Acta Informatica 16, 427–464 (1981)

A Characterization of Attribute Evaluation in Passes

Henk Alblas

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

Contents *page*

Summary 427
1. Introduction 428
2. Basic Concepts 430
3. Evaluation in Passes 431
   3.1. Pure Multi-Pass Evaluation 433
   3.2. Simple Multi-Pass Evaluation 433
4. A First Investigation of Simple Multi-Pass Evaluation: Some Relations 436
5. Comparison of Pure and Simple Multi-Pass Evaluation: Some Examples 442
6. Simple Left-to-right and/or Right-to-left Multi-Pass Evaluation 446
7. Simple Left-to-right Multi-Pass Evaluation 454
8. Simple Alternating Multi-Pass Evaluation 459
9. Conclusions and Further Research 463
References 463

Summary. This paper describes the evaluation of semantic attributes in a bounded number of passes from left-to-right and/or from right-to-left over the derivation tree of a program. Evaluation strategies where different instances of the same attribute in any derivation tree are restricted to be evaluated in one pass, with for every derivation tree the same pass number, are referred to as simple multi-pass whereas the unrestricted pass-oriented strategies are referred to as pure multi-pass.

A graph theoretic characterization is given, showing in which cases an attribute grammar meets the simple multi-pass requirements and what are the minimal pass numbers of its attributes for a given sequence of pass directions. For the special cases where only left-to-right passes are made or where left-to-right and right-to-left passes strictly alternate, new algorithms are developed that associate minimal pass numbers with attributes and indicate in case of failure the attributes that cause the rejection of the grammar. Mixing of a simple multi-pass strategy with other evaluation strategies, in case the grammar is not simple multi-pass, is discussed.

1. Introduction

Attribute grammars [10] are used as a formal tool to describe programming languages and their compilers.

Several methods have been developed to evaluate the semantic attributes within the derivation tree of a program [3, 5, 7–9, 11, 12]. An overview is given in [6].

In [3] Bochmann suggested to evaluate the attributes in a fixed number of depth-first left-to-right traversals (called passes) of the tree. In [7, 8] Jazayeri and Walter extended this method by making alternately left-to-right and right-to-left passes.

In both papers broadly the same algorithm is presented to determine for a given input grammar the number of passes necessary to evaluate all the attributes of the nodes within the derivation tree of any program. Both algorithms presuppose that the evaluation strategy is restricted in such a way that with each attribute it is possible to associate a fixed pass number such that the evaluation of all instances of that attribute in all derivation trees of the grammar can be performed in that pass.

A grammar meeting this requirement can have a simple evaluation strategy. During each pass the evaluator simply consults the pass number of an attribute to decide whether an instance of that attribute has to be computed. If the above requirement is dropped the evaluation strategy has to be more complicated. During each pass it is necessary to check for each attribute instance whether it is already defined and if not, to check whether the arguments of the corresponding evaluation rule are already defined, in order to decide whether this instance should be evaluated.

In this paper we will refer to the restricted way of multi-pass evaluation as *simple* multi-pass evaluation and to the unrestricted way of multi-pass evaluation as *pure* multi-pass evaluation.

Since Bochmann and Jazayeri and Walter did not consider pure multi-pass evaluation, a comparison of pure and simple multi-pass evaluation could not be made. In this paper we make a clear distinction between both strategies, consider their formal properties and especially investigate the limitations of simple multi-pass evaluation compared to pure multi-pass evaluation.

This paper is organized as follows:

Section 2 provides an introduction to the basic concepts associated with attribute grammars.

In Sect. 3 we consider the idea of attribute evaluation in passes and define the distinction between pure and simple multi-pass evaluation.

In Sect. 4 we present a first investigation of simple multi-pass evaluation. The principle that the same pass number is associated with different instances of the same attribute in each context leads to precedence relations among attributes. These relations characterize the possible distributions of the attributes over the passes.

We formalize these relations into a graph model where the vertices are associated with attributes, arcs denote dependencies and labels of the arcs indicate whether the direction of a pass is consistent with the "direction" of the dependency. Special attention is paid to the role of cycles in this precedence graph. For every cycle in the graph, the same pass number should be associated with all its vertices and its labels should all be consistent with one of the possible pass directions.

In Sect. 5 we compare the results of pure and simple multi-pass evaluation by discussing some examples. These examples show that the pure multi-pass strategies apply to more attribute grammars than the simple ones.

In Sect. 6 we further investigate the simple multi-pass strategies in the context of the precedence graph and show that for each sequence of pass directions each attribute has a minimal pass-number which can be expressed in terms of the labels of the paths ending in the attribute. Also a graph theoretic characterization is given, indicating when an attribute grammar meets the simple multi-pass requirements: an attribute grammar is simple multi-pass if and only if none of its attributes are involved in a cycle whose labels are not consistent with one of the possible pass directions.

At the end of this section we consider a graph theoretic version of the algorithm of Bochmann [3] and Jazayeri and Walter [8] to associate pass numbers with attributes. This algorithm computes for each sequence of pass directions the minimal pass number for each attribute and thus can be viewed as a "path finding" algorithm in the above graph.

In Sects. 7 and 8 simple multi-pass evaluation with respect to two special sequences of pass directions is considered. The case where only left-to-right passes are executed is discussed in Sect. 7, whereas in Sect. 8 attention is paid to the case where left-to-right and right-to-left passes strictly alternate.

Both sections end with two algorithms to compute pass numbers. For both sequences of pass directions we consider an adapted version of the algorithm mentioned above.

We also present new algorithms which are special cases of a "path finding" algorithm of Aho, Hopcroft and Ullman [1]. If a grammar is simple multi-pass with respect to one of these special sequences of pass directions they produce the minimal pass numbers for that sequence and in case of failure they indicate the attributes that cause the rejection of the grammar, i.e., the attributes that are involved in a cycle whose labels are not consistent with one of the possible pass directions. The latter algorithms produce sufficient information of the paths in the graph to be able to mix the simple multi-pass strategy with other evaluation strategies in case the attribute grammar is not completely simple multi-pass.

The importance of the algorithm for strict alternating simple multi-pass evaluation follows from the fact that: an attribute grammar is simple multi-pass with respect to any sequence of pass directions if and only if it is simple multi-pass with respect to the sequence of pass directions where left-to-right and right-to-left passes strictly alternate.

In Sect. 9 we mention subjects for further research.

2. Basic Concepts

An attribute grammar is a context-free grammar augmented with attributes and attribute evaluation rules.

The underlying context-free grammar is (as usual) a four-tuple $G = (V_N, V_T, P, S)$. The finite sets V_N of nonterminal and V_T of terminal symbols form the vocabulary $V = V_N \cup V_T$. P is the finite set of productions and $S \in V_N$ is the start symbol, which does not appear on the right-hand side of any production.

Associated with each symbol $X \in V$ is a finite set $A(X)$ of attributes. $A(X)$ is partitioned into two disjoint subsets $I(X)$ and $S(X)$ of inherited and synthesized attributes respectively. For $X = S$ and for $X \in V_T$ we require $I(X) = \emptyset$.

The set of all attributes will be denoted by A, i.e., $A = \bigcup_{X \in V} A(X)$. In this paper we consider the attributes $a \in A(X)$ and $a \in A(Y)$ as different if the symbols X and Y are different. If necessary we will denote an attribute a of symbol X by $a(X)$.

Let P consist of r productions, numbered from 1 to r and let the p-th production be

$$X_{p0} \rightarrow X_{p1} X_{p2} \dots X_{pn_p}$$

where $n_p \geqq 0$, $X_{p0} \in V_N$ and $X_{pk} \in V$ for $1 \leqq k \leqq n_p$.

Production p is said to have the attribute occurrence (a, p, k) if $a \in A(X_{pk})$. The set of all attribute occurrences of production p will be denoted by $\mathbf{AO}(p)$. This set can be partitioned into two disjoint subsets of defined occurrences and used occurrences denoted by $\mathbf{DO}(p)$ and $\mathbf{UO}(p)$ respectively.

These subsets are defined as follows:

$$\begin{aligned}\mathbf{DO}(p) = {} & \{(s, p, 0) \mid s \in S(X_{p0})\} \cup \\ & \{(i, p, k) \mid i \in I(X_{pk}) \wedge 1 \leqq k \leqq n_p\}, \\ \mathbf{UO}(p) = {} & \mathbf{AO}(p) - \mathbf{DO}(p) \\ = {} & \{(i, p, 0) \mid i \in I(X_{p0})\} \cup \\ & \{(s, p, k) \mid s \in S(X_{pk}) \wedge 1 \leqq k \leqq n_p\}.\end{aligned}$$

Associated with each production p are a number of attribute evaluation rules which specify for each attribute occurrence in $\mathbf{DO}(p)$ how to compute the value of such an occurrence as a function of certain other attribute occurrences in $\mathbf{AO}(p)$.

For production p an attribute evaluation rule is written as

$$(a, p, k) := f((a_1, p, k_1), (a_2, p, k_2), \dots, (a_m, p, k_m))$$

where $(a, p, k) \in \mathbf{DO}(p)$, f is a total function and $(a_j, p, k_j) \in \mathbf{AO}(p)$ for $1 \leqq j \leqq m$. It is easy to transform every attribute evaluation rule (by a sequence of substitutions) such that only attribute occurrences in $\mathbf{UO}(p)$ appear as arguments of function f (cf. [3]). This means that the extra conditions

$$(a_j, p, k_j) \in \mathbf{UO}(p) \quad \text{for } 1 \leqq j \leqq m$$

are satisfied. Therefore, we will allow only used occurrences as arguments in attribute evaluation rules.

If production p has an associated attribute evaluation rule such that attribute occurrence (a, p, j) is used as argument for the evaluation of attribute occurrence (b, p, k) then we say: (b, p, k) *depends on* (a, p, j).

For each sentence of G a derivation tree exists. The nodes of the tree are labeled with symbols from V.

For each interior node there is a production $X_{p0} \rightarrow X_{p1} X_{p2} \quad X_{pn_p}$, such that the node is labeled with X_{p0} and its n_p sons are labeled with $X_{p1}, X_{p2}, \ldots, X_{pn_p}$ respectively. We say that production p applies at that node.

Given a derivation tree, instances of attributes are attached to the nodes in the following way: If node η is labeled with grammar symbol X, then for each attribute $a \in A(X)$ an instance of a is attached to node η. We say that the derivation tree has attribute instance $a(\eta)$.

Let η_0 be a node, p the production applied at η_0 and $\eta_1, \eta_2, \ldots, \eta_{n_p}$ the sons of η_0 from left-to-right respectively.

An attribute evaluation instruction

$$a(\eta_k) := f(a_1(\eta_{k_1}), a_2(\eta_{k_2}), \ldots, a_m(\eta_{k_m}))$$

is associated with attribute instance $a(\eta_k)$ if the attribute evaluation rule

$$(a, p, k) := f((a_1, p, k_1), (a_2, p, k_2), \ldots, (a_m, p, k_m))$$

is associated with production p.

The task of an attribute evaluator is to compute the values of all attribute instances attached to the derivation tree, by executing the attribute evaluation instructions associated with these attribute instances. In general the order of evaluation is free, with the only restriction that an attribute evaluation instruction cannot be executed before the values of its arguments are defined. Initially the values of all attribute instances attached to the derivation tree are undefined, with the exception of the (synthesized) attribute instances associated with terminal symbols. The latter are determined by the parser.

At each step an attribute instance is chosen, whose value can be computed. The evaluation process continues until all attribute instances in the tree are defined or until none of the remaining attribute instances can be evaluated.

An attribute grammar is circular if a derivation tree exists for which it is not possible to evaluate all attribute instances.

3. Evaluation in Passes

In order to choose an attribute evaluation strategy which holds for every derivation tree of an attribute grammar, one could analyse the dependency relations between the attribute occurrences in the evaluation rules of the grammar. Such a strategy is flexible [6] in the sense that the way of walking along the nodes of the derivation trees is determined by the dependencies of the attribute occurrences in the grammar.

As opposed to such a flexible strategy a rigid strategy can be considered, where the visiting order of the nodes is chosen a priori, i.e., independent of the attribute dependencies.

An example of such a rigid strategy is to make a number of passes over the derivation tree, where a *pass* is defined to be a depth-first left-to-right or right-to-left traversal of the derivation tree.

If no further restrictions are made, attribute evaluation in left-to-right passes is defined by the following algorithm (which works for every noncircular attribute grammar).

Algori-hm 3.1. Attribute evaluation in left-to-right passes.

Input: an attributed derivation tree where only the attribute instances of the terminal symbols are evaluated.

Output: an attributed derivation tree where all attribute instances are evaluated.

Algorithm:
```
begin
   procedure visit subtree ($n$: node);
      begin {assume production $p$ is applied at node $n$}
         for $k$ from 1 to $n_p$
         do if $X_{pk} \in V_N$
            then for each $a \in I(X_{pk})$
                   do if instance of $a$ is undefined
                          and all argument instances of the
                              evaluation rule for $a$ are defined
                      then evaluate instance of $a$
                      fi
                   od;
                   visit subtree ($k$-th descendant of $n$)
            fi
         od;
         for each $a \in S(X_{p0})$
         do if instance of $a$ is undefined
               and all argument instances of the
                   evaluation rule for $a$ are defined
            then evaluate instance of $a$
            fi
         od
      end {of visit subtree};
   while not all attribute instances are evaluated
   do visit subtree (root) od
end □
```

Each call of visit subtree (root) corresponds to a pass.

