AUTOMATIC GENERATION OF NEAR-OPTIMAL LINEAR-TIME TRANSLATORS FOR NON-CIRCULAR ATTRIBUTE GRAMMARS

by

Rina Cohen and Eli Harry
TECHNION - Israel Institute of Technology

ABSTRACT

Attribute grammars are an extension of context-free grammars devised by Knuth as a formalism for specifying the semantics of a context-free language along with the syntax of the language. The syntactic phase of the translation process has been extensively studied and many techniques are available for automatically generating efficient parsers for context-free grammars. Attribute grammars offer the prospect of similarly automating the implementation of the semantic phase. In this paper we present a general method of constructing, for any non-circular attribute grammar, a deterministic translator which will perform the semantic evaluation of each syntax tree of the grammar in time linear with the size of the tree. Each tree is traversed in a manner particularly suited to the shape of the tree, yielding a near optimal evaluation order for that tree. Basically, the translator consists of a finite set of "Local Control Automata", one for each production; these are ordinary finite-state acyclic automata augmented with some special features, which are used to regulate the evaluation process of each syntax tree. With each node in the tree there will be associated the Local Control Automaton of the production applying at the node. At any given time during the translation process all Local Control Automata are inactive, except for the one associated with the currently processed node, which is responsible for directing the next steps taken by the translator until control is finally passed to a neighbour node, reactivating its Local Control Automaton. The Local Control Automata of neighbour nodes communicate with each other.

The construction of the translator is custom tailored to each individual attribute grammar. The dependencies among the attributes occurring in the semantic rules are analysed to produce a near-optimal evaluation strategy for that grammar. This strategy ensures that during the evaluation process, each time the translator enters some subtree of the syntax tree, at least one new attribute evaluation will occur at each node visited. It is this property which distinguishes the method presented here from previously known methods of generating translators for unrestricted attribute grammars, and which causes the translators to be near-optimal.

INTRODUCTION. Attribute grammars are an extension of context free grammars devised by Knuth [K1] for specifying the semantics of languages along with the syntax. Each grammar symbol has an associated set of attributes specifying the various components of its "meaning", and each production is provided with semantic rules, defining the attributes of symbols in the production in terms of other attributes associated with the production. To find the meaning of a string, first we construct its syntax tree and then we determine the values of all the attributes of symbols in the tree, a process which is called *semantic evaluation* of the tree. An attribute grammar is *non-circular* if the system of semantic definitions given by the grammar avoids circularity in all possible instances. The problem of detecting circularity in an attribute grammar, first solved by Knuth in [K1,2], was shown in [J&O&R] to be of inherent exponential complexity.

Since their definition, attribute grammars have attracted widespread interest in the area of programming languages. They have been used by investigators in fields such as natural language recognition and question answering systems [P], program optimization ([N&A]),[J2]) and the theory of program correctness [G]. But their most important contribution was towards the formal definition of programming languages and automatic design of compilers. Two high-level programming languages, namely SIMULA and PL360, were fully defined and implemented using the attribute grammar model ([Wi],[D]).

When an attribute grammar is used to specify the translation and code generation performed by a compiler, the attributes may represent such things as data types of expressions, symbol table records for use in translating identifiers, register usage information or machine code generated for a statement. The definition of a programming language and its compilation process by means of an attribute grammar offers the following advantages: (i) The semantics is given in a declarative, non-procedural way, and is independent of any parsing scheme. (ii) The semantic description is modular, given on a production by production basis, which makes the definitions more understandable and facilitates the addition and removal of features from the language. (iii) The context sensitive features of the language can be naturally expressed. (iv) The description of the language can be checked for consistency and used for automatic compiler generation.

Despite these advantages, attribute grammars have not become a widely used tool for compiler generation because of the difficulty in obtaining implementations efficient enough for practical use. Until now, efficient translation algorithms have been developed only for restricted classes of attribute grammars ([L&R&S1,2], [A&U2,3],[B],[J1], [K&W]). A general framework for studying deterministic implementation of attribute grammars was developed in [W].

In this paper we present a general scheme for automatically constructing a near-optimal linear-time deterministic translator for any non-circular attribute grammar. The translator will traverse each parse tree of the grammar in a manner which is particularly suited to the shape of the tree, yielding a near-optimal evaluation order for the tree. As a result, our translator will never wander around aimlessly in the tree before reaching a place where new attributes can be computed. Instead, it is guaranteed that each time the translator enters some subtree of the syntax tree, at least one new attribute evaluation will occur at each of the nodes visited.

The construction of the translator is custom tailored to each individual attribute grammar. The dependencies among the attributes in the grammar are analyzed to produce a near-optimal evaluation strategy for that grammar. Basically, the translator consists of a finite set of "Local Control Automata", one for each production. These are finite-state acyclic automata augmented with some special features, which are used to regulate the translation process of each given parse tree. With each interior node of the tree is associated the Local Control Automaton of the production applying at the node. During the translation process, whenever control reaches the particular node, its Local Control Automaton is reactivated, starting from the state in which it was last suspended. The automaton is responsible for directing the evaluation process while control is at the node. When finally control is passed to a neighbor node (along with some parameter), the currently active Local Control Automaton is suspended, and the Local Control Automaton associated with the neighbor node will be reactivated. This goes on until eventually the Local Control Automaton associated with the root of the tree enters a final state, at which point the translation process is complete.

ATTRIBUTE GRAMMARS. An attribute grammar is a context-free grammar augmented with attributes and semantic functions. Formally, an attribute grammar consists of the following:

1. Underlying grammar: A context free grammar $G = (V_N, V_T, P, S)$, where V_N is the set of non-terminals, V_T is the set of terminals, P is the set of productions and $S \in V_N$ is the the start symbol. We assume that there are no useless nonterminals and that S appears on the left hand side of a unique initial production P_0, and does not appear on the right side of any production. Let V stand for $V_N U_T$.

