1, n]$ (see Fig. 16);

(5) *For every derivation tree t of* G_w *with root B, if e is an edge of* $is(t)$ *then e runs from* i_k *to* s_k *for some* $k \in [1, n]$.

(6) *All the semantic rules used in* G_w *are of two simple types: they are either identity assignment or constant assignment rules, i.e., of the form* $a(F) = b(F')$ *or* $a(F) = constant$, *respectively.* □

By means of this result it is possible to show exponential lower bounds for many problems concerning AG (see [14], [9]). Because of point(2) of Theorem 6.1, this lower bound can be taken as $2^{cn/\log n}$ for some constant c (cf. [14] for the reasoning involved). For this reason, instead of always having to repeat the actual time bound, we prefer to use the following "ad hoc" terminology (see also [9]).

A property P, concerning a class Y, is *exponential time hard* to decide if there is a constant $c > 0$, such that any deterministic Turing Machine which decides whether an arbitrary $X \in Y$ has property P must run for more than $2^{cn/\log n}$ steps for infinitely many X, where n is the size of X.

As last thing, before showing that reducing AG is hard, we define the semantic domain AND as follows: the possible values of the attributes are 1 and 0 and the functions which can be used in semantic rules are the constants, the identity and the AND-operation with any number of arguments, i.e., the allowed semantic rules are of the forms, $a(F) = 1/0$, $a(F) = b(F')$ and $a(F_0) = \text{AND}(a_1(F_1), \ldots, a_m(F_m))$.

Theorem 6.2. *There is a constant* $d > 0$ *such that any algorithm which, from a given AG G over the domain AND, produces a reduced AG G' still over AND and with the same output set as G, takes more than* $2^{dn/\log n}$ *steps infinitely often, where n is the size of G. This holds even if we restrict G to be L and hence, noncircular.*

Proof. Using Theorem 6.1, we first prove, following a suggestion of [15], that it is exponential hard to decide whether 1 is in the output set of an AG over the

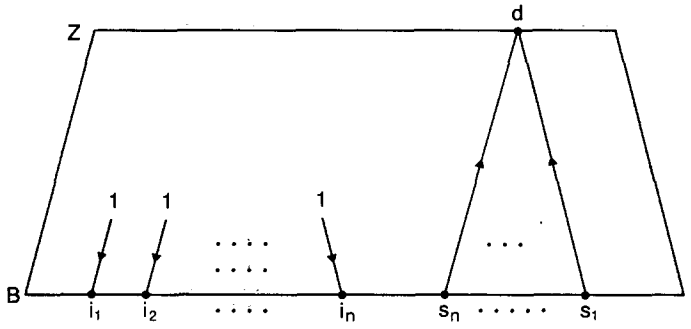

Fig. 17. The production-graph of $Z \rightarrow B$ of G'_w

domain AND, and then we show that any reduction algorithm could be used to decide this problem and, therefore any algorithm for reducing AG must run $O(2^{cn/\log n})$ steps infinitely often. We change the AG G_w of Theorem 6.1 into an AG G'_w over the domain AND as follows.

1) Add to G_w the production $Z \rightarrow B$ (recall that B is the special nonterminal of Theorem 6.1(3)), Z is the initial symbol of G'_w and has only the designated s-attribute d. The production-graph of $Z \rightarrow B$ is represented in Fig. 17, the corresponding semantic rules are as follows: for all $k \in [1, n]$, $i_k = 1$ and $d = \text{AND}(s_1, \ldots, s_n)$.

2) All the productions of G_w are also productions of G'_w, but the semantic rules associated to them are modified as follows. Recall from point (6) of Theorem 6.1 that G_w has semantic rules of only two types: identity assignment and constant assignment. In G'_w the identity assignment rules are unchanged, whereas the rules of the form $a(F) = \text{constant}$ become in G'_w, $a(F) = 0$.

From Theorem 6.1 the following is immediate: w is in the exponential time language K iff 1 is in the output set of G'_w. Hence, the problem of deciding whether 1 is in the output set of an AG over the domain AND is exponential time hard.

Let us now assume that we have a reduction algorithm and that we feed it which any AG G over AND obtaining a reduced AG G^r also over AND and with the same output set as G. Clearly, 1 is in the output set of G iff it is in that of G^r. In G^r we say that a production is of type 1 if all its constant assignment semantic rules are of the form $a(F) = 1$. We claim that the following is true: 1 is in the output set of G^r iff there is at least one complete derivation tree of G^r consisting only of type 1 productions. For proving this, it is sufficient to observe that G^r is reduced and is on the domain AND. It is clear, in fact, that in a complete tree t of G^r, if even only one attribute is assigned the value 0, then the value of the designated attribute d is also 0 whereas if all constant assignment semantic rules of t assign value 1, then all attributes of t and, in particular d, have value 1. Hence, if $\bar{G}$ is the subgrammar of the underlying CFG of G^r, consisting only of type 1 productions, then, 1 is in the output set of G^r iff $L(\bar{G}) \neq \phi$.