Observe that each attribute instance is evaluated at the earliest possible pass and that it may happen that different instances of the same attribute are evaluated at different passes.

3.1. Pure Multi-Pass Evaluation

When the number of passes cannot exceed a finite upper bound, we refer to the evaluation strategy described in Algorithm 3.1 as *pure multi-pass evaluation.* By "pure" we mean that besides the restriction on the number of passes no other restrictions are imposed on the grammar.

Remark. The only reason why Algorithm 3.1 is displayed in this paper is to explain evaluation in passes (and not in order to be used in practice).

Definition 3.1. For $m \geqq 1$, an attribute grammar is *pure LR m-pass* if for each derivation tree of the grammar evaluation is possible in at most m left-to-right passes.

An attribute grammar is *pure LR multi-pass* if it is pure LR m-pass for some m. □

If an attribute grammar is pure $LR\,m$-pass the while statement of Algorithm 3.1 can be replaced by

```
for $i$ from 1 to $m$
do visit subtree (root) od
```

In Algorithm 3.1 only left-to-right (**for** k **from** 1 **to** n_p) passes were made. In the following we also discuss evaluation algorithms where alternately left-to-right and right-to-left (**for** k **from** n_p **downto** 1) passes are executed. In the case of multi-pass evaluation where passes in *B*oth *D*irections (left-to-right and right-to-left) are allowed we indicate the directions of the successive passes by a sequence $\langle d_1, \ldots, d_m \rangle$ where d_i $(1 \leqq i \leqq m)$ denotes the direction of the i-th pass, which is either L (left-to-right) or R (right-to-left).

Definition 3.2. For $m \geqq 1$, an attribute grammar is *pure BD m-pass* if for some sequence $\langle d_1, \ldots, d_m \rangle$ of pass directions, for each derivation tree of the grammar evaluation is possible in at most m passes (where pass i has direction d_i for $1 \leqq i \leqq m$).

An attribute grammar is *pure BD multi-pass* if it is pure $BD\,m$-pass for some m. □

3.2. Simple Multi-Pass Evaluation

The pure multi-pass evaluation strategy is inefficient in the sense that during each pass for each attribute instance it is necessary to check whether the instance is already defined and if it is not defined to check whether the arguments of the corresponding evaluation instruction are already defined to be able to decide that this instance can or cannot be evaluated.

In order to achieve more efficiency with respect to the evaluation algorithm we impose a further restriction on the grammar and require that all different instances of the same attribute in any derivation tree have to be evaluated in the same pass, with for each derivation tree the same pass number. With this restriction in mind we try, for a given attribute grammar to partition the set A

of attributes into a sequence $\langle A_1, \ldots, A_m \rangle$ of mutually disjoint subsets of A, such that the instances of the attributes in set A_i $(1 \leqq i \leqq m)$ can be evaluated during the i-th pass. If such a partition exists, a simple evaluation strategy can be used. During pass i simply the set A_i is consulted in order to decide that an instance of an attribute has to be computed.

We will refer to this evaluation strategy as *simple multi-pass evaluation.*

Since we will also consider attribute grammars for which only a subset of the set A of attributes can be evaluated in passes by a simple multi-pass evaluator, we need also "partial partitions" which do not exhaust the whole of A.

Definition 3.3. A *partial partition* of the set A of attributes is a sequence $\langle A_1, \ldots, A_m \rangle$ of mutually disjoint subsets of A. It is *complete* if $\bigcup_{i=1}^{m} A_i = A$. □

Simple multi-pass evaluation by making left-to-right passes only is defined by the following algorithm, where $\langle A_1, \ldots, A_m \rangle$ is a given partial partition of A.

Algori-hm 3.2. Simple left-to-right multi-pass evaluation.

Input: an attributed derivation tree where only the attribute instances of the terminal symbols are defined.

Output: an attributed derivation tree where all attribute instances of $\bigcup_{i=1}^{m} A_i$ are defined.

Algorithm:

```
begin
   procedure visit subtree (n: node; i: pass number);
      begin {assume production p is applied at node n}
         for k from 1 to $n_p$
         do if $X_{pk} \in V_N$
            then for each $a \in I(X_{pk})$
                  do if $a \in A_i$
                     then evaluate instance of a
                     fi
                  od;
                  visit subtree (k-th descendant of n, i)
            fi
         od;
         for each $a \in S(X_{p0})$
         do if $a \in A_i$
            then evaluate instance of a
            fi
         od
      end {of visit subtree};
   for i from 1 to m
   do visit subtree (root, i) od
end □
```

We define a partial partition of the set A of attributes of an attribute grammar to be *correct* when Algorithm 3.2 always works, i.e., when 'evaluate instance of a' never gives the error message that one of the argument instances of the evaluation instruction for a is undefined.

Definition 3.4. A partial partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar is *LR-correct* if the instances of all attributes in set A_i $(1 \leqq i \leqq m)$ can be evaluated during the i-th pass of Algorithm 3.2. □

Definition 3.5. An attribute grammar is *simple LR m-pass* if an *LR*-correct complete partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes exists. An attribute grammar is *simple LR multi-pass* if it is simple *LR m*-pass for some m. □

In the following we also discuss simple multi-pass evaluation where left-to-right as well as right-to-left passes are allowed.

Simple *BD* multi-pass evaluation where $\langle A_1, \ldots, A_m \rangle$ is a given partial partition of the set A of attributes and $\langle d_1, \ldots, d_m \rangle$ is a given sequence of pass directions, is defined by an algorithm which is almost the same as Algorithm 3.2: the first for statement of procedure visit subtree has to be changed into

```
**if** $d_i = L$
**then for** $k$ **from** 1 **to** $n_p$ **do**... **od**
**else for** $k$ **from** $n_p$ **downto** 1 **do**... **od**.
```

We refer to the algorithm for simple *BD* multi-pass evaluation as Algorithm 3.2-*BD*.

Definition 3.6. A partial partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar is *BD-correct with respect to a given sequence* $\langle d_1, \ldots, d_m \rangle$ *of pass directions* if the instances of all attributes in set A_i $(1 \leqq i \leqq m)$ can be evaluated during the i-th pass of Algorithm 3.2-*BD*. □

Definition 3.7. An attribute grammar is *simple BD m-pass* if with respect to some sequence $\langle d_1, \ldots, d_m \rangle$ of pass directions a *BD*-correct complete partition of the set A of attributes exists.

An attribute grammar is *simple BD multi-pass* if it is simple *BD m*-pass for some m. □

Given a partition of the set of attributes of an attribute grammar a pass function can be defined indicating for each attribute the number of the pass during which its instances should be evaluated.

Definition 3.8. For each partial partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar a *pass function* pass(a), where a is an attribute of A, is defined as follows:

$$\begin{aligned} \text{pass}(a) &= i & &\text{if } a \in A_i \\ \text{pass}(a) &= \infty & &\text{if } a \in A - \bigcup_{i=1}^{m} A_i. \end{aligned}$$

The pass function is *complete* if the partition is complete.
The pass function is *LR-correct* if the partition is *LR*-correct.
The pass function is *BD-correct* with respect to some sequence of pass directions if the partition is *BD*-correct with respect to that sequence. □

It should be clear that there is a one-to-one correspondence between partial partitions and pass functions. Notice that for a complete partition pass(a) is finite for all $a \in A$.

We end this section with three observations concerning the relationship of simple multi-pass evaluation and other attribute evaluation strategies.

1. The simple multi-pass evaluation strategies are not flexible inasmuch for the development of a flexible strategy the starting point has to be an analysis of the dependencies of the grammar. See for example [9]. For the simple multi-pass strategy the starting point is to make a bounded number of passes with given directions over the possible derivation trees. In the second place dependency relations of the grammar are analysed in order to find the number of passes necessary, and to distribute the attributes over the passes. The problem how to find for a given grammar the sequence of pass directions such that the number of passes is minimized, will not be discussed in this paper. See [12].

2. Given an LR/BD correct partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar, it is easy to see that after the i-th pass of the pure LR/BD multi-pass evaluator all instances of attributes in A_i have been evaluated. From this we conclude: if an attribute grammar is simple LR/BD m-pass then for each derivation tree the number of passes required for pure multi-pass evaluation is $\leqq m$.

3. Figure 1 shows the relationship between pass-oriented evaluation strategies. By simple LR, pure LR, simple BD and pure BD we denote respectively the classes of attribute grammars that are simple LR, pure LR, simple BD and pure BD multi-pass. From observation 2 we immediately conclude: simple $LR \subseteq$ pure LR and simple $BD \subseteq$ pure BD. Given an LR-correct partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar, this partition is clearly a BD-correct partition with respect to the sequence $\langle d_1, \ldots, d_m \rangle$ of pass directions where $d_i = L$ for $1 \leqq i \leqq m$. Hence, simple $LR \subseteq$ simple BD. In the same way, pure $LR \subseteq$ pure BD. In Sect. 5 we will show by discussing some examples that the inclusions are proper and that the unconnected classes (pure LR and simple BD) are incomparable.

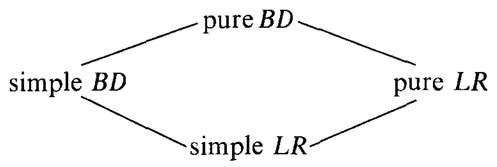

Fig. 1. Relationship between pass-oriented evaluation strategies

4. A First Investigation of Simple Multi-Pass Evaluation: Some Relations

Let AG be an attribute grammar and A the set of attributes of AG. To be able to construct a correct partition of A (and to define the associated pass function) we have to consider:

1. the dependency relations between attribute occurrences in the evaluation rules,

2. the order in which attribute occurrences are considered during a left-to-right or a right-to-left pass.

Starting from the dependency relations between attribute occurrences and for the present ignoring the effects of the directions of the passes, we define a precedence relation 'prec' among attributes.

Definition 4.1. The relation *a prec b* between attributes *a* and *b* holds if a production $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$ exists with attribute occurrences (a,p,j) and (b,p,k) such that (b,p,k) depends on (a,p,j). □

Example 4.1. Consider attribute grammar *AG0* with $S=Z$, $V_N=\{Z,A,B\}$, $V_T=\{t\}$, and

attributes: $I(Z)=\emptyset$	$I(A)=\{\text{in}(A)\}$	$I(B)=\{\text{in}(B)\}$
$S(Z)=\{\text{result}(Z)\}$	$S(A)=\{\text{out}(A)\}$	$S(B)=\{\text{out}(B)\}$

productions	*evaluation rules*
1: $Z \to AB$	$\text{result}(Z) := \text{out}(A)$
	$\text{in}(A) := \text{out}(B)$
	$\text{in}(B) := \text{constant}$
2: $A \to t$	$\text{out}(A) := \text{in}(A)$
3: $B \to t$	$\text{out}(B) := \text{in}(B)$

For attribute grammar *AG0* the only possible sentence is *tt*.

Figure 2 shows the attributed derivation tree of *tt* with attribute instances and their dependencies.

From the dependencies in the evaluation rules of the grammar we find the precedence relations:

prod. 1: out(A) prec result(Z) prod. 2: in(A) prec out(A)
out(B) prec in(A) prod. 3: in(B) prec out(B). □

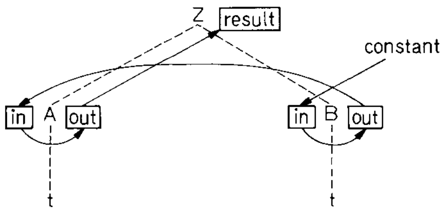

Fig. 2. Attributed derivation tree of grammar *AG0*

Let *a* and *b* be attributes of attribute grammar *AG* and let pass be a correct pass function.

Notice that if *a* prec *b* then pass(a)≦pass(b) because the instances of *b* cannot be computed in an earlier pass than the instances of *a*; moreover for pass(a) and pass(b) finite, pass(a)=pass(b) can only hold if during the concerning pass for each instance of attribute *b* that depends on a certain instance of attribute *a*, that instance of *a* is found before the instance of *b*. Hence we have to introduce stronger precedence relations between attributes *a* and *b* that indicate, given the direction of the concerning pass, whether for each pair of instances of a and b such that the instance of b depends on the instance of a, the instance of a is found before the instance of b.

Before considering such relations between attributes, we discuss the corresponding relations between attribute occurrences.

The direction of a pass imposes an ordering relation among the instances of the attributes in the derivation trees and hence also of the occurrences in each production.

Consider a production p: $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$. For a left-to-right pass the visiting order of the attribute occurrences is as follows: occurrences of inherited attributes of X_{p0}, occurrences of inherited attributes of X_{p1}, occurrences of synthesized attributes of $X_{p1}, \dots$, occurrences of inherited attributes of X_{pn_p}, occurrences of synthesized attributes of X_{pn_p}, occurrences of synthesized attributes of X_{p0}. For a right-to-left pass the only difference is that the symbols of the right-part of production p are visited in the order $X_{pn_p}, \dots, X_{p2}, X_{p1}$.