A production $p \in P$ is written in the following form: $p: X_0 \rightarrow X_1 X_2 \ldots X_{n_p}$ where $n_p \geq 1$, $X_0 \in V_N$ and $X_k \in V$ for $1 \leq k \leq n_p$. For convenience, we sometimes write p[k] to mean X_k for $k = 0,1 \ldots n_p$; i.e. p[k] denotes the k'th symbol (from V0 appearing in production p.

2. Attributes: For each symbol $X \in V$, there are two finite disjoint sets, I(X) - the inherited attributes of X, and S(X) - the synthesized attributes of X. For X = S and for $X \in V_T$, we require that $I(X) = \emptyset$. We write A(X) for I(X) U S(X). A production

$$p: \quad X_0 \rightarrow X_1 X_2 \ldots X_{n_p}$$

has the attribute occurrence (a,k), $0 \leq k \leq n_p$, if $a \in A(X_k)$.

3. Semantic functions: For each production $p \in P$ there is a finite set of semantic functions as follows: for every synthesized attribute occurrence (a,k) with k = 0, and for every inherited attribute occurrence (a,k) with $1 \leq k \leq n_p$, there is a semantic function $f^p_{(a,k)}$ which defines the value of attribute occurrence (a,k) from the values of other attribute occurrences appearing in the same production. The value of an attribute occurrence defined by a semantic function is taken from a given set (possibly infinite) of attribute values. The semantic functions usually involve simple operators like assignment, Boolean operators, numerical sums, etc.

Example 1: The following is an example of an attribute grammar:

$V_N = \{S,A\ B\}$	$V_T = \{a,b\}$
$I(S) = \emptyset$	$S(S) = \{s_1\}$
$I(A) = \{i_1\}$	$S(A) = \{s_1,s_2\}$
$I(B) = \{i_1,i_2\}$	$S(B) = \{s_1\}$

Production	Semantics
0: $S \rightarrow abA$	$(s_1\ 0) = (s_1,3) + (s_2,3)$
	$(i_1,3) = (s_1,3)$
1: $A \rightarrow aAbB$	$(s_1,0) = (s_1,2)$
	$(s_2,0) = (s_2,2) + (s_1,4)$
	$(i_2,4) = (i_1,0)$
	$(i_1,4) = 1$
	$(i_1,2) = 1$
2: $A \rightarrow abb$	$(s_1,0) = 1$
	$(s_2,0) = 1$
3: $A \rightarrow ab$	$(s_1,0) = 1$
	$(s_2,0) = (i_1,0)$
4: $B \rightarrow ab$	$(s_1,0) = (i_1,0)$
5: $B \rightarrow ba$	$(s_1,0) = (i_2,0)$

Evaluation of Attribute Grammars: A parse tree of an attribute grammar is a derivation tree of the underlying context free grammar, i.e. a finite tree whose nodes are labelled with symbols from V. For each interior node N there is a production $p \in P$ such that N is labelled with the symbol p[0], has n_p sons, and for each k, $1 \leq k \leq n_p$, the k-th son of N is labelled with p[k]. In this case we say that production p applies at node N. By the k-th neighbor of node N, for $0 \leq k \leq n_p$, we shall mean the father of N in case k = 0, or the k-th son of N in case $1 \leq k \leq n_p$. A parse tree with root X, $X \in V_N$, is a parse tree in which the root is labelled by nonterminal X.

A semantic tree of an attribute grammar is a parse tree augmented with attributes; i.e. each node labelled with $X \in V$ is a structured variable whose field selectors are the elements of A(X). In order to determine the "meaning" or "translation" of a string generated by the underlying context-free grammar, we first parse the string and build the semantic tree. Then we have to fill in the fields of each node by computing the attribute values according to the semantic functions. This process is called evaluation of the semantic tree.

An algorithm which accepts as input a semantic tree generated by a given attribute grammar G and evaluates it is called a translator (or evaluator) for G.

The only restriction imposed on the translator is that at any point in time during the evaluation process it can only evaluate an attribute occurrence whose semantic function is ready to evaluate; that is, all attribute occurrences which are arguments of the semantic function have been previously evaluated. Thus the translator will move in the tree from node to node, at each node evaluate some attributes which are ready to evaluate until all attribute occurrences in the tree have been evaluated.

Consider the attribute grammar of Example 1 and the semantic tree of Fig.1. The fields have been filled in as prescribed by the semantic functions. Note, for example, that i_1 of node 11 cannot be evaluated before s_1 of the same node is evaluated, which in turn demands pre-evaluation of s_1 of node 6.

In later sections we shall describe how to systematically construct an efficient translator for each attribute grammar. The construction of the translator will be carried out once and for all at construction time. Once the translator for the attribute grammar has been constructed, the evaluation process for each semantic tree, as described above, will take place at run time.

The Dependency Graph: Let p be the production $p: X_0 \rightarrow X_1 X_2 \ldots X_{n_p}$. A convenient tool for describing production p and its semantic dependencies is the dependency graph of production p, denoted D_p. The nodes of D_p are the attribute occurrences of p. There is a directed arc from node (a',k') to node (a,k) if (a,k) depends on (a',k') i.e. the semantic function $f^p_{(a,k)}$ uses (a',k') as argument. This means that D_p reflects the dependencies among attribute occurrences imposed by the semantic functions of p. Fig.2 shows the dependency graph of production 1 of Example 1.

In order to represent the semantic dependencies of a whole semantic tree rather than a single production, we define the (compound) dependency graph of a semantic tree T, denoted D(T). This graph is constructed by "pasting together" copies of the D_p's for the productions occurring in the tree.

Each one of the D_p's is selected according to the production p applying at the interior node of the tree. Fig.3 shows the compound dependency graph of the tree of Fig.1.

Circularity: The compound dependency graph of a semantic tree reflects the flow of information among the attributes in the tree. In the semantic tree there is a flow of information from the root towards the leaves by the inherited attributes, and from the leaves towards the root by the synthesized attributes. Since information may flow in both directions, a cycle may be created. An attribute grammar is said to be circular if there exists at least one semantic tree whose dependency graph contains a cycle. Circular grammars are clearly ill-defined in that there is at least one semantic tree which cannot be evaluated.