Let us draw some conclusions from this fact. On the one hand, we know that deciding whether 1 is in the output set of any AG G over AND takes more than $2^{cn/\log n}$ steps infinitely often. On the other hand we have just shown that we can decide this problem by producing from G a reduced AG G^r with the same output set and then testing whether the subgrammar $\bar{G}$ of the underlying CFG of G^r generates an empty language or not. Hence the following is true: the time for reducing G ($R(n)$), plus the time for testing whether $L(\bar{G})$ is empty ($T(\bar{G})$), must be more than $2^{cn/\log n}$ for infinitely many AG G (over AND). It is well known that, testing whether the language generated by a given CFG is empty, can be done in polynomial time in the size of the grammar (namely n^3). Thus, because the size of $\bar{G}$ is at most $R(n)$ (the time taken by the reduction). We have that $T(\bar{G})$ is at most $(R(n))^3$. Therefore $R(n) + (R(n))^3 > R(n) + T(\bar{G}) > 2^{cn/\log n}$ and so $(R(n))^4 > 2^{cn/\log n}$ and finally $R(n) > 2^{\frac{c}{4} n/\log n}$

This proves the theorem where the constant d, mentioned there, is $c/4$. □

This result may not seem very interesting because it only applies to AG on the particular domain AND. Note however that, if we ask from a reduction-algorithm to be general, that is, to reduce any AG on any domain, then, clearly, our result shows that any such reduction-algorithm would run for exponential time infinitely often.

As a last thing, we point out that the same result could be proven in a very similar way for other semantic domains, for instance, the domain of all strings over some finite alphabet and whose only operation is string concatenation. Of course, whether for a given domain D an efficient reduction algorithm for AG over D exists, depends completely on the properties of the sets and functions in D.

References

1. Alblas, H.: A characterization of attribute evaluation in passes. Acta Informat. **16**, 427-464 (1981)
2. Bochmann, G.V.: Semantic evaluation from left to right; CACM **19**, 55-62 (1976)
3. Bochman, G.V.: Semantic equivalence of Covering attribute grammars. Int. J. Comput. Inf. Sci. **8**, 523-539 (1979)
4. Cremers, A., Ginsburg, S.: Context free grammar forms; J. Comput. Syst. Sci. **11**, 86-117 (1975)
5. Döbeling, K.: Festlegung Zweier Sprachklassen mit Hilfe Attributierter Grammatiken. Diplomarbeit, Technische Universität Hannover, 1978
6. Duske, J., Parchmann, R., Sedello, M., Specht, J.: *IO*-macrolanguages and attributed translations. Inf. Control **35**, 87-105 (1977)
7. Engelfriet, J., Filè, G.: The formal power of one-visit attribute grammars. Acta Informat. **16**, 275-302 (1981)
8. Engelfriet, J., Filè, G.: Formal properties of one-visit and multi-pass attribute grammars. Proc. of the 7th ICALP at Noordwÿkerhout, pp. 182-194, Lecture Notes in Computer Science 85, Berlin, Heidelberg, New York: Springer 1980
9. Engelfriet, J., Filè, G.: Passes and paths of attribute grammars; Inf. Control **49**, 125-169 (1981)
10. Engelfriet, J., Filè, G.: Simple multi-visit attribute grammars. J. Comput. Syst. Sci. **24**, 283-314 (1982)
11. Engelfriet, J., Filè, G.: Passes, sweeps and visits. In: Proc. of the 8th ICALP at Akko, pp. 193-207, Lecture Notes in Computer Science 115. Berlin, Heidelberg, New York: Springer 1981

12. Fisher, M.J.: Grammars with macro-like productions. Ph.D. Thesis, Harvard University, 1968 (Also: Proc. 9-th Symp. on SWAT, pp. 131-142, 1968)
13. Hopcroft, J.E., Ullman, J.D.: Introduction to automata Theory, language and computation. Addison-Wesley Publishing Company, Reading, Massachusetts, 1979
14. Jazayeri, M., Ogden, W.F., Rounds, W.C.: The intrinsically exponential complexity of the circularity problem for attribute grammars. CACM **18**, 679-706 (1975)
15. Jones, N.D.: Personal communication
16. Kennedy, K., Warren, S.K.: Automatic generation of efficient evaluators for attribute grammars. Conf. Record of 3rd Symp. of Principles of Programming Languages, pp. 32-49, 1976
17. Knuth, D.E.: Semantics of context-free languages. Math. Syst. Theor. **2**, 127-145 (1968). Correction: Math. Syst. Theor. **5**, 95-96 (1971)
18. Riis, H.: Subclasses of attribute grammars. DAIMI PB-114, Aarhus University 1980
19. Riis, H., Skyum, S.: *k*-visit attribute grammars. Math. Syst. Theory **15**, 17-28 (1981)
20. Rosenkrantz, D.J., Stearns, R.E.: Properties of deterministic topdown grammars. Inf. Control **17**, 226-256 (1970)
21. Salomaa, A.: Grammatical families. Proc. of the 7-th ICALP at Noordwÿkerhout, pp. 543-554. Lecture Notes in Computer Science 85. Berlin, Heidelberg, New York: Springer, 1980
22. Vuurboom, R.: Restricting semantic behaviour of attribute grammar. Internal report, Twente University of Technology, 1980
23. Wood, D.: Grammar and *L*-forms: An Introduction. Lecture Notes in Computer Science 91. Berlin, Heidelberg, New York: Springer, 1980

Received April 15, 1982/September 12, 1982/October 14, 1982