In accordance with this we define, for Left-to-Right and Right-to-Left passes respectively, precedence relations between used attribute occurrences and defined attribute occurrences which indicate the order in which the occurrences are found during such a pass.

Definition 4.2. In production p: $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$ with $(b,p,k) \in \mathbf{DO}(p)$ and $(a,p,j) \in \mathbf{UO}(p)$ the relation (a,p,j) *LR-occurs-before* (b,p,k) holds if and only if the following condition is satisfied: if $1 \leqq k \leqq n_p$ then $j < k$. □

Definition 4.3. In production p: $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$ with $(b,p,k) \in \mathbf{DO}(p)$ and $(a,p,j) \in \mathbf{UO}(p)$ the relation (a,p,j) *RL-occurs-before* (b,p,k) holds if and only if the following condition is satisfied: if $1 \leqq k \leqq n_p$ then $j = 0$ or $j > k$. □

Now taking into account the direction of the passes we define the following precedence relations between attributes.

Definition 4.4. The relation $a L b$ between attributes a and b holds if a prec b and for each production $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$ with attribute occurrences (a,p,j) and (b,p,k) such that (b,p,k) depends on (a,p,j), (a,p,j) LR-occurs-before (b,p,k). □

Note that if a prec b then at least one production p exists such that (b,p,k) depends on (a,p,j) for some j, k.

Definition 4.5. The relation $a \bar{L} b$ between attributes a and b holds if a prec b but not $a L b$. □

$a \bar{L} b$ indicates that a production exists where a defined occurrence of b depends on a used occurrence of a, but where the used occurrence of a is not found before the defined occurrence of b.

Now we define analogous relations concerning right-to-left passes.

Definition 4.6. The relation $a R b$ between attributes a and b holds if a prec b and for each production $X_{p0} \to X_{p1} X_{p2} \dots X_{pn_p}$ with attribute occurrences (a,p,j) and (b,p,k) such that (b,p,k) depends on (a,p,j), (a,p,j) RL-occurs-before (b,p,k). □

Definition 4.7. The relation $a \bar{R} b$ between attributes a and b holds if a prec b but not $a R b$. □

Example 4.2. From the dependencies in the evaluation rules of grammar *AG0* (see Example 4.1) we find the precedence relations:

$\mathrm{out}(A)\ L\ \mathrm{result}(Z)$	$\mathrm{out}(A)\ R\ \mathrm{result}(Z)$
$\mathrm{out}(B)\ \bar{L}\ \mathrm{in}(A)$	$\mathrm{out}(B)\ R\ \mathrm{in}(A)$
$\mathrm{in}(A)\ L\ \mathrm{out}(A)$	$\mathrm{in}(A)\ R\ \mathrm{out}(A)$
$\mathrm{in}(B)\ L\ \mathrm{out}(B)$	$\mathrm{in}(B)\ R\ \mathrm{out}(B)$.

To understand the consequences of Definitions 4.4 up to and including 4.7 we give the following theorems, which characterize the correct partitions in terms of the introduced precedence relations.

Theorem 4.1. A partial partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar AG is BD-correct with respect to a given sequence $\langle d_1, \ldots, d_m \rangle$ of pass directions if and only if for the corresponding pass function 'pass' and for all attributes a and b:

(i) if a prec b then $\mathrm{pass}(a) \leq \mathrm{pass}(b)$.

(ii) if $a \bar{L} b$ and $\mathrm{pass}(a)$ or $\mathrm{pass}(b)$ is finite and a left-to-right pass then $\mathrm{pass}(a) < \mathrm{pass}(b)$.

(iii) if $a \bar{R} b$ and $\mathrm{pass}(a)$ or $\mathrm{pass}(b)$ is finite and a right-to-left pass then $\mathrm{pass}(a) < \mathrm{pass}(b)$.

Proof. ($\Rightarrow$)

(i) From a prec b it follows that the instances of b cannot be computed in an earlier pass than the instances of a.

(ii) If $\mathrm{pass}(b) = \infty$ then $\mathrm{pass}(a) < \mathrm{pass}(b)$. Now assume that $\mathrm{pass}(b)$ is finite. If $a \bar{L} b$ then a prec b and not $(a L b)$. From a prec b follows that $\mathrm{pass}(a) \leq \mathrm{pass}(b)$; from not $(a L b)$ follows that the instances of a and b cannot be computed during the same left-to-right pass. Hence $\mathrm{pass}(a) < \mathrm{pass}(b)$.

(iii) Proceeds along the same lines as (ii).

($\Leftarrow$) we prove this part of the theorem by induction on the size, i.e., the number of subsets of the partition.

I. $A_1 = \{a \mid \mathrm{pass}(a) = 1\}$. From attribute grammar AG a grammar AG_1 is constructed in the following way.

(1) A_1 is the set of attributes of grammar AG_1.

(2) Each evaluation rule $(a, p, k) := f(\ldots)$, where a is in A_1, is an evaluation rule of grammar AG_1.

Clearly, since by (i) there is no attribute $b \in A_1$ such that one of its occurrences depends on an occurrence of an attribute $a \notin A_1$, AG_1 is a proper attribute grammar.

Now we distinguish two cases:

1.: $d_1 = L$, i.e., A_1 is associated with a left-to-right pass.

2.: $d_1 = R$, i.e., A_1 is associated with a right-to-left pass.

Case 1. By (ii), for attributes a and $b \in A_1$, if a prec b then not $(a \bar{L} b)$ and hence $a L b$. Hence, all dependencies between attribute occurrences satisfy the LR-occurs-before condition. Consequently (cf. [3]), grammar AG_1 is simple LR one-pass. Hence $\langle A_1 \rangle$ is an LR-correct and, with respect to the sequence $\langle L \rangle$ of pass directions, also a BD-correct (partial) partition of the set A of attributes.

Case 2. From (iii) it follows (analogously to the case $d_1 = L$) that all dependencies between attribute occurrences of grammar AG_1 satisfy the RL-occurs-before condition. Hence $\langle A_1 \rangle$ is, with respect to the sequence $\langle R \rangle$ of pass directions, a BD-correct (partial) partition of the set A of attributes.

II. Induction step.

Induction hypothesis: with respect to the sequence $\langle d_1, \ldots, d_i \rangle$ of pass directions, $\langle A_1, \ldots, A_i \rangle$ is a BD-correct partial partition. We have to prove that $\langle A_1, \ldots, A_i, A_{i+1} \rangle$ is a BD-correct partial partition with respect to the sequence $\langle d_1, \ldots, d_i, d_{i+1} \rangle$.

$A_{i+1} = \{a \mid \text{pass}(a) = i+1\}$. From attribute grammar AG a grammar AG_{i+1} is constructed in the following way.

(1) A_{i+1} is the set of attributes of grammar AG_{i+1}.

(2) Each evaluation rule $(a, p, k) := f(\ldots)$, where a is in A_{i+1}, is an evaluation rule of grammar AG_{i+1}; in these evaluation rules the occurrences of attributes in $\bigcup_{j=1}^{i} A_j$ are replaced by constants.

Clearly, since by (i) there is no attribute $b \in A_{i+1}$ such that one of its occurrences depends on an occurrence of an attribute $a \notin A_{i+1}$, AG_{i+1} is a proper attribute grammar.

Now assume that $d_{i+1} = L$ (for $d_{i+1} = R$ the proof is similar).

From (ii) it follows that for attributes a and $b \in A_{i+1}$, if a prec b then not $(a \bar{L} b)$ and so $a L b$. Hence, all dependencies between attribute occurrences satisfy the LR-occurs-before condition. Hence, as before, grammar AG_{i+1} is simple LR one pass, i.e., all attributes in A_{i+1} can be evaluated in the $(i+1)$th pass. This implies that, with respect to the sequence $\langle d_1, \ldots, d_i, d_{i+1} \rangle$ of pass directions, $\langle A_1, \ldots, A_i, A_{i+1} \rangle$ is a correct partial partition of the set A of attributes.

Thus we have proved that $\langle A_1, \ldots, A_m \rangle$ is a BD-correct (partial) partition with respect to the sequence $\langle d_1, \ldots, d_m \rangle$ of pass directions. □

Clearly, for left-to-right passes only, the following theorem holds.

Theorem 4.2. A partial partition $\langle A_1, \ldots, A_m \rangle$ of the set A of attributes of an attribute grammar AG is LR-correct if and only if for the corresponding pass function 'pass' and for all attributes a and b:

(i) if a prec b then $\text{pass}(a) \leqq \text{pass}(b)$.

(ii) if $a \bar{L} b$ and $\text{pass}(a)$ or $\text{pass}(b)$ is finite, then $\text{pass}(a) < \text{pass}(b)$. □

Theorems 4.1 and 4.2 show that knowledge of the relations L, $\bar{L}$, R and $\bar{R}$ between the attributes of an attribute grammar AG suffices to determine whether AG is simple LR- and/or BD multi-pass.

To make the discussion of these relations easier, we introduce a graph model determined by these relations. For each attribute grammar AG a directed graph is set up as follows.

Each vertex of the graph is associated with an attribute of AG. Arc (a,b) is contained in the graph if the relation a prec b holds between attributes a and b. To each arc two labels are assigned in the following way: if the relation $a L b$ holds, then arc (a,b) has label L, otherwise $\bar{L}$; if the relation $a R b$ holds, then arc (a,b) has label R, otherwise $\bar{R}$.

This labeled directed graph associated with attribute grammar AG will be denoted by $P_{BD}(AG)$ and will be called the *BD-precedence graph* of attribute grammar AG. For LR multi-pass evaluation we only need the labels L and $\bar{L}$. The graph where the labels R and $\bar{R}$ are omitted will be denoted by $P_{LR}(AG)$ and will be called the *LR-precedence graph* of attribute grammar AG.

Notice that the vertices of the precedence graphs are attributes and not attribute instances. The requirement that different instances of the same attribute have to be evaluated during the same pass makes that different dependency properties of different instances of the same attribute cannot be distinguished.

Example 4.3. The BD-precedence graph of grammar $AG0$ (see Example 4.1) is given in Figure 3.

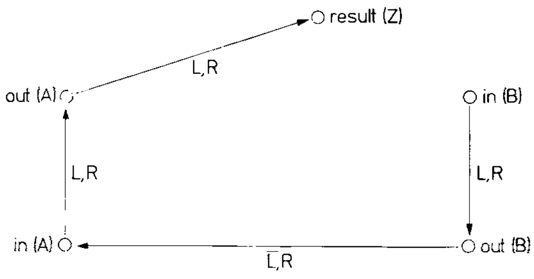

Fig. 3. $P_{BD}(AG0)$ □

In the context of precedence graphs we need the notions of path and cycle. For paths we will use the following notation. $p[a_1, a_2, \ldots, a_n]$ for $n \geq 1$ stands for a (possibly empty) path of length $n-1$, composed of the arcs (a_1, a_2), $(a_2, a_3), \ldots, (a_{n-1}, a_n)$. $p[a]$ stands for an empty path with attribute a involved. A cycle is a path $p[a_1, a_2, \ldots, a_n]$ with $n \geq 2$ and $a_n = a_1$.

In the following we will prove that the labeling of the cycles plays an essential role in the acceptance or rejection of an attribute grammar in the context of the simple multi-pass strategies. For that reason we define the notions of $\bar{L}$-, $\bar{R}$- and $\bar{L}-\bar{R}$-cycle.

Definition 4.8. With respect to the precedence graphs $P_{LR}(AG)$ and $P_{BD}(AG)$ of an attribute grammar AG,
an *$\bar{L}$-cycle* is a cycle where at least one of the arcs has label $\bar{L}$;
an *$\bar{R}$-cycle* is a cycle where at least one of the arcs has label $\bar{R}$;
an *$\bar{L}-\bar{R}$-cycle* is a cycle where at least one of the arcs has label $\bar{L}$ and at least one of the arcs has label $\bar{R}$. □

Now we collect some easy conclusions of Theorems 4.1 and 4.2 in a corollary.

Corollary 4.1. Let $p[a_1, a_2, \ldots, a_n]$ be a cycle in the graphs $P_{LR}(AG)$ and $P_{BD}(AG)$ of attribute grammar AG.

If attributes $a_1, a_2, \quad , a_n$ have a finite pass number during simple multi-pass evaluation, the following holds:

(i) all instances of all attributes of the cycle have to be evaluated during the same pass, i.e., $\text{pass}(a_1) = \text{pass}(a_2) = \ldots = \text{pass}(a_n)$;

(ii) if the cycle is an $\bar{L}$-cycle (respectively an $\bar{R}$-cycle), then it is impossible to evaluate the instances of the attributes of the cycle during a left-to-right pass (respectively a right-to-left pass);

(iii) if the cycle is an $\bar{L}-\bar{R}$-cycle, then it is impossible to evaluate the instances of the attributes of the cycle during any pass.

Proof. (i) From Theorem 4.1 (i) it follows that $\text{pass}(a_i) \leqq \text{pass}(a_{i+1})$ for $1 \leqq i \leqq n-1$. Hence, $\text{pass}(a_1) \leqq \text{pass}(a_2) \leqq \quad \leqq \text{pass}(a_n) = \text{pass}(a_1)$. Hence, all pass numbers are equal.