Knuth [kl,2] has presented an algorithm which tests an attribute grammar for circularity. The time taken by the algorithm is exponential in the size of the grammar. A faster but still expotential algorithm appears in [J&O&R], where it was also established that the circularity problem for attribute grammars is of inherent exponential time complexity.

A Normal Form for Attribute Grammars: In the general definition of attribute grammar, for each production p, every semantic function $f^p_{(a,k)}$ may be defined in terms of all other attribute occurrences, excluding (a,k) itself. Without loss of generality we may assume that a semantic function $f^p_{(a,k)}$ does not use as argument an attribute occurrence, which is defined in the same production. This leads to the following normal form [J1].

Definition: An attribute grammar is said to be in normal form if it satisfies the following restrictions. For each production $p: X_0 \rightarrow X_1 \ldots X_{n_p}$:

(a) A synthesized attribute occurrence (s,0), $s \in S(X_0)$, may depend only on:

(1) Inherited attribute occurrences (i,0), $i \in I(X_0)$;

(2) Synthesized attribute occurrences (s,k), $s \in S(X_k)$, $1 \leq k \leq n_p$;

(b) An inherited attribute occurrence (i,k), $i \in I(X_k)$, $1 \leq k \leq n_p$, may depend only on:

(1) Inherited attribute occurrences (i,0), $i \in I(X_0)$;

(2) Synthesized attribute occurrences (s,k), $s \in S(X_k)$, $1 \leq k \leq n_p$.

There is no loss of generality in imposing the above restrictions, provided there is no local circularity for any production p in the grammar. Local circularity means that the dependency graph D_p of production p contains a cycle, and its existence clearly implies circularity.

THE CHARACTERISTIC GRAPHS. We now construct for each nonterminal of the attribute grammar a collection of directed graphs, called the characteristic graphs of the nonterminal. These graphs play a major role in Knuth's test for circularity [K1] and are essential for the understanding of the evaluation process and translator construction to be described in subsequent sections.

Definition: For each semantic tree T_X with root X, the root dependency graph of the tree T_X is derived from the compound dependency graph of T_X by "projecting" the attribute dependencies on the root X as follows. The set of nodes of the root dependency graph is A(X), and there is an arc from inherited attribute i to synthesized attribute s if the compound dependency graph of T_X has a path from i to s.

Figure 4 shows the root dependency graph of the subtree rooted at node 4 of the tree in Fig. 1.