(ii) Assuming that the pass direction is L, from $a_i \bar{L} a_{i+1}$ for some i $(1 \leqq i \leqq n-1)$ follows (Theorem 4.1 (ii)): $\text{pass}(a_i) < \text{pass}(a_{i+1})$. This contradicts the equality of the pass numbers of the attributes of the cycle. For the pass direction R the proof is analogous.

(iii) Follows immediately from (ii). □

In Sects. 6 and 7 we will prove that an attribute grammar is simple BD multi-pass *if and only if* its BD-precedence graph has no $\bar{L}-\bar{R}$-cycles and that an attribute grammar is simple LR multi-pass *if and only if* its LR-precedence graph has no $\bar{L}$-cycles.

5. Comparison of Pure and Simple Multi-Pass Evaluation: Some Examples

To give an impression of the limitations of simple multi-pass evaluation in comparison with pure multi-pass evaluation, we consider some examples. Firstly we consider three attribute grammars where the simple LR or even the simple BD multi-pass strategy fails, but where the pure strategies succeed. Secondly we discuss a grammar which belongs to the classes pure BD and simple BD but not to the classes pure LR and simple LR.

In all examples the initial nonterminal Z will have one synthesized attribute result(Z), whereas all other nonterminals X will have one inherited attribute in(X) and one synthesized attribute out(X). Multiple occurrences of an X in the same derivation tree or in different derivation trees are distinguished by subscripts in order to be able to distinguish different instances of the same attribute of X in the possible derivation trees of the grammar. In the description of the grammars the evaluation rules are omitted. The dependencies between the attribute occurrences can be found in the possible derivation trees.

At the end of this section – after having discussed the properties of the example grammars – we will come back to the classification of pass-oriented attribute evaluation strategies.

Example 5.1. Consider attribute grammar *AG1* with $V_N = \{Z, E\}$, $V_T = \{e\}$, and $P = \{Z \rightarrow EE, E \rightarrow e\}$.

The only possible sentence is *ee*. Figure 4 shows the attributed derivation tree of *ee* with attribute instances and their dependencies. The associated *BD*-precedence graph is given in Fig. 5.

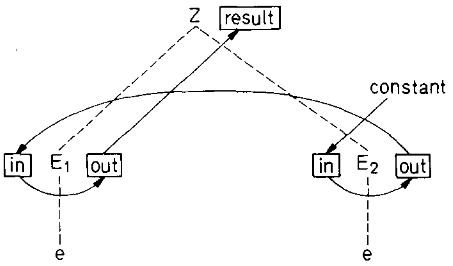

Fig. 4. Attributed derivation tree of grammar *AG1*

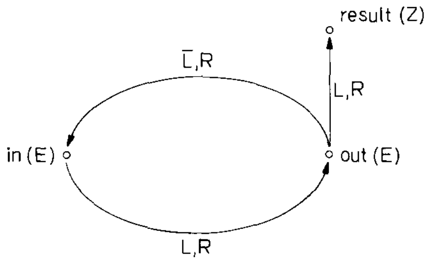

Fig. 5. $P_{BD}(AG1)$

The grammar is pure *LR* 2-pass. During the first pass the attribute instances $\text{in}(E_2)$ and $\text{out}(E_2)$ are evaluated, during the second pass the instances $\text{in}(E_1)$, $\text{out}(E_1)$ and $\text{result}(Z)$.

Considering the associated *LR*-precedence graph we find that the grammar is not simple *LR* multi-pass. The graph includes the $\bar{L}$-cycle $p[\text{in}(E), \text{out}(E), \text{in}(E)]$. Hence, the instances of attributes $\text{in}(E)$ and $\text{out}(E)$ cannot be evaluated during a left-to-right pass (Corollary 4.1 (ii)).

Observe however that all attribute instances can be evaluated during one right-to-left pass. Hence, the grammar is simple *BD* one-pass.

Notice that in the derivation tree of the grammar there is no nonempty dependency path from instance $\text{in}(E_1)$ to itself or from instance $\text{in}(E_2)$ to itself, although in the associated *BD*-precedence graph there is a nonempty path from attribute $\text{in}(E)$ to itself. Hence, the graph $P_{BD}(AG1)$ suggests a dependency that does not exist in the only possible derivation tree of grammar *AG1*. The reason for this suggested dependency is that in the precedence graph it is impossible to distinguish different instances of attribute $\text{in}(E)$. The same holds for attribute $\text{out}(E)$.

Other examples of such suggested dependencies can be found in the associated precedence graphs of the following grammar examples. □

Example 5.2. Consider attribute grammar *AG2* with $V_N=\{Z,C,D,E\}$, $V_T=\{d,e\}$, and $P=\{Z\rightarrow EC,\ C\rightarrow ED,\ E\rightarrow e,\ D\rightarrow d\}$.

The only possible sentence is *eed*. Figure 6 shows the attributed derivation tree of *eed* with attribute instances and their dependencies. The associated *BD*-precedence graph is given in Fig. 7.

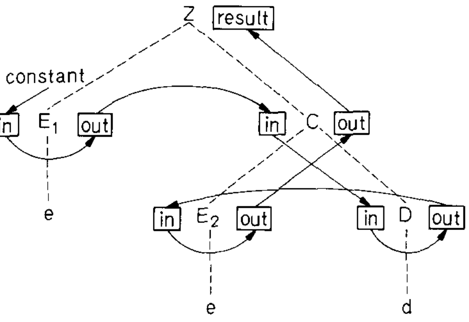

Fig. 6. Attributed derivation tree of grammar *AG2*

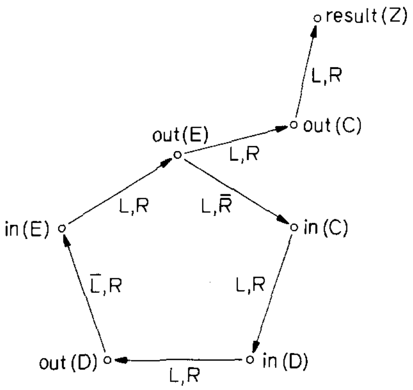

Fig. 7. $P_{BD}(AG2)$

The grammar is pure *BD* 2-pass with respect to the sequences $\langle L,L\rangle$, $\langle L,R\rangle$ and $\langle R,R\rangle$ of pass directions. With respect to the sequences $\langle L,L\rangle$ and $\langle L,R\rangle$ the same order of evaluation holds: during the first pass the instances in(E_1), out(E_1), in(C), in(D) and out(D); during the second pass the instances in(E_2), out(E_2), out(C) and result(Z). Observe that for the sequence $\langle R,R\rangle$ the evaluation of the instances in(C), in(D) and out(D) has to be deferred to the second pass.

Clearly, since the grammar is pure *BD* 2-pass with respect to the sequence $\langle L,L\rangle$, the grammar is also pure *LR* 2-pass.

All simple multi-pass evaluation strategies fail! The graph $P_{BD}(AG2)$ includes the $\bar{L}-\bar{R}$-cycle $p[\text{in}(E), \text{out}(E), \text{in}(C), \text{in}(D), \text{out}(D), \text{in}(E)]$. Hence, the instances of the attributes which form part of this cycle cannot be evaluated during any pass (Corollary 4.1(iii)).

Example 5.3. Consider attribute grammar *AG3* with $V_N = \{Z, D, E\}$, $V_T = \{a, b, d, e\}$, and $P = \{Z \rightarrow aED, Z \rightarrow bED, D \rightarrow d, E \rightarrow e\}$.

The possible sentences are *aed* and *bed*. Figure 8 shows the attributed derivation trees for these sentences. The associated *BD*-precedence graph is given in Fig. 9.

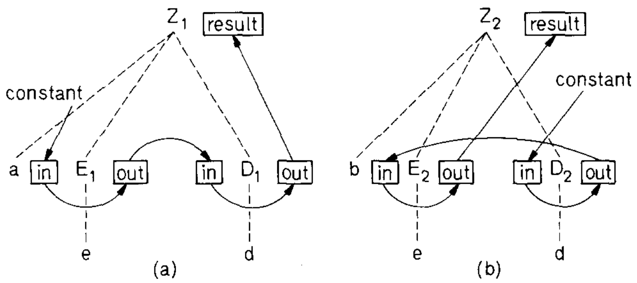

Fig. 8. Attributed derivation trees of grammar *AG3*

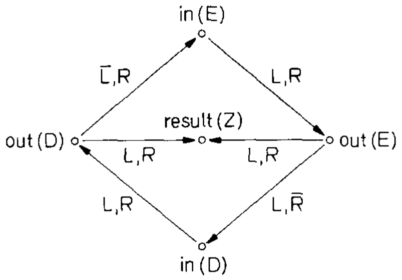

Fig. 9. $P_{BD}(AG3)$

Pure multi-pass evaluation is possible for any sequence $\langle d_1, d_2 \rangle$ of pass directions.

From the existence of the $\bar{L}-\bar{R}$-cycle $p[\text{in}(E), \text{out}(E), \text{in}(D), \text{out}(D), \text{in}(E)]$ in graph $P_{BD}(AG3)$ follows that simple *BD* multi-pass evaluation is not possible.

Observe that, in order to construct a path from in(E) to itself in graph $P_{BD}(AG3)$, we have to combine two different paths from the two possible derivation trees in Fig. 8. □

Example 5.4. Consider attribute grammar *AG4* with $V_N = \{Z, A, B\}$, $V_T = \{a, b\}$, and $P = \{Z \rightarrow A, A \rightarrow AB, A \rightarrow a, B \rightarrow b\}$.

The possible sentences are ab^n $(n \geq 0)$. For sentence *abb* the derivation tree is given in Fig. 10. Figure 11 shows the associated *BD*-precedence graph.

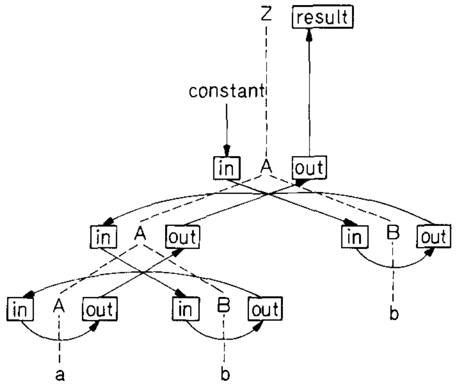

Fig. 10. Attributed derivation tree for sentence *abb* of grammar *AG4*

Notice that the number of pure left-to-right passes necessary to evaluate all the attribute instances depends on the depth of the recursion. So there is no fixed upper bound to the number of left-to-right passes and hence the grammar is not pure *LR* multi-pass.

Of course the simple *LR* multi-pass strategy also fails. (Notice the existence of the $\bar{L}$-cycle $p[\text{in}(A), \text{in}(B), \text{out}(B), \text{in}(A)]$ in $P_{BD}(AG4)$). Observe that evaluation is possible in one right-to-left pass. Hence, the grammar is simple *BD* one-pass. □

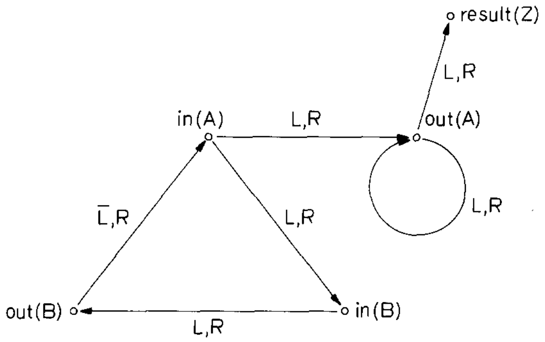

Fig. 11. $P_{BD}(AG4)$

Now we come back to the classification of pass-oriented evaluation methods as given at the end of Sect. 3 (see Fig. 1).

Grammar *AG4* is simple *BD* but not pure *LR*. Grammars *AG2* and *AG3* are pure *LR* but not simple *BD*. Hence the classes simple *BD* and pure *LR* are incomparable. From this follows that the inclusions depicted in Fig. 1 are proper.

6. Simple Left-to-right and/or Right-to-left Multi-Pass Evaluation

As shown in Theorem 4.1 knowledge of the relations $L, \bar{L}, R$ and $\bar{R}$ between the attributes of attribute grammar *AG* suffices to determine whether a given partial partition of the set of attributes of AG is BD-correct with respect to a given finite sequence of pass directions.

In this section we consider the case where a partitition is not given in advance but where we have to find a correct partition with respect to a given sequence of pass directions. Since in general also the number of passes to be made is unknown in advance, we need sequences of pass directions where the number of passes is unbounded. For such an infinite sequence we use the notation $\langle d_1, d_2, \ldots \rangle$.

We define a partition $\langle A_1, \ldots, A_m \rangle$ of the set of attributes of attribute grammar AG to be BD-correct with respect to the infinite sequence $\langle d_1, \ldots, d_m, \ldots \rangle$ if and only if it is BD-correct with respect to the finite sequence $\langle d_1, \ldots, d_m \rangle$ (and similarly for pass functions).

We will show that the precedence relations between attributes yield sufficient information to be able to decide whether an attribute grammar is simple multi-pass and if so, to compute the BD-correct complete partition with minimal pass numbers and, if not, to compute the BD-correct partial partition with minimal pass numbers, with respect to a given infinite sequence of pass directions.