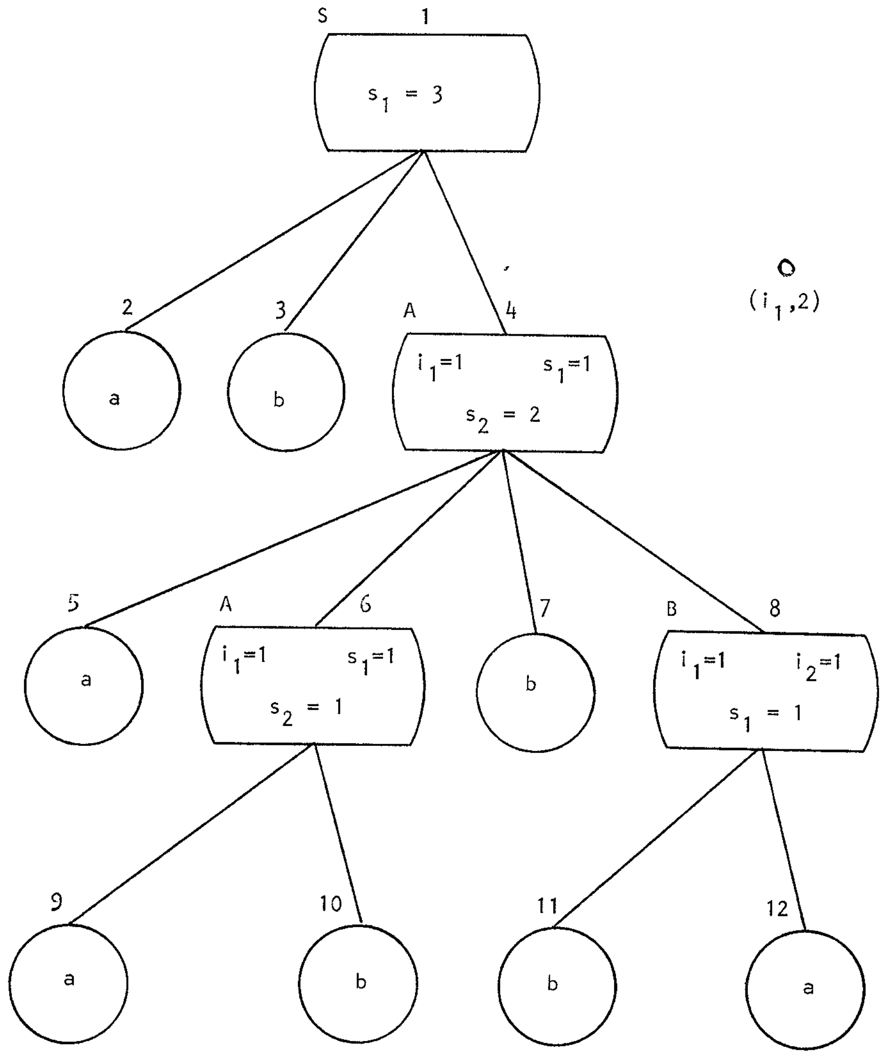


Figure 1

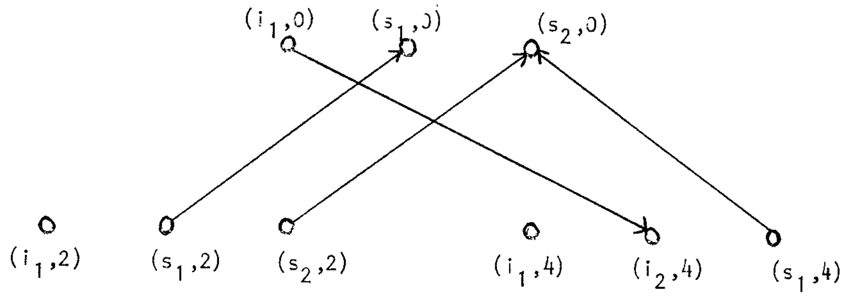


Figure 2

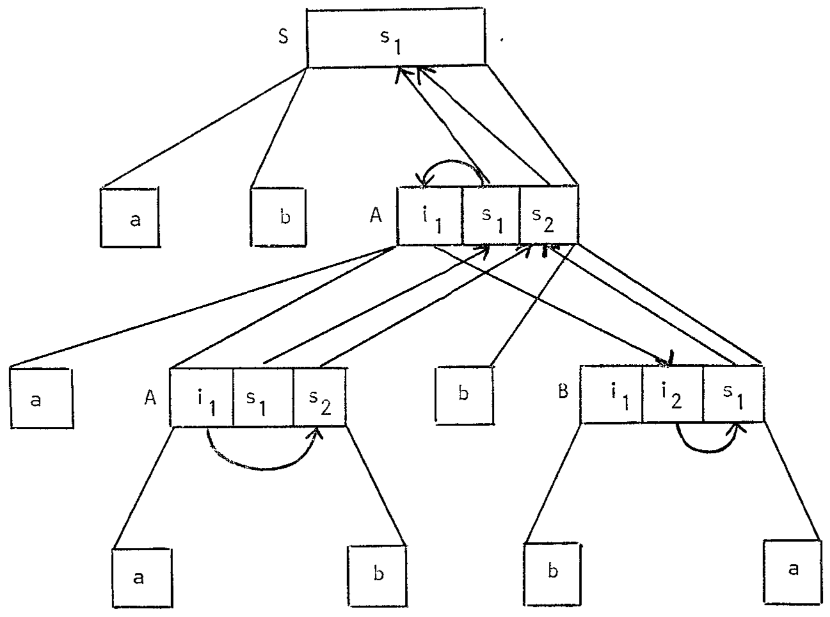


Figure 3.

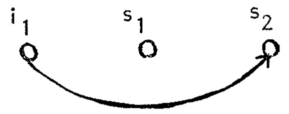


Figure 4

Observing the compound dependency graph of Fig. 3, one can see that there is a path from i_1 of the node 4 to i_2 of node 8 continuing to s_1 of the same node and ending in s_2 of node 4. This path yields the arc from i_1 to s_2 in Fig. 4.

The root dependency graph of a semantic tree T_X reflects the dependencies among the attribute occurrences of the root X induced by the tree T_X. Since the node set is A(X), there can be only finitely many distinct root dependency graphs for trees with root X. In the evaluation process of semantic trees to be described in the following sections, each subtree of the semantic tree will be characterized by its root dependency graph. Define a characteristic graph of nonterminal X to be any graph C with node set A(X). s.t. C is the root dependency graph of at least one semantic tree T with root X. Let $\mathcal{C}_X = \{C_1, C_2, \ldots, C_{m_X}\}$ denote the set of all characteristic graphs of nonterminal X.

Definition: Let D_p be the dependency graph for production $p\colon X_0 \rightarrow X_1 X_2 \ldots X_{n_p}$, and let $G_1, G_2, \ldots, G_{n_p}$ be any given set of directed graphs s.t. for each $k = 1, \ldots, n_p$, the nodes of G_k are the elements of $A(X_k)$. Then the merged graph, denoted $D_p[G_1, G_2, \ldots, G_{n_p}]$, is the directed graph obtained from D_p by adding an arc from (a,k) to (a',k') whenever the graph G_k has an arc from a to a'.

The above definition is illustrated in Fig. 5. Let D_p be the graph of production A→ aAbB in Fig.2, let G_2 and G_4 be the graphs shown in Fig.5 (on top) and let G_1 and G_3 be the empty graphs (corresponding to the terminals a and b with no attributes). The merged graph $D_p[G_1, G_2, G_3, G_4]$ is shown in Fig.5. for instance, the arc from $(i_1, 2)$ to $(s_1, 2)$ originates from the arc from i_1 to s_1 in G_2.

Algorithm 1 - Construction of the Sets of Characteristic Graphs:
(i) Initialization: For each $X \in V_N$ let $\mathcal{C}_X$ be the empty set, and for each $X \in V_T$ let $\mathcal{C}_X$ contain a single graph whose nodes are A(X) and which has no arcs. (ii) Repeat until no more graphs can be added to any of the sets $\mathcal{C}_X$: For each production $p\colon X_0 \rightarrow X_1 X_2 \ldots X_{n_p}$ and for every choice of n_p graphs $C_1, \ldots, C_{n_p}$ s.t. $C_k \in \mathcal{C}_{X_k}$ for $k = 1, \ldots, n_p$, form the graph C' with node set $A(X_0)$, s.t. C' has an arc from node i, $i \in I(X_0)$, to node s, $s \in S(X_0)$, whenever the merged graph $D_p[C_1, C_2, \ldots, C_{n_p}]$ has a path from (i,0) to (s,0). If C' is not yet in $\mathcal{C}_{X_0}$ then add it to $\mathcal{C}_{X_0}$.

It is clear that the above process must ultimately terminate with no more graphs created since there exist only finitely many such graphs. Fig.6 shows the sets of characteristic graphs generated by Algorithm 1 for the attribute grammar of Example 1. Knuth has shown that for each nonterminal X the set $\mathcal{C}_X$ constructed in Algorithm 1 is precisely the set of all characteristic graphs of X. Thus, for every semantic tree T with root X, there is in $\mathcal{C}_X$ a corresponding characteristic graph of X (which coincides with the root dependency graph of T), and vice-versa, for each graph in $\mathcal{C}_X$ there corresponds at least one semantic tree with root X. From now on, the root dependency graph of a semantic tree T will be referred to as the characteristic graph of the tree T.

AN OVERVIEW OF THE EVALUATION PROCESS. Given a semantic tree with root S (the start symbol), the evaluation process consists of evaluating the semantic functions of the attribute occurrences in the tree. Sometimes the goal of the evaluation process is defined to be the evaluation of some distinguished attribute of the root S. Here we shall consider evaluation to be complete only when all attribute occurrences in the tree have been evaluated.

The evaluation process is carried out in two phases. The first phase serves as a preparation for the evaluation itself, which takes place in the second phase.

The first phase consists of a depth-first left to right postfix traversal of the semantic tree. The computation in each node takes place after all of its sons have been computed. During this phase, the translator will compute for each node N_X (labelled by $X \in V$) of the semantic tree, the characteristic graph of the subtree rooted at N_X. This characteristic graph belongs to the set $\mathcal{C}_X$ of characteristic graphs of X, and represents the dependencies among the attributes of N_X imposed by the structure of the subtree of N_X.

During the second phase the translator will again process the semantic tree; however, this time the order in which the nodes are processed is not known a priori, but is determined dynamically at runtime according to the individual structure of the tree.

We may view the second phase as a processor with control that always points to some "current" node of the semantic tree. While at a node N_X, the processor may perform one of the following types of elementary actions (or instructions):
(i) CALL(a,k) - call for the evaluation of semantic function $f^p_{(a,k)}$, where p is the production applying at the node N_X.
(ii) TRANSFER DOWN - transfer control down to a specific son of the current node N_X.
(iii) TRANSFER UP - transfer control up to the father of the current node.

Starting with control pointing to the root of the semantic tree, the translator will process the tree until all attribute occurrences in the tree have been evaluated, at which point control returns.

Let us examine the evaluation process from the point of view of an individual node N_X. During evaluation control is moving around the tree, and in the meanwhile the processor computes new attribute values. From the point of view of our node N_X, nothing seems to happen until for the first time control arrives at N_X from above. Some semantic functions associated with the production applying at N_X may then be evaluated, and afterwards control leaves N_X in some direction. Later on control returns from the same direction it left, perhaps some more attribute occurrences are evaluated, and again control leaves the node. This goes on until eventually all attribute occurrences of the production applying at N_X have been computed and control leaves N_X going towards the root, never to return. Thus from a local point of view, evaluation consists of comings and goings of control, interspersed with evaluations of attributes performed while at our node. We shall refer to a part of the evaluation process from the point control leaves our node N_X

towards its k-th neighbour, and until the next time control returns to our node, as a visit to the k-th neighbour of N_X.

At each stage of the above evaluation process, the choice of the next elementary action to be taken by the evaluator at the particular node N_X may depend on the following: (a) The production p which applies at the node N_X, and its associated semantic functions; (b) The current "state of affairs" of the evaluation process, as regards our node N_X, i.e. the set of attribute occurrences of the production p applying at N_X, which are currently available; (c) The dependencies among the attributes of N_X, which are induced by the subtree rooted at N_X; these dependencies are represented by the characteristic graph of this subtree, which has been computed during the first phase.

To keep track of the sets of available attributes and to regulate the evaluation process at the particular node N_X, the Local Control Automaton, (LCA) $\underline{A_p}$ for production p is introduced. The Local Control Automaton is an ordinary finite state automaton augmented with some special features in its transition function. Each state of the Local Control Automaton represents a set of attribute occurrences of production p, which are available upon entering this state. Each transition in the automaton represents an instruction of one of the types CALL (a,k), TRANSFER DOWN or TRANSFER UP.

Associated with each node of the semantic tree is the Local Control Automaton $\underline{A_p}$ of the production p which applies at the node. The LCA $\underline{A_p}$ is responsible for directing control when it arrives at the node, and it calls for the evaluation of the semantic functions of production p. Whenever control arrives at a node, it (re-)activates its LCA; if it is the first time control reaches this node, then the LCA starts from its initial state; otherwise it resumes action from the state in which it left off.

In our translator control must be capable of passing parameters among the various LCA's. This passing of parameters serves as a means of communication between the LCA's of neighbour nodes and is used for two main purposes: (a) When control is transferred down to a son of the current node, the translator has to inform the receiving LCA what activities took place while the latter was inactive. Specifically, the parameter passed down is a set of inherited attributes of the receiving node, which are currently available. These inherited attributes are used by the receiving LCA to determine its next move, and their values may be used by the translator as arguments in computing semantic functions. (b) When control is transferred up in the tree, the translator has to inform the father's LCA about the activities which took place in the subtree just visited. Specifically, the parameter returned to the father is the set of synthesized attributes of the son just visited, which are currently available.

In both cases, we see that the parameter carried along by control is a set of currently available attributes of thenode visited. Such parameters will be called transmitted sets.

The evaluation process described above uses the LCA's as basic building blocks. In order to obtain a translator for a given attribute grammar, a finite set of LCA's, one for each production, will have to be constructed.

LOCAL CONTROL AUTOMATA. As mentioned above, an LCA $\underline{A_p}$ will be associated with each production p of the attribute grammar. At runtime, to each node of the semantic tree there will be attached the LCA of the production applying at the node. Whenever control reaches a particular node, the LCA of the node will be reactivated, and will dictate the next elementary actions to be taken by the translator. When eventually control is transferred to a neighbour node, the LCA will be suspended and will send along with control a parameter - the transmitted set.

Definition: Consider a production p: $X_0 \to X_1 X_2 \ldots X_{n_p}$. Let A(p) denote the set of all attribute occurrences of p. For each k = 0,1,...,n_p, define:

(a) For each set of attributes $T \subseteq A(X_k)$

$$T * k = \{(a,k) \mid a \in T\}.$$

(b) For each set of attribute occurrences $A \subseteq A(p)$

$$A/k = \{a \mid (a,k) \in A \quad \left.\begin{matrix}\text{if } k = 0 \text{ then } a \in S(X_k) \\ \& \\ \text{if } 1 \le k \le n_p \text{ then } a \in I(X_k)\end{matrix}\right\}$$

The operator * transforms a set of attributes of X_k onto the corresponding set of attribute occurrences of production p, while the operator / maps a set of A of attribute occurrences of production p onto a set of A/k of attributes of X_k. Note that for k = 0, A/k contains only synthesized attributes, while for $1 \le k \le n_p$, A/k contains only inherited attributes. A/k will be called the k-th projection of A.