In this section we consider simple multi-pass evaluation in general, where passes are allowed in both directions and in any order. In Sect. 7 we will consider the special case where only left-to-right passes are made and in Sect. 8 the case where left-to-right and right-to-left passes strictly alternate.

In order to find a correct partition of the attributes of an attribute grammar with respect to a given infinite sequence of pass directions we consider the labeling of the paths in the associated BD-precedence graph. Since it is our intention to evaluate the instances of each attribute at the earliest possible pass, we have to evaluate during a left-to-right pass as many attribute instances as possible. For a further explanation of that point we define a longest possible L-subpath of a path as follows.

Definition 6.1. Given a path $p[a_1, a_2, \ldots, a_n]$ where $n \geqq 1$. A subpath $p[a_r, a_{r+1}, \ldots, a_s]$ for $1 \leqq r \leqq s \leqq n$ is the *longest possible L-subpath* starting from a_r if all arcs (a_i, a_{i+1}) for $r \leqq i \leqq s-1$ are labeled L and
either $s = n$
or $s < n$ and $\mathrm{arc}(a_s, a_{s+1})$ is labeled $\bar{L}$. □

The *longest possible R-subpath* starting from a_r is defined in a similar manner (replace L by R everywhere in Definition 6.1).

Example 6.1. Consider in the BD-precedence graph of grammar $AG0$ in Fig. 3 path p[in(B), out(B), in(A), out(A), result(Z)]. The subpath p[in(B), out(B)] is the longest possible L-subpath starting from in(B). The subpath p[in(A), out(A), result(Z)] is the longest possible R-subpath starting from in(A). □

We now decompose any given path in a unique way into longest possible subpaths.

Definition 6.2. With respect to a given infinite sequence $\langle d_1, d_2, \ldots \rangle$ of pass directions, the *decomposition* of a path $p[a_1, a_2, \ldots, a_n]$ is the sequence of m longest possible L- and R-subpaths such that the i-th subpath is a longest possible d_i-subpath ($1 \leqq i \leqq m$), the first subpath starts with a_1, the last subpath ends with a_n and if $p[a_q, \ldots, a_r]$ is the i-th subpath and $p[a_s, \ldots, a_t]$ is the $(i+1)$th subpath then $s = r + 1$.

The arc between two successive longest possible subpaths of a decomposition will be called a *barrier*. □

Notice that the barrier after a longest possible d_i-subpath of a decomposition is labeled $\bar{d}_i$ and that the number of barriers of a decomposition is equal to the number of longest possible subpaths of the decomposition minus 1.

Example 6.2. Consider again in Fig. 3 the path $p[\text{in}(B), \text{out}(B), \text{in}(A), \text{out}(A), \text{result}(Z)]$. The sequence $p[\text{in}(B), \text{out}(B)]$, $p[\text{in}(A), \text{out}(A), \text{result}(Z)]$ is the decomposition with respect to the sequence $\langle L, R, \ldots \rangle$ of pass directions. The arc $(\text{out}(B), \text{in}(A))$ with label $\bar{L}$ is the barrier between the two subpaths of the decomposition. □

Lemma 6.1. Let 'pass' be a *BD*-correct pass function with respect to a sequence $\langle d_1, \ldots, d_l, d_{l+1}, \ldots \rangle$ of pass directions for an attribute grammar *AG*. If a path exists in $P_{BD}(AG)$ from attribute a to attribute b, $\text{pass}(a) = l$, and k is the number of barriers of the decomposition of the path with respect to the sequence $\langle d_l, d_{l+1}, \ldots \rangle$, then $\text{pass}(b) \geqq \text{pass}(a) + k$.

Proof. We prove the lemma by induction on the length, i.e., the number of arcs of the path.

I. The lemma is correct for a path of length 0. The path of length 0 from a to b is $p[a]$, where $a = b$. The decomposition consists of $p[a]$ only and hence its number of barriers is 0. Clearly, $\text{pass}(a) = \text{pass}(b)$.

II. The induction hypothesis is: the lemma holds for all paths of length $\leqq n$. We have to prove that it holds for all paths of length $\leqq n+1$.

Consider an arbitrary path $p[a_0, a_1 \ldots, a_n, a_{n+1}]$, where $a_0 = a$ and $a_{n+1} = b$. Let the decomposition of $p[a_0, a_1, \ldots, a_n]$ with respect to the sequence $\langle d_l, d_{l+1}, \ldots \rangle$ of pass directions be $p[a_0, \ldots], \ldots, p[\ldots, a_n]$ and let k be the number of barriers of the decomposition.

We have to consider the cases $d_{l+k} = L$ and $d_{l+k} = R$. We only discuss the case $d_{l+k} = L$ (for $d_{l+k} = R$ the proof is similar). Note that $d_{l+k} = L$ means that $p[\ldots, a_n]$ is a longest possible L-subpath. Now we distinguish 2 cases:

1. $\text{arc}(a_n, a_{n+1})$ has label L.
2. $\text{arc}(a_n, a_{n+1})$ has label $\bar{L}$.

Case 1. The decomposition of $p[a_0, \ldots, a_n, a_{n+1}]$ with respect to $\langle d_l, d_{l+1}, \ldots \rangle$ is $p[a_0, \ldots], \ldots, p[\ldots, a_n, a_{n+1}]$ with k barriers.

From the induction hypothesis follows that $\text{pass}(a_n) \geqq \text{pass}(a_0) + k$. From Theorem 4.1(i) follows that $\text{pass}(a_{n+1}) \geqq \text{pass}(a_n)$. Hence $\text{pass}(a_{n+1}) \geqq \text{pass}(a_0) + k$.

Case 2. The decomposition of $p[a_0, \ldots, a_n, a_{n+1}]$ with respect to $\langle d_l, d_{l+1}, \ldots \rangle$ is $p[a_0, \ldots], \ldots, p[\ldots, a_n], p[a_{n+1}]$ with $k+1$ barriers.

From the induction hypothesis follows that $\text{pass}(a_n) \geqq \text{pass}(a_0) + k = l + k$. If $\text{pass}(a_n) = l + k$ (a left-to-right pass), then $\text{pass}(a_{n+1}) > \text{pass}(a_n)$ by Theorem 4.1(ii), and hence $\text{pass}(a_{n+1}) \geqq \text{pass}(a_0) + k + 1$.

If $\text{pass}(a_n) > l+k$, then $\text{pass}(a_n) > \text{pass}(a_0)+k$; hence $\text{pass}(a_{n+1}) \geqq \text{pass}(a_n) > \text{pass}(a_0)+k$ by Theorem 4.1 (i), and so $\text{pass}(a_{n+1}) \geqq \text{pass}(a_0)+k+1$. Hence, $\text{pass}(a_{n+1}) \geqq \text{pass}(a_0)+k+1$.

Thus we have proved the lemma. □

In the following we will use the result of Lemma 6.1 to compute, for a given infinite sequence of pass directions, the correct pass function with the smallest possible values.

We start from the BD-precedence graph of an attribute grammar AG and an infinite sequence $\langle d_1, \ldots, d_l, d_{l+1}, \ldots\rangle$ of pass directions. Consider a path from vertex a to vertex b such that the decomposition associated with the sequence $\langle d_l, d_{l+1}, \ldots\rangle$ has k barriers. Assume that all attributes associated with vertices not on the path have pass numbers smaller than l. From the starting point that during pass l at least the instances of attribute a are evaluated, we try successively to evaluate all instances of attributes associated with the vertices on the path. From Lemma 6.1 we know that, with respect to $\langle d_l, d_{l+1}, \ldots\rangle$, the number of passes "to follow" the path is at least k. So it is meaningful to define cost-functions of a path in the following way.

Definition 6.3. With respect to a sequence $\langle d_1, \ldots, d_l, d_{l+1}, \ldots\rangle$ of pass directions the *l-start cost* of a path in $P_{BD}(AG)$ is the number of barriers of the decomposition associated with $\langle d_l, d_{l+1}, \ldots\rangle$. □

Now we consider all possible paths from a to b. As above we leave out of consideration all attributes not on any path from a to b. From the starting point that during pass l at least the instances of attribute a are computed, we try successively to evaluate all the instances of the attributes associated with all vertices on all paths from a to b. The number of passes needed "to follow" all these paths (starting with pass l) is at least the maximal l-start cost over all paths from a to b. Therefore we define cost-functions for each ordered pair of attributes.

Definition 6.4. Let the BD-precedence graph of an attribute grammar and an infinite sequence of pass directions be given. For each pair of attributes a and b,

$$\text{cost}_l(a,b) = \begin{cases} \text{the maximal } l\text{-start cost over all paths from } a \text{ to } b: \\ \qquad \text{if a path from } a \text{ to } b \text{ exists.} \\ -\infty: \text{ if no path from } a \text{ to } b \text{ exists.} \end{cases}$$ □

Notice that $\text{cost}_l(a,b) \in \{-\infty, 0, 1, 2, \ldots, +\infty\}$.

To get more insight in the properties of the cost-functions we consider the cases $\text{cost}_l(a,b) = +\infty$ for all possible infinite sequences of pass directions with an infinite number of passes in both directions.

Lemma 6.2. Let a and b be attributes of an attribute grammar AG. For every number l and for all possible infinite sequences of pass directions where both the number of L's and the number of R's is infinite, the following holds: $\text{cost}_l(a,b) = +\infty$ if and only if a path from a to b exists that includes and $\bar{L}-\bar{R}$-cycle.

Proof. ($\Rightarrow$) Given $\mathrm{cost}_l(a,b)=+\infty$ for some l and for some sequence $\langle d_l,\dots\rangle$ of pass directions with an infinite number of both L's and R's. This implies that, with respect to $\langle d_l,\dots\rangle$, for each finite k, there is a path from a to b such that the l-start cost is greater than k. Let n be the number of attributes of the grammar.

Since k is unbounded, there is a path from a to b such that it is possible to select from the decomposition of the path $2m$ times ($m \geqq n+1$) alternately a longest possible L-subpath and a longest possible R-subpath in the following way. The selected subpaths are respectively $p[a_{q_1},\dots,a_{r_1}]$, $p[a_{s_1},\dots,a_{t_1}],\dots,p[a_{q_i},\dots,a_{r_i}]$, $p[a_{s_i},\dots,a_{t_i}],\dots,p[a_{q_m},\dots,a_{r_m}]$, $p[a_{s_m},\dots,a_{t_m}]$, where $m\geqq n+1$, and where $r_i \geqq q_i$, $t_i \geqq s_i$ and $s_i > r_i$ for $1\leqq i\leqq m$, and $q_{i+1}>t_i$ for $1\leqq i\leqq m-1$. The paths $p[a_{q_i},\dots,a_{r_i}]$ for $1\leqq i\leqq m$ are longest possible L-subpaths and the paths $p[a_{s_i},\dots,a_{t_i}]$ for $1\leqq i\leqq m$ are longest possible R-subpaths. The arcs (a_{r_i}, a_{r_i+1}) *for* $1\leqq i\leqq m$ are labeled $\bar{L}$ and the arcs (a_{t_i}, a_{t_i+1}) for $1\leqq i\leqq m$ are labeled $\bar{R}$. Since the number n of different attributes is less than m, at least two of the attributes $a_{q_1}, a_{q_2},\dots,a_{q_m}$ must be equal. Hence, the subpath from a_{q_1} to a_{q_m} includes an $\bar{L}-\bar{R}$-cycle and hence there is a path from a to b that includes an $\bar{L}-\bar{R}$-cycle.

($\Leftarrow$) Given a path from a to b that includes an $\bar{L}-\bar{R}$-cycle. There is no finite upper bound to the number of repetitions of the $\bar{L}-\bar{R}$-cycle over all paths from a to b that include the $\bar{L}-\bar{R}$-cycle. Hence, with respect to each sequence $\langle d_l,\dots\rangle$ of pass directions there is no finite upper bound to the number of barriers over the decompositions of all paths from a to b that include the $\bar{L}-\bar{R}$-cycle. Hence $\mathrm{cost}_l(a,b)=+\infty$ for each l and for each sequence of pass directions. □

Corollary 6.1. Let a be an attribute of attribute grammar AG. For every number l and for all possible infinite sequences of pass directions where both the number of L's and the number of R's is infinite, the following holds: $\mathrm{cost}_l(a,a)=+\infty$ if and only if attribute a is involved in an $\bar{L}-\bar{R}$-cycle.

Proof. Follows immediately from Lemma 6.2. □

In order to know the number of passes needed at least before the instances of an attribute can be computed, we have to "follow" all possible paths to the node associated with that attribute. Hence we define a cost-function for each attribute that considers all possible paths to a, starting the evaluation process with the first pass.

Definition 6.5. Given the BD-precedence graph of an attribute grammar. With respect to a given infinite sequence of pass directions, for each attribute a, $\mathrm{COST}(a)$ is the maximal 1-start cost over all paths leading to a. Hence, $\mathrm{COST}(a)=\max\limits_{b}\mathrm{cost}_1(b,a)$ for b an attribute. □

From Lemma 6.1 and Definitions 6.3, 6.4 and 6.5 we immediately conclude the following lemma.