Definition: A Local Control Automaton (LCA) $\underline{A_p}$ for production $X_0 \to X_1 X_2 \ldots X_{n_p}$ is an ordinary finite state automaton augmented with some features in its transition function.

The set of states is divided into two subsets: (a) Active states. Each active state is labelled with a set of attribute occurrences $A \subseteq A(p)$. No two active states are labelled with the same set A. An active state will therefore be identified by its label A, and will be denoted by $[A]^p$ (the superscript p will be omitted when p is understood).
(b) SUSPEND states. Each suspend state is labelled by a pair : (k,A), where $0 \le k \le n_p$ and $A \subseteq A(p)$ as above. For each pair (k,A) there is at most one SUSPEND state labelled with this pair. We shall denote this state by $SUS_k[A]^p$ (with the superscript p omitted when p is understood).

During the evaluation process, whenever LCA $\underline{A_p}$ is found in an active state [A] or enters a SUSPEND state $SUS_k[A]$ ($0 \le k \le n_p$), then A is precisely the set of attribute occurrences of production p which have been evaluated so far.

There are severaltypes of transitions in an LCA: Transitions leaving Active states: Each transition leaving an active state corresponds to one of the following three types of instructions:
(i) CALL(a,k) - call for the evaluation of semantic function $f^p_{(a,k)}$;
(ii) $TRANSFER_0$ - transfer control up to the 0-th neighbour (i.e. father) of the current node;
(iii) $TRANSFER_k$ ($1 \le k \le n_p$) - transfer control down to the k-th neighbour (i.e. k-th son) of the current node.

Every active state [A] may have precisely one of the following three types of exits:
(i) CALL transition - Exit from [A] leading to another active state [A]. Such a transition is labelled by CALL(a,k), where $(a,k) \in A(p)-A$ and $A' = A \cup \{(a,k)\}$. There can be no other transition leaving state [A].

A transition CALL(a,k) represents an instruction for evaluating the semantic function $f^p_{(a,k)}$ and storing the result in the appropriate field of the semantic tree. Such an instruction is executable only if the semantic function $f^p_{(a,k)}$ is ready to evaluate. Therefore we shall impose the additional requirement that whenever transition CALL(a,k) leaves state $[A]^p$, all attribute occurrences (a',k') which are used as arguments in $f^p_{(a,k)}$ are contained in A. Figure 7 illustrates a CALL transition; note that the active states are represented by circles.

(ii) Unconditional TRANSFER Transition - A transition from active state [A] to a SUSPEND state $SUS_k[A]$, where $0 \leq k \leq n_p$, and $X_k \in V_N$. The transition is labelled with $TRANSFER_k$ and represents an instruction to the translator to transfer control to the k-th neighbour LCA. Again this is the only transition leaving state [A]. Fig. 8 illustrates an unconditional TRANSFER transition. Note that the SUSPEND states are represented by squares.
(iii) Conditional TRANSFER Transitions - A set of t+1 exits (for some $1 \leq t \leq n_p$) from active state [A], leading to t+1 distinct suspend states $SUS_{i_1}[A]$, $SUS_{i_2}[A],\ldots,SUS_{i_t}[A]$, $SUS_0[A]$ (same A as for the active state), where $1 \leq i_1 < i_2 <\ldots< i_t \leq n_p$ and X_j is a nonterminal for $j = i_1,i_2,\ldots,i_t$. The transition into state $SUS_j[A]$ is labelled by the pair $(TRANSFER_j,\mathcal{C}^{(j)})$, where $\mathcal{C}^{(j)}$ is a subset of $\mathcal{C}_{X_j}$, the set of characteristic graphs of nonterminal X_j. The last transition into state $SUS_0[A]$ is labelled by $TRANSFER_0$ only. Figure 9 illustrates a set of conditional TRANSFER transitions.

The above set of conditional TRANSFER transitions leaving state [A] has the following meaning. During the evaluation process of a given semantic tree, if LCA $\mathcal{A}_p$ is associated with node N_{X_0}, and $\mathcal{A}_p$ enters state [A], the exit to be taken from [A] depends on the characteristic graphs of the sons of N_{X_0} in the tree. Specifically, for each $j = i_1,i_2,\ldots,i_t$, the conditional TRANSFER transition labelled by $(TRANSFER_j, \mathcal{C}^{(j)})$ will be called admissible iff the characteristic graph attached to the j-th son of N_{X_0} in the semantic tree belongs to $\mathcal{C}^{(j)}$. As we shall see below, a transition $(TRANSFER_j,\mathcal{C}^{(j)})$ is admissible precisely when the corresponding visit to the j-th son of the current node N_{X_0} is guaranteed to produce at least one new synthesized attribute of the j-th son. Therefore when the LCA of node N_{X_0} enters state [A], the conditional TRANSFER transitions leaving state [A] will be checked one by one until the first admissible transition is encountered; this will be the transition to be taken. If, however, none of the first t transitions is admissible, then the $TRANSFER_0$ transition will be taken. Thus control will be transferred up to the father of N_{X_0} only if in the current situation, no more attribute occurrences can be evaluated for production p, not even after some more processing of the subtree of N_{X_0}. Before any further evaluation at N_{X_0} can take place, some new inherited attributes of N_{X_0} must become available.

Transmitted Sets. Whenever a $TRANSFER_k$ instruction (either conditional or unconditional) leading into state $SUS_k[A]$ is executed, a transmitted set T is passed as parameter to the k-th neighbour LCA along with control. Specifically, this set is T = A/k - the projection of A on the k-th neighbour. For k=0 this set T consists of all synthesized attributes of X_0, which are currently available, and for $k = 1,2,\ldots,n_p$, T consists of all inherited attributes of X_k which are currently available.