Lemma 6.3. Let pass be a BD-correct pass function with respect to an infinite sequence of pass directions for an attribute grammar AG. Then for each a, such that pass(a) is finite, $\mathrm{pass}(a)\geqq\mathrm{COST}(a)+1$. □

Now, with respect to any infinite sequence of pass directions, we consider the function $\text{pass}(a)=\text{COST}(a)+1$, and we prove that this function is a correct pass function.

Theorem 6.1. Let 'COST' be the COST-function associated with an infinite sequence $\langle d_1, d_2, \ldots\rangle$ of pass directions for an attribute grammar *AG*. Let $\langle A_1, A_2, \ldots, A_m\rangle$ be the partial partition of the set *A* of attributes of *AG*, such that $A_i=\{a \mid \text{COST}(a)=i-1\}$ for $1 \leqq i \leqq m$ and $a \in \bigcup_{i=1}^{m} A_i$ for each attribute *a* for which $\text{COST}(a)$ is finite.

(a) This partition is *BD*-correct with respect to $\langle d_1, d_2, \ldots, d_m, \ldots\rangle$.

(b) The pass function associated with this partition is, with respect to $\langle d_1, d_2, \ldots, d_m, \ldots\rangle$, the *BD*-correct pass function with the smallest possible values.

Proof. (a) From the definition of the COST-function it easily follows that the pass function associated with the partition has the property that, with respect to the given sequence of pass directions, for attributes *a* and *b*:

(i) if *a* prec *b* then $\text{pass}(a) \leqq \text{pass}(b)$

(ii) if $a\bar{L}b$ and $\text{pass}(a)$ or $\text{pass}(b)$ is finite and a left-to-right pass then $\text{pass}(a)<\text{pass}(b)$.

(iii) if $a\bar{R}b$ and $\text{pass}(a)$ or $\text{pass}(b)$ is finite and a right-to-left pass then $\text{pass}(a)<\text{pass}(b)$.

From Theorem 4.1 it follows that partition $\langle A_1, A_2, \ldots, A_m\rangle$ is *BD*-correct with respect to $\langle d_1, d_2, \ldots, d_m\rangle$. This implies that partition $\langle A_1, A_2, \ldots, A_m\rangle$ is also *BD*-correct with respect to all infinite sequences of pass directions where the first *m* pass directions are $d_1, d_2, \ldots, d_m$.

(b) From Lemma 6.3 follows that the function $\text{pass}(a)=\text{COST}(a)+1$ is, with respect to the given sequence of pass directions, the correct pass function with the smallest possible values. □

Given an attribute grammar, its *BD*-precedence graph and the COST-function 'COST' with respect to an infinite sequence of pass directions, we consider two cases:

1. $\text{COST}(a)$ is finite for each attribute *a*. From Theorem 6.1 it follows that, with respect to the sequence of pass directions, at least one correct complete partition exists and hence, at least one correct complete pass function exists.

2. $\text{COST}(a)=+\infty$ for at least one attribute *a*. Lemma 6.3 states: if, with respect to the given sequence of pass directions, a correct partition exists with a corresponding pass function 'pass', then $\text{pass}(a) \geqq \text{COST}(a)+1$ for each attribute *a* such that $\text{pass}(a)$ is finite. Hence, if an attribute *a* exists such that $\text{COST}(a)=+\infty$ then, with respect to the given sequence of pass directions, no complete correct pass function exists.

From these two observations we immediately conclude the following theorem.

Theorem 6.2. With respect to a given infinite sequence of pass directions of an attribute grammar, a correct complete partition of the set *A* of attributes exists if and only if for each attribute *a* $\text{COST}(a)$ is finite. □

Corollary 4.1 already indicated that an attribute grammar with an $\bar{L}-\bar{R}$-cycle in its BD-precedence graph cannot be simple BD multi-pass. Now we can easily prove the 'if and only if' of the following theorem.

Theorem 6.3. An attribute grammar is simple BD-multi-pass if and only if its BD-precedence graph has no $\bar{L}-\bar{R}$-cycles.

Proof. By Definition 3.7 an attribute grammar AG is simple BD multi-pass if and only if with respect to some finite sequence $\langle d_1,\ldots,d_m\rangle$ of pass directions a BD-correct complete partition $\langle A_1,\ldots,A_m\rangle$ of the set A of attributes exists. An equivalent definition is that AG is simple BD multi-pass if and only if with respect to some infinite sequence $\langle d_1,\ldots,d_m,\ldots\rangle$ of pass directions, with an infinite number of L's and R's, a BD-correct complete partition $\langle A_1,\ldots,A_m\rangle$ of A exists.

Theorem 6.2 states that, with respect to such an infinite sequence, a correct partition exists if and only if COST(a) is finite for each attribute a. From Definition 6.5 it follows that, with respect to such a sequence, COST$(a)=+\infty$ if and only if an attribute b exists such that $\text{cost}_1(b,a)=+\infty$. Finally, Lemma 6.2 states that, given such a sequence of pass directions, $\text{cost}_1(b,a)=+\infty$ if and only if a path from b to a exists that includes an $\bar{L}-\bar{R}$-cycle. □

Example 6.3. Consider the BD-precedence graph $P_{BD}(AG1)$ in Fig. 5. With respect to sequence $\langle R,R,\ldots\rangle$ of pass directions the values of the cost_1-function are:

cost_1	in(E)	out(E)	result(Z)
in(E)	0	0	0
out(E)	0	0	0
result(Z)	$-\infty$	$-\infty$	0

For each attribute the value of the COST-function is 0. Hence, simple multi-pass evaluation is possible in one right-to-left pass. □

To conclude this section we discuss an algorithm to calculate, for a given infinite sequence of pass directions, pass numbers according to the function pass$(a)=$COST$(a)+1$ for all attributes of an attribute grammar. In Sects. 7 and 8 where we discuss special sequences of pass directions, we will come back to this algorithm and also discuss algorithms that provide more information. All these algorithms can be viewed as "path finding" algorithms in the BD- or LR-precedence graph of the grammar.

Algorithm 6.1 is essentially the algorithm given by Bochmann in [3] and Jazayeri and Walter in [8]. In this paper it is presented in terms of the BD-precedence graph of the grammar.

The method to calculate the values of the COST-function with respect to a given infinite sequence $\langle d_1,d_2,\ldots\rangle$ of pass directions is as follows. We define

$$S_i=\{a\,|\,\text{COST}(a)=i\}\quad\text{for finite } i.$$
$$S_\infty=\{a\,|\,\text{COST}(a)=\infty\}.$$

For finite m subset S_m will be computed after the sets $S_0, \ldots, S_{m-1}$ have been determined completely. Notice that for finite m

$$S_m = \{a \mid \text{for each path } p[a_1, \ldots, a_k], \text{ where } a_k = a \text{ and } a_j \notin \bigcup_{i=1}^{m-1} S_i \text{ for } 1 \leqq j \leqq k, \text{ all arcs have label } d_{m+1}\}.$$

The computation of S_m proceeds as follows. All attributes in $\bigcup_{i=0}^{m-1} S_i$ are considered to be marked. Initially it is assumed that all unmarked attributes belong to S_m. Non-members of S_m, i.e., attributes a such that $\mathrm{COST}(a) > m$, will be successively deleted. To that end all attributes a are deleted for which an unmarked attribute b exists such that arc (b, a) is labeled $\bar{d}_{m+1}$ (i.e., if $d_{m+1} = L$ then $\bar{L}$ else $\bar{R}$); moreover all those attributes are deleted that depend (indirectly) on such attributes a. The deletion process continues until no more deletions are possible. In this manner all subsets S_i for finite i are computed.

The process terminates successfully when all attributes are distributed over some set S_i for finite i. In this case, with respect to the given sequence of pass directions, a correct complete partition of the set A of attributes is found.

The process terminates unsuccessfully when it becomes clear that all future passes (according to the given sequence of pass directions) will deliver empty subsets, while the set of unmarked attributes is still not empty. This is the case when for some $m > 1$, $S_m = \emptyset$ and $S_{m-1} = \emptyset$ while the associated pass directions d_{m+1} and d_m are different or when for some $m > 0$, $S_m = \emptyset$ and the associated pass direction d_{m+1} is equal to all the following pass directions. The set of remaining attributes will be denoted by S_∞. In this case, with respect to the given sequence of pass directions, only a partial BD-correct partition of the set A of attributes is found.

Algorithm 6.1 {see [3] and [8]}. Computation of the smallest pass function for simple BD-multi-pass evaluation with respect to a given infinite sequence of pass directions.

Input: attribute grammar AG; sequence $\langle d_1, d_2, \ldots \rangle$ of pass directions.
Output: with respect to $\langle d_1, d_2, \ldots \rangle$ the BD-correct pass function (complete if possible) with the smallest possible function-values.

Algorithm:
```
begin
    construct the graph $P_{BD}(AG)$.
    set COST($a$) to undefined for all vertices $a$;
    $m := -1$;
    repeat
        $m := m + 1$;
        set COST($a$) to $m$ for all vertices for which COST($a$) is undefined;
        repeat
            for a vertex a such that COST($a$) $= m$
            set COST($a$) to undefined
            if there exists a vertex $b$ and an arc $(b, a)$ such that
                (COST($b$) $= m$ and arc($b, a$) has label $\bar{d}_{m+1}$
                or
                COST($b$) is undefined)
        until no vertex $a$ can be found such that COST($a$) can be set to undefined
    until {termination condition}
        either for all $a$ COST($a$) is defined
        or     (($d_{m+1} \neq d_m$ and
                 no vertices $b$ and $c$ exist such that COST($b$)$=m$ and COST($c$)$=m-1$)
                 or
                 ($d_{m+1}=d_{m+2}=d_{m+3}=\ldots$ and
                 no vertex $b$ exists such that COST($b$)$=m$)
               );
    for all $a \in A$
    do if COST($a$) is defined then pass($a$):=COST($a$)+1
                               else pass($a$):= $+\infty$
        fi
    od
end □
```

Let n be the number of attributes of a grammar. If we count the number of times the label of an arc is examined, then Algorithm 6.1 takes time $O(m_f\, n^2)$, where m_f is the number of times the outer repeat statement is executed.

Remark 1. Obviously a direct computation of the values of the pass function is possible without computing the values of the COST-function beforehand. In that case for m the initial value 0 should be used and d_i should be replaced by d_{i-1} for $i \geqq m$.

Remark 2. For a finite sequence $\langle d_1, \ldots, d_{m_r} \rangle$ of pass directions the termination condition of Algorithm 6.1 has to be changed into

either for all a COST(a) is defined

or $\quad m+1=m_r$.

Note that this version of Algorithm 6.1 takes time $O(m_r\, n^2)$.

In Sect. 7 and 8 we will discuss other algorithms for the cases where only left-to-right passes are made or where left-to-right and right-to-left passes strictly alternate.

7. Simple Left-to-right Multi-Pass Evaluation

As a special case of simple multi-pass evaluation we discuss the strategy where only left-to-right passes are executed. Since for this problem we only need the labels L and $\bar{L}$ of the precedence graph, the LR-graph $P_{LR}(AG)$ will be used.

We start our investigation with two observations concerning the cost-functions.

1. The number of barriers of the decomposition of a path with respect to $\langle L, L, \ldots \rangle$ is equal to the number of arcs labeled $\bar{L}$ on the path.

2. The l-start cost of a path, i.e., the number of arcs labeled $\bar{L}$ on the path, is independent of l. So for each path one cost-function suffices. These observations lead to the following particular cases of Definitions 6.3, 6.4 and 6.5.

Definition 7.1. The *cost* of a path is the number of arcs labeled $\bar{L}$ on the path. □

Definition 7.2. For each pair of attributes a and b,

$$\mathrm{cost}(a,b)=\begin{cases}\text{the maximal cost over all paths from } a \text{ to } b\text{:} \\ \qquad \text{if a path from } a \text{ to } b \text{ exists.} \\ -\infty\text{: if no path from } a \text{ to } b \text{ exists.}\end{cases} \quad \square$$

Definition 7.3. For each attribute a, $\mathrm{COST}(a)=\max_b \mathrm{cost}(b,a)$ for b an attribute. $\square$

Observe that Theorems 6.1 and 6.2 hold for any infinite sequence of pass directions and hence in particular for the sequence $\langle L, L, \ldots\rangle$.

Now, as for *BD*-multi-pass evaluation, also for *LR* multi-pass evaluation we consider the cases where $\mathrm{cost}(a,b)=+\infty$ and $\mathrm{COST}(a)=+\infty$.

Notice that since Lemma 6.2 is restricted to sequences of pass directions with an infinite number of both L's and R's the next lemma cannot be considered as a special case of Lemma 6.2 for the sequence $\langle L, L, \ldots\rangle$. Also instead of Theorem 6.3 a new theorem has to be formulated for the *LR*-case.

Lemma 7.1. For each pair of attributes a and b, $\mathrm{cost}(a,b)=+\infty$ if and only if a path from a to b exists that includes an $\bar{L}$-cycle. $\square$

The proof of this lemma proceeds along the same lines as the proof of Lemma 6.2 and is omitted. From Lemma 7.1 immediately follows the next corollary.

Corollary 7.1. For each attribute a, $\mathrm{cost}(a,a)=+\infty$ if and only if attribute a is involved in an $\bar{L}$-cycle. $\square$

Now, we find parallel to Theorem 6.3.