Transitions Leaving SUSPEND States. Every state $s = SUS_k[A]$ may have several, say $\ell \geq 1$, exits, each labelled with a distinct set T_i, $1 \leq i \leq \ell$ of attributes of X_k, s.t. $T_i * k \nsubseteq A$. Let these exits be denoted by $E(s,T_1),E(s,T_2),\ldots,E(s,T_\ell)$. If k=0 then T_i is a set of inherited attributes of the $\ell.h.s.$ X_0, and if k > 0 then T_i is a set of synthesized attributes of the son X_k. The exit labelled by $E(s,T_i)$ will lead to the active state $[A \cup T_i * k]$. Since by assumption $T_i * k \nsubseteq A$, no transition leaving state $SUS_k[A]$ will lead into state [A] again. Fig. 10 illustrates the exits from a SUSPEND state.

To explain the meaning of the sets T_i, recall that during runtime, when the LCA $\mathcal{A}_p$ attached to node N_{X_0} enters state $SUS_k[A]$, a $TRANSFER_k$ instruction leading into this state is executed. The LCA of node N_{X_0} is then suspended, and control is transferred to the k-th neighbour LCA. After some activities take place in other regions of the tree, control finally returns to the node N_{X_0} from its k-th neighbour, carrying with it a transmitted set T. At this point the LCA $\mathcal{A}_p$ at N_{X_0} is reactivated, starting from the same state $SUS_k[A]$. The exit taken from this state is chosen according to the transmitted set T received; namely, it will be precisely the exit whose label T_i coincides with T. In this way, the transmitted set returned by the k-th neighbour determines the next move of the LCA after it resumes action in state $SUS_k[A]$. The sets T_i labelling the exits from $SUS_k[A]$ represent all possible transmitted sets, which can be returned by any k-th neighbour LCA in any particular situation.

Initial and Final States.
(1) Each LCA has precisely one initial state, a state with no in-arcs, which is a state $SUS_0[A]$ (for some set A).
(2) Each LCA has at least one terminal state, a state with no out-arcs. Every terminal state is a state $SUS_0[A]$ (for some set A).

Let us now summarize the structural properties of LCA's:

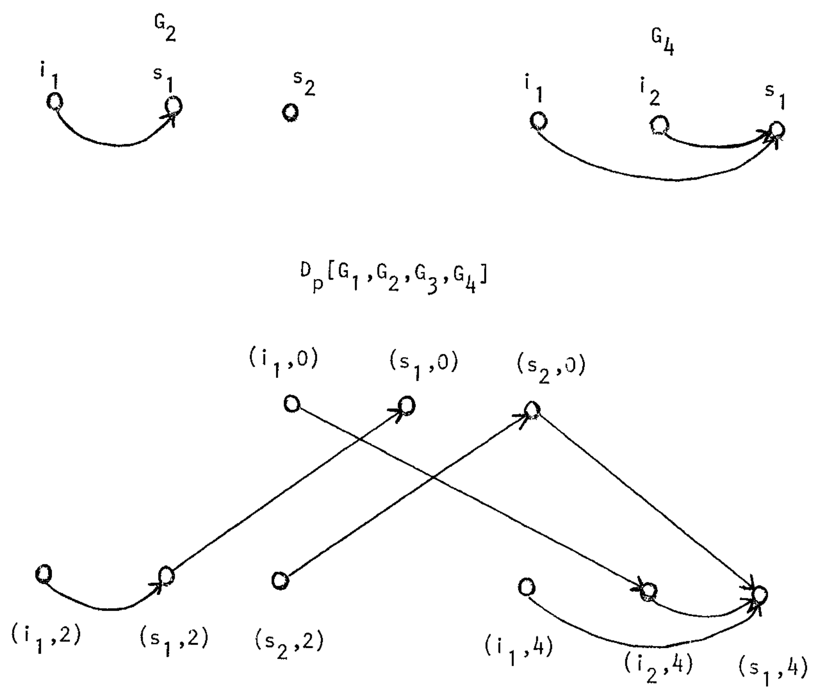


Figure 5

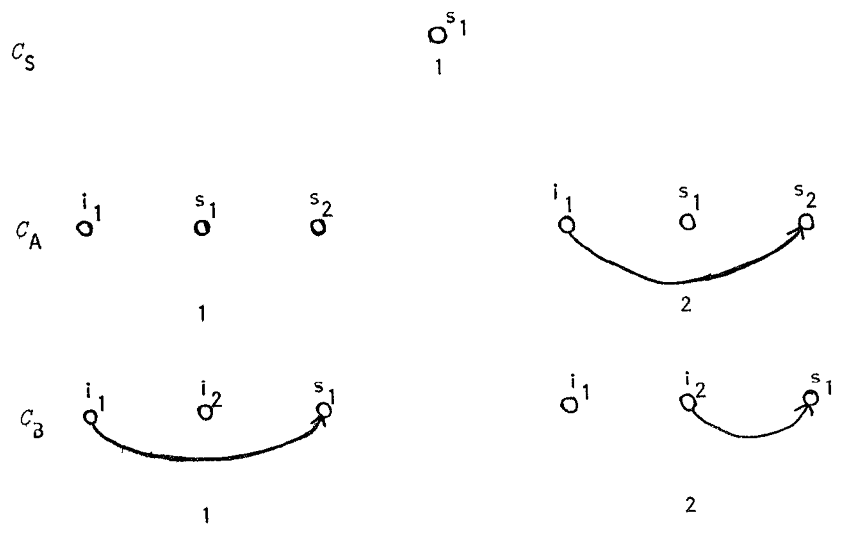


Figure 6.

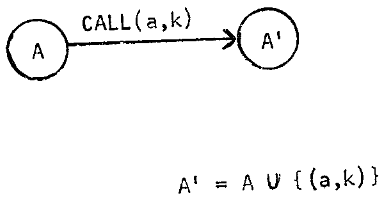


Figure 7

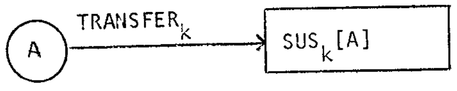


Figure 8.