Theorem 7.1. An attribute grammar is simple *LR* multi-pass if and only if its *LR*-precedence graph has no $\bar{L}$-cycles.

Proof. As the proof of Theorem 6.3. $\square$

At the end of this section we present a new algorithm to calculate pass numbers, where we use the following lemma on circular paths.

Lemma 7.2. For each attribute a

(a) $\mathrm{cost}(a,a)$ is either 0 or $+\infty$

(b) $\mathrm{cost}(a,a)=0$ iff all arcs of all paths from a to a are labeled L.

(c) $\mathrm{cost}(a,a)=+\infty$ iff at least one arc of a path from a to a is labeled $\bar{L}$.

Proof. Follows immediately from Definition 7.2 and Corollary 7.1. $\square$

Example 7.1. Consider grammar *AG1* (see Example 5.1). $P_{LR}(AG1)$ is given in Fig. 5. The values of the cost-function are:

cost	in(E)	out(E)	result(Z)
in(E)	$+\infty$	$+\infty$	$+\infty$
out(E)	$+\infty$	$+\infty$	$+\infty$
result(Z)	$-\infty$	$-\infty$	0

For each attribute the value of the COST-function is $+\infty$. The grammar is not simple LR multi-pass. From the values of the cost-function it follows (Corollary 7.1) that all attributes, except result(Z) are involved in an $\bar{L}$-cycle.

Notice that COST(result(Z)) $= +\infty$, but cost(result(Z), result(Z)) $= 0$. Hence (Lemma 6.3) it is impossible to assign a finite pass-number to result(Z), but (Corollary 7.1) result(Z) is not involved in an $\bar{L}$-cycle. □

We end this section with two algorithms to calculate, for the simple LR strategy, pass numbers according to the function pass$(a)=$COST$(a)+1$ for all attributes of an attribute grammar. Both algorithms can be viewed as "path finding" algorithms in the LR-precedence graph of the grammar.

Firstly we adapt Algorithm 6.1 for this particular case. The criterion for unsuccessful termination is concluded from the observation that all pass directions are equal. Hence, when the deletion process delivers an empty subset S_m while the set of undefined attributes is still not empty, all succeeding subsets will also be empty.

So, the termination condition of the adapted version of Algorithm 6.1 is:

either for all a COST(a) is defined

or no vertex b exists such that COST$(b)=m$.

Since m_f (the number of times the outer repeat statement of Algorithm 6.1 is executed) is at most n (the number of attributes of the grammar), the adapted version of the algorithm takes time $O(n^3)$ in the worst case. If the grammar is 1-pass then it takes $O(n^2)$ steps.

Notice that this adapted version of Algorithm 6.1 is essentially the algorithm given by Bochmann in [3].

Now we present Algorithm 7.1, a new algorithm to calculate pass numbers. It first computes cost(a,b) for each pair of attributes a and b and then, using Definition 7.3, it calculates the values of the COST-function and from this the pass numbers. If the grammar is not simple LR multi-pass the values of the cost-function can be used to indicate the attributes that are involved in an $\bar{L}$-cycle.

Algorithm 7.1 is a particular case of Algorithm 5.5 in [1, pp. 195–199]. The method is as follows.

Consider the directed graph $P_{LR}(AG)$ associated with attribute grammar AG and let the vertices, associated with attributes, be $a_1, a_2, \ldots, a_n$. We define C_{ij}^k $(1 \leqq i,j,k \leqq n)$ to be the maximum cost over all paths from a_i to a_j such that all the vertices on the path except possibly the endpoints are in the set $\{a_1, a_2, \ldots, a_k\}$. $C_{ij}^k = -\infty$ if no such path exists. Thus $C_{ij}^n = \text{cost}(a_i, a_j)$. To compute C_{ij}^k, by induction on k, we split up the paths from a_i to a_j with no intermediate vertices higher than k into two groups:

1) paths with no intermediate vertex higher than $k-1$; the maximum cost of those paths is C_{ij}^{k-1}.

2) paths composed of a sequence of subpaths: a subpath from a_i to a_k, then a sequence of zero or more subpaths from a_k to a_k and finally a subpath from a_k to a_j, where each subpath has no intermediate vertex higher than $k-1$. Such paths exist iff $C_{ik}^{k-1} \neq -\infty$ and $C_{kj}^{k-1} \neq -\infty$.

If such paths exist, then the maximum cost over all those paths (abbreviated by $C^k_{ij\,\mathrm{via}\,k}$) is: $C^{k-1}_{ik} + \mathrm{seq}\, C^{k-1}_{kk} + C^{k-1}_{kj}$, where seq C^{k-1}_{kk} is the maximum cost over any (possibly empty) sequence of subpaths from a_k to a_k, where each subpath has no intermediate vertex higher than $k-1$. Hence (Lemma 7.2), clearly

$$\mathrm{seq}\, C^{k-1}_{kk} = \begin{cases} 0: & \text{if } C^{k-1}_{kk} = 0 \\ +\infty: & \text{if } C^{k-1}_{kk} \geqq 1 \end{cases}$$

Thus C^k_{ij} can be computed as follows:

$$C^k_{ij} := \textbf{if } C^{k-1}_{ik} \geqq 0 \textbf{ and } C^{k-1}_{kj} \geqq 0 \textbf{ then } \max(C^{k-1}_{ij}, C^k_{ij\,\mathrm{via}\,k}) \textbf{ else } C^{k-1}_{ij} \textbf{ fi}.$$

To initialize the induction, note that for $i \neq j$

$$C^0_{ij} = \begin{cases} -\infty: & \text{if arc}(a_i, a_j) \text{ does not exist.} \\ 0\ : & \text{if arc}(a_i, a_j) \text{ exists and has label } L. \\ 1\ : & \text{if arc}(a_i, a_j) \text{ exists and has label } \bar{L}. \end{cases}$$

and

$$C^0_{ii} = \begin{cases} 0\ : & \text{if arc}(a_i, a_i) \text{ does not exist or} \\ & \text{exists and has label } L \\ 1\ : & \text{if arc}(a_i, a_i) \text{ exists and has label } \bar{L}. \end{cases}$$

Algorithm 7.1. Computation of the smallest pass function for simple *LR* multipass evaluation.

Input: attribute grammar *AG*.

Output: cost(a,b) for each pair of attributes a and b of *AG* and the *LR*-correct pass function (complete if possible) with the smallest possible function-values.

Algorithm:

```
begin
   var $C^{\mathrm{old}}$, $C^{\mathrm{new}}$: array [$1...n, 1...n$] of $(-\infty, 0, 1, 2, ..., +\infty)$;
   {$n$ is the number of attributes of nonterminals of $AG$}
   Construct the graph $P_{LR}(AG)$;
   for $1 \leqq i,j \leqq n$
   do
      $C^{\mathrm{old}}_{ij}$ := if arc$(a_i, a_j)$ does not exist
            then if $i \neq j$ then $-\infty$ else 0 fi
            else if arc$(a_i, a_j)$ has label $L$
                 then 0 else 1
                 fi
            fi
   od;
   for $k$ from 1 to $n$
   do
      if $C^{\mathrm{old}}_{kk} \geqq 1$
      then for $1 \leqq i,j \leqq n$
           do $C^{\mathrm{new}}_{ij}$ := if $C^{\mathrm{old}}_{ik} = -\infty$ or $C^{\mathrm{old}}_{kj} = -\infty$
                    then $C^{\mathrm{old}}_{ij}$ else $+\infty$
                    fi
           od
```

```
        else for $1 \leq i,j \leq n$
            do $C_{ij}^{\text{new}} := $ if $C_{ik}^{\text{old}} = -\infty$ or $C_{kj}^{\text{old}} = -\infty$
                    then $C_{ij}^{\text{old}}$ else $\max(C_{ij}^{\text{old}}, C_{ik}^{\text{old}} + C_{kj}^{\text{old}})$
                    fi
            od
        fi;
        $C^{\text{old}} := C^{\text{new}}$
    od;
    $\{\text{cost}(a_i, a_j) = C_{ij}^{\text{new}}\}$
    for $1 \leq i \leq n$
    do
        $\text{COST}(a_i) = \max_{1 \leq j \leq n} C_{ji}^{\text{new}}$;
        if $\text{COST}(a_i)$ is finite then $\text{pass}(a_i) = \text{COST}(a_i) + 1$
                                  else $\text{pass}(a_i) = +\infty$
        fi
    od
end
```

If we count the number of times C_{ij}^{old} is examined for any i, j then Algorithm 7.1 takes time $O(n^3)$, just as Algorithm 6.1. Notice however that Algorithm 7.1, since it not only computes the COST-function but also the cost-function, gives more information than Algorithm 6.1. The cost-function indicates the attributes that are involved in an $\bar{L}$-cycle.

Instances of attributes involved in an $\bar{L}$-cycle have to be evaluated by another strategy (for example the pure *LR* multi-pass strategy), but the instances of all other attributes can be computed by the simple *LR* multi-pass strategy as soon as the attributes involved in an $\bar{L}$-cycle on which they depend, have been computed.

Notice that Algorithm 7.1 can be used to develop such a mixture of evaluation strategies. Let A' be a subset of the set A of attributes of attribute grammar AG. Suppose we want to know whether AG is simple *LR* multi-pass under the assumption that the instances of all attributes of A' have already been computed. For each A' Algorithm 6.1 has, in order to compute the values of this new COST-function, to do its work over again completely. Algorithm 7.1 contains already to a large extent for all A' the necessary information in the cost-function. For all $a \notin A'$ the new COST-function is easily computed as follows: $\text{COST}(a) = \max_{b \notin A'} \text{cost}(b, a)$. Notice that the value of $\text{cost}(b, a)$ is contained in the matrix C^{new}.

Evaluation by mixing the simple *LR* multi-pass strategy with other (more powerful) strategies is illustrated in the following example.

Example 7.2. For grammar *AG5* the *LR* precedence graph $P_{LR}(AG5)$ is given in Fig. 12. Algorithm 7.1 finds the *LR*-correct not complete pass-function $\text{pass}(a) = 1$ and $\text{pass}(b) = 2$. It also indicates that attributes c and d are involved in an $\bar{L}$-cycle and that these attributes do not depend on other attributes, besides a and b. So after two simple *LR* passes for the evaluation of instances of a and b a more powerful strategy has to be applied in order to evaluate the instances of c and d. Now, under the assumption that the instances of all attributes of $A' = \{a, b, c, d\}$ have already been computed, the result of Algorithm 7.1 is applied again to recompute the COST-function. We then find the *LR*-correct complete pass function $\text{pass}(e) = 1$ and $\text{pass}(f) = 2$. □

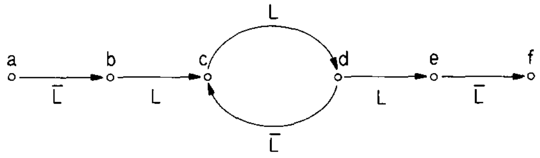

Fig. 12. $P_{LR}(AG5)$

8. Simple Alternating Multi-Pass Evaluation

By simple alternating multi-pass evaluation we mean simple BD multi-pass evaluation where left-to-right and right-to-left passes strictly alternate.

Definition 8.1. An attribute grammar is *simple ALT m-pass* if with respect to the sequence $\langle L, R, L, R, \ldots \rangle$ of pass directions a BD-correct complete partition of the set A of attributes exists.

An attribute grammar *simple ALT multi-pass* if it is simple ALT m-pass for some m. □

With respect to simple alternating multi-pass evaluation we make the following observations.

1. By definition, if an attribute grammar is simple ALT m-pass, then it is simple BD m-pass.

2. Let AG be simple BD m-pass with respect to the sequence $\langle d_1, \ldots, d_i, d_{i+1}, \ldots, d_m \rangle$ of pass directions. This means that with respect to this sequence a correct complete partition $\langle A_1, \ldots, A_i, A_{i+1}, \ldots, A_m \rangle$ of the set A of attributes exists. Let $d_i = d_{i+1} = L$. Notice that the partition $\langle A_1, \ldots, A_i, \emptyset, A_{i+1}, \ldots, A_m \rangle$ is correct and complete with respect to $\langle d_1, \ldots, d_i, R, d_{i+1}, \ldots, d_m \rangle$. Hence, clearly a grammar which is BD m-pass is simple ALT n-pass for some $n \leqq 2m-1$. From these two observations we immediately conclude the following theorem.

Theorem 8.1. An attribute grammar is simple BD multi-pass if and only if it is simple ALT multi-pass. □

From the strict alternation of the pass directions it follows that for each path only two different decompositions have to be distinguished. The decomposition where the first subpath is a longest possible L-subpath will be called the *L-start decomposition*, and the decomposition where the first subpath is a longest possible R-subpath will be called the *R-start decomposition* of the path.

In both cases the number of barriers of the decomposition is the number of alternations of longest possible L- and R-subpaths.

These considerations lead to the following particular cases of Definitions 6.3, 6.4 and 6.5.

Definition 8.2. The *L-start cost* of a path is the number of barriers of the L-start decomposition of the path.

The *R-start cost* of a path is the number of barriers of the R-start decomposition of the path. □

Definition 8.3. For each pair of attributes a and b,

$$\text{cost}_L(a,b)=\begin{cases}\text{the maximal } L\text{-start cost over all paths from } a \text{ to } b\text{:} \\ \qquad \text{if a path from } a \text{ to } b \text{ exists.} \\ -\infty\text{: if no path from } a \text{ to } b \text{ exists.}\end{cases}$$

$$\text{cost}_R(a,b)=\begin{cases}\text{the maximal } R\text{-start cost over all paths from } a \text{ to } b\text{:} \\ \qquad \text{if a path from } a \text{ to } b \text{ exists.} \\ -\infty\text{: if no paths from } a \text{ to } b \text{ exists.}\end{cases} \quad \square$$

Notice that for each path,

$$\text{the } l\text{-start cost}=\begin{cases}\text{the } L\text{-start cost: if } l \text{ is odd} \\ \text{the } R\text{-start cost: if } l \text{ is even.}\end{cases}$$

Also for each pair of attributes a and b,

$$\text{cost}_l(a,b)=\begin{cases}\text{cost}_L(a,b)\text{: if } l \text{ is odd.} \\ \text{cost}_R(a,b)\text{: if } l \text{ is even.}\end{cases}$$

So, we can use the following definition as a particular case of Definition 6.5.

Definition 8.4. For each attribute a, $\text{COST}(a)=\max_b \text{cost}_L(b,a)$ for b an attribute. $\square$

Since Theorems 6.1 and 6.2 hold for any infinite sequence of pass directions, they also hold for the particular sequence $\langle L, R, L, R, \ldots\rangle$. Combining Theorems 8.1 and 6.3 we find that an attribute grammar is simple ALT multi-pass if and only if its BD-precedence graph has no $\bar{L}-\bar{R}$-cycles.

At the end of this section we sketch a new algorithm to compute minimal pass numbers for simple alternating multi-pass evaluation. For the explanation of that algorithm we need the following lemma on circular paths.

Lemma 8.1. For each attribute a, the pair $[\text{cost}_L(a,a), \text{cost}_R(a,a)]=$

either	$[0,0]$	iff all arcs of all paths from a to a are labeled L and R.
or	$[0,1]$	iff all arcs of all paths from a to a are labeled L and at least one arc of one path from a to a is labeled $\bar{R}$.
or	$[1,0]$	iff all arcs of all paths from a to a are labeled R and at least one arc of one path from a to a is labeled $\bar{L}$.
or	$[+\infty, +\infty]$	iff at least one arc of one path from a to a is labeled $\bar{L}$ and at least one arc of one path from a to a is labeled $\bar{R}$.

Proof. Follows immediately from Definition 8.3 and Corollary 6.1. $\square$

Parallel to the algorithms to calculate minimal pass numbers for simple left-to-right multi-pass evaluation we discuss two algorithms to calculate minimal pass numbers for simple multi-pass evaluation where left-to-right and right-to-left passes strictly alternate.

Firstly we indicate how to adapt Algorithm 6.1 for this particular case. The criterion for unsuccessful termination is concluded from the observation that successive passes have opposite directions. Hence, when the deletion process delivers two empty subsets S_{m-1} and S_m, while the set of remaining attributes is still not empty, all succeeding subsets will also be empty.

Hence the termination condition becomes:

either for all a COST(a) is defined
or no vertices b and c exist such that COST$(b)=m$ and COST$(c)=m-1$.

Notice that this version of Algorithm 6.1 is essentially the algorithm of Jazayeri and Walter in [8]. It takes time $O(m_f n^2)=O(n^3)$.

Now we sketch, parallel to Algorithm 7.1, a new algorithm to calculate minimal pass numbers for simple alternating multi-pass evaluation. It first computes $\text{cost}_L(a,b)$ and $\text{cost}_R(a,b)$ for each pair of attributes a and b and then, using Definition 8.4, it calculates the values of the COST-function and from this the pass function.

If the grammar is not simple ALT multi-pass the values of the $\text{cost}_{\bar{L}}$-function can be used to indicate the attributes that are involved in an $\bar{L}-\bar{R}$-cycle (Lemma 8.1).

The method is in broad lines the same as for simple LR multi-pass evaluation.

Consider the directed graph $P_{BD}(AG)$ associated with attribute grammar AG and let the vertices, associated with attributes, be $a_1, a_2, \ldots, a_n$. We define LC^k_{ij} and RC^k_{ij} $(1 \leqq i, j, k \leqq n)$ respectively to be the maximum L-start cost and R-start cost over all paths from a_i to a_j such that all vertices on the path except possibly the endpoints are in the set $\{a_1, a_2, \ldots, a_k\}$. $LC^k_{ij}=RC^k_{ij}=-\infty$ if no such path exists. Thus $LC^n_{ij}=\text{cost}_L(a_i, a_j)$ and $RC^n_{ij}=\text{cost}_R(a_i, a_j)$.

The computation of LC^k_{ij} and RC^k_{ij} (by induction on k) proceeds analogously to the computation of C^k_{ij} for the LR-case. The paths from a_i to a_j with no intermediate vertex higher than k are split up into two groups:

1) paths with no intermediate vertex higher than $k-1$.

2) paths composed of a sequence of subpaths: a subpath from a_i to a_k, then a sequence of subpaths from a_k to a_k and finally a subpath from a_k to a_j, where each subpath has no intermediate vertex higher than $k-1$.

If such paths exist, then the maximum L-start cost over all those paths (abbreviated by $LC^k_{ij\,\text{via}\,k}$) can be computed as follows.

For reasons of explanation we introduce $\text{seq}\,LC^{k-1}_{kk}$ and $\text{seq}\,RC^{k-1}_{kk}$ respectively defined as the maximum L-start cost and the maximum R-start cost over any (possibly empty) sequence of subpaths from a_k to a_k, where each subpath has no intermediate vertex higher than $k-1$.

From Lemma 8.1 it follows that

$$\text{seq}\,LC^{k-1}_{kk}=\begin{cases}0 & : \text{ if } LC^{k-1}_{kk}=0\\ 1 & : \text{ if } LC^{k-1}_{kk}\geqq 1 \text{ and } RC^{k-1}_{kk}=0\\ +\infty & : \text{ if } LC^{k-1}_{kk}\geqq 1 \text{ and } RC^{k-1}_{kk}\geqq 1,\end{cases}$$

$$\text{seq}\,RC^{k-1}_{kk}=\begin{cases}0 & : \text{ if } RC^{k-1}_{kk}=0\\ 1 & : \text{ if } RC^{k-1}_{kk}\geqq 1 \text{ and } LC^{k-1}_{kk}=0\\ +\infty & : \text{ if } RC^{k-1}_{kk}\geqq 1 \text{ and } LC^{k-1}_{kk}\geqq 1.\end{cases}$$

Now, the computation of $LC^k_{ij\,\mathrm{via}\,k}$ can be expressed as follows.

```
$LC^k_{ij\,\mathrm{via}\,k}$:= if $LC^{k-1}_{ik} = +\infty$ or seq $LC^{k-1}_{kk} = +\infty$
              {seq $LC^{k-1}_{kk} = +\infty$ means that also seq $RC^{k-1}_{kk} = +\infty$}
         then $+\infty$
         else $LC^{k-1}_{ik}$ + if even($LC^{k-1}_{ik}$)
                   then if seq $LC^{k-1}_{kk} = 0$
                          then $LC^{k-1}_{kj}$ else $1 + RC^{k-1}_{kj}$
                          fi
                   else if seq $RC^{k-1}_{kk} = 0$
                          then $RC^{k-1}_{kj}$ else $1 + LC^{k-1}_{kj}$
                          fi
                   fi
         fi
```

$RC^k_{ij\,\mathrm{via}\,k}$ is computed in a similar manner.

Thus LC^k_{ij} and RC^k_{ij} can be computed as follows.

```
$LC^k_{ij}$:= if $LC^{k-1}_{ik} \geqq 0$ and $LC^{k-1}_{kj} \geqq 0$ then max($LC^{k-1}_{ij}$, $LC^k_{ij\,\mathrm{via}\,k}$)
                                          else $LC^{k-1}_{ij}$ fi,
$RC^k_{ij}$:= if $RC^{k-1}_{ik} \geqq 0$ and $RC^{k-1}_{kj} \geqq 0$ then max($RC^{k-1}_{ij}$, $RC^k_{ij\,\mathrm{via}\,k}$)
                                          else $RC^{k-1}_{ij}$ fi.
```

To initialize the induction, note that for $i \neq j$

$$LC^0_{ij} = \begin{cases} -\infty: & \text{if arc } (a_i, a_j) \text{ does not exist.} \\ 0 \quad : & \text{if arc } (a_i, a_j) \text{ exists and has label } L. \\ 1 \quad : & \text{if arc } (a_i, a_j) \text{ exists and has label } \bar{L}. \end{cases}$$

and

$$LC^0_{ii} = \begin{cases} 0: & \text{if arc } (a_i, a_j) \text{ does not exist or exists and has label } L. \\ 1: & \text{if arc } (a_i, a_i) \text{ exists and has label } \bar{L}. \end{cases}$$

For RC^0_{ij} $(i \neq j)$ and RC^0_{ii} similar definitions can be given.

A complete description of the algorithm can be found in [2].

Notice that the algorithm sketched above takes the same time as the algorithm of Jazayeri and Walter in [8], but gives more information since it computes not only the COST-function but also the cost_L-function. The latter indicates the attributes that are involved in an $\bar{L} - \bar{R}$-cycle.

From Theorem 8.1 we know that this algorithm indicates whether an attribute grammar is simple *BD* multi-pass. However, since the algorithm requires strict alternation, in general not the sequence of pass directions with the minimal number of passes is found.

An algorithm that finds an optimal evaluation strategy in most practical cases, is presented in [12].

Observe that the algorithm sketched can be used to develop a mixture of evaluation strategies in a similar manner as has been described for left-to-right multi-pass evaluation at the end of Sect. 7.

9. Conclusions and Further Research

We discussed attribute evaluation in passes, made a clear distinction between pure and simple multi-pass evaluation and especially investigated simple multi-pass evaluation strategies.

We gave a graph theoretic characterization showing in which cases an attribute grammar meets the simple multi-pass requirements and developed, for particular sequences of pass directions, algorithms that associate minimal pass numbers with attributes and in case of failure indicate the attributes that cause the rejection of the grammar.

To characterize grammars we proved that an attribute grammar is simple multi-pass if and only if none of its attributes are involved in a cycle whose labels are not consistent with one of the possible pass directions. This criterion was independently found by Räihä and Ukkonen [12]. They were mainly interested in finding an optimal evaluation strategy where it is not dictated that left-to-right and right-to-left passes strictly alternate.

In this paper we also discussed the mixing of the simple multi-pass strategy with other evaluation strategies in case the grammar is not completely simple multi-pass.

Another approach is to transform a grammar that is pure multi-pass but not simple multi-pass into a grammar that is simple multi-pass. In [4] it is proved that such a transformation is always possible, but because of the increased number of attributes of the resulting attribute grammar the method suggested in [4] is in its generality (without optimizations) not very attractive. It is subject for further research to find more efficient transformations.

Acknowledgements. I thank dr. Joost Engelfriet, Gilberto Filè, Kari-Jouko Räihä and prof. Leo Verbeek for their comments on an earlier version of this paper. Many fruitful discussions with dr. Joost Engelfriet contributed to the precision and clarity of the final version.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Reading, Mass.: Addison-Wesley 1974
2. Alblas, H.: A characterization of attribute evaluation in passes, Memorandum 315, Dept. of Appl. Math., Twente University of Technology (1980)
3. Bochmann, G.V.: Semantic evaluation from left to right. Comm. ACM **19**, 55-62 (1976)
4. Engelfriet, J., Filè, G.: Passes and paths of attribute grammars, Memorandum 323, Dept. of Appl. Math., Twente University of Technology (1980). Also to appear in Information and Control
5. Fang, I.: FOLDS, a declarative formal language definition system. Rep. STAN-CS-72-329, Computer Science Dept., Stanford University (1972)
6. Giegerich, R., Wilhelm, R.: Attribute evaluation, in: Le point sur la compilation, Proc. IRIA Symposium on state of the art and future trends in compilation, 337-365 (1978)
7. Jazayeri, M.: On attribute grammars and the semantic specification of programming languages, Rep. 1159, Case Western Reserve University (1974)
8. Jazayeri, M., Walter, K.G.: Alternating semantic evaluator. Proc. ACM 1975 Annual Conference, 230-234 (1975)

9. Kennedy, K., Warren, S.K.: Automatic generation of efficient evaluators for attribute grammars. Proc. Third ACM Symposium on Principles of Programming Languages, 32–49 (1976)
10. Knuth, D.E.: Semantics of context-free languages. Math. Systems Theory **2**, 127–145 (1968) Correction in: Math. Systems Theory **5**, 95–96 (1971)
11. Räihä, K.-J., Saarinen, M.: An optimization of the alternating semantic evaluator. Information Processing Letters **6**, 97–100 (1977)
12. Räihä, K.-J., Ukkonen, E.: Minimizing the number of evaluation passes for attribute grammars. Proc. 7th Int. Colloq. on Automata, Languages and Programming 1980, Lecture Notes in Computer Science **85**, pp. 500–511. Berlin-Heidelberg-New York: Springer 1980

Received October 7, 1980/September 10, 1981