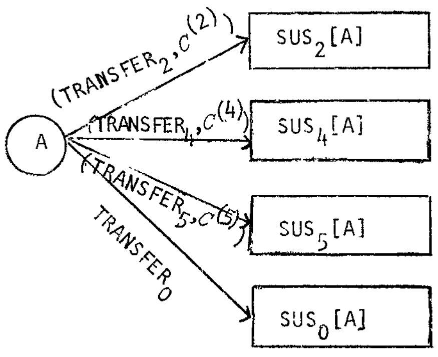


Figure 9

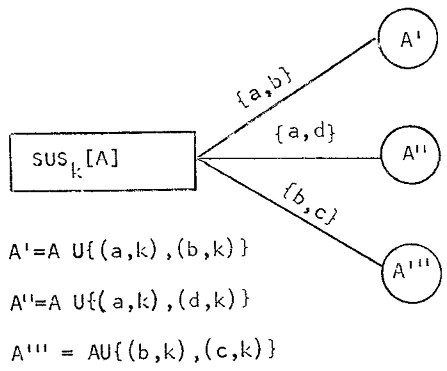


Figure 10

(1) Each LCA is acyclic. This is because for every path in the LCA, the sets of attributes A labelling both active and SUSPEND states along the path must be ordered by inclusion, and no more than two consecutive states can be labelled by the same set.
(2) Every SUSPEND state $SUS_k[A]$ (excluding the initial state) has precisely one in-arc (from the active state with same label [A]), while there may be several in-arcs for any active state.

Figure 11 illustrates a complete set of LCA's for the attribute grammar of Example 1. Each state is given a number, C is an abbreviation for CALL transitions, the set A of available attribute occurrences and the label $TRANSFER_k$ were omitted.

HOW TO EVALUATE A SEMANTIC TREE WITH A GIVEN SET OF LCA's. Suppose that we are given a translator for some noncircular attribute grammar. The translator is made up of a finite set of LCA's, one for each production.

In order to carry out the evaluation process, two variables will be associated with each node N_X of the semantic tree:
(1) A variable indicating one of the characteristic graphs C_i of non-terminal X, namely the characteristic graph of the subtree rooted at node N_X. This variable is computed during the first phase of the evaluation process. (2) A variable which, during the second phase, will store the state of the LCA of N_X at the time it is suspended, while control wanders in other regions of the semantic tree. When control returns to N_X, the LCA of N_X will be reactivated, starting from this state. This variable is initialized during the first phase.

The First Phase: The first phase consists of a depth-first left to right postfix traversal of the semantic tree. During this traversal, with each node of the tree, there will be associated the characteristic graph of the subtree rooted at the node. The characteristic graph associated with a terminal node N_t, $t \in V_T$, will be the trivial graph with node set A(t) and no arcs. Due to the postfix manner of traversal, when the translator reaches a nonterminal node N_X to compute its characteristic graph, the characteristic graphs of all sons of N_X have already been computed.

Let us describe the construction of the characteristic graph for node N_X, where the production applying at the node is p:

$X_0 \to X_1 X_2 \dots X_{n_p}$, and $X = X_0$.

Let the characteristic graphs associated with nodes N_{X_i}, $i = 1,2\dots,n_p$, be D_i. Form a graph C whose nodes are $A(X_0)$ such that C has an arc, from inherited attribute i to synthesized attribute s whenever the merged graph $D_p[D_1,D_2,\dots,D_{n_p}]$ has a path from (i,0) to (s,0). C is precisely the characteristic graph of node X_N.

To avoid the need for storing characteristic graphs as part of the translator, and then at runtime comparing the characteristic graph C constructed above against all graphs in the set $\mathcal{C}_{X_0}$, some kind of Göedel numbering for graphs can be used. Each graph will be identified by its "Göedel number" (which will constitute the first variable of the node N_X) and during runtime comparison will take place only between the Göedel numbers.

An alternative way for obtaining the characteristic graph C would be to use a look up table, which gives for each production p and for each set of characteristic graphs $\{D_1,D_2,\dots,D_{n_p}\}$ as above, the corresponding characteristic graph C. Such a look up table can be prepared once and for all at construction time, while computing the characteristic graphs. When using this method there is no need for Göedel numbering and each graph will be represented simply by its index in the set $\mathcal{C}_{X_0}$. The graphs themselves need not be kept in memory, but the look up table will have to be stored as part of the translator.

After determining the characteristic graph of N_X and the appropriate LCA $\mathcal{A}_p$ according to the production p applying at node N_X, the second variable of the node is initialized to the initial state of $\mathcal{A}_p$.

The Second Phase. The manner in which the semantic tree is traversed during the second phase is custom tailored to the individual structure of the tree. Evaluation begins by sending control to the root LCA $\mathcal{A}_{p_0}$. Control begins executing the instructions of $\mathcal{A}_{p_0}$, which after a while transfers control down to one of its sons. Thus control will start wandering in the tree from node to node in an order dictated by the LCA's, and by the characteristic graphs associated with the nodes. At each point in time during the evaluation process, all LCA's of the tree are dormant except for one which is active. The active LCA may call for the evaluation of semantic functions, or it may direct control to one of its neighbour LCA's. Eventually, when all subtrees of the semantic tree have been evaluated, control will return for the last time to the root LCA, which (possibly after executing a few CALL instructions) will enter its final state $SUS_0[A]$, and control will leave the tree from above. Evaluation is then complete.

Algorithm 2 - Evaluation of a Semantic Tree
(1) Perform the first phase.
(2) Transfer control to the initial state of the root LCA $\mathcal{A}_{p_0}$ along with an empty transmitted set.
(3) Repeat until a terminal state of $\mathcal{A}_{p_0}$ is entered:

Let the currently active LCA $\mathcal{A}_p$ be in state $s = SUS_k[A]$ and let T be the transmitted set received upon reactivation of this LCA.
(i) Take the exit from s labelled by T; let t be the active state entered.
(ii) Execute the instructions of $\mathcal{A}_p$, starting from the state t, until a SUSPEND state, say $SUS_{k'}[A]$, is reached.
(iii) Transfer control to the k'-th neighbour LCA along with the transmitted set A'/k'. Reactivate the k'-th neighbour LCA, starting from the SUSPEND state stored in the second variable of the k'-th neighbour node.

Figure 12 illustrates the evaluation process of a semantic tree of the grammar of Ex. 1 according to Algorithm 2, using the set of LCA's in Fig.11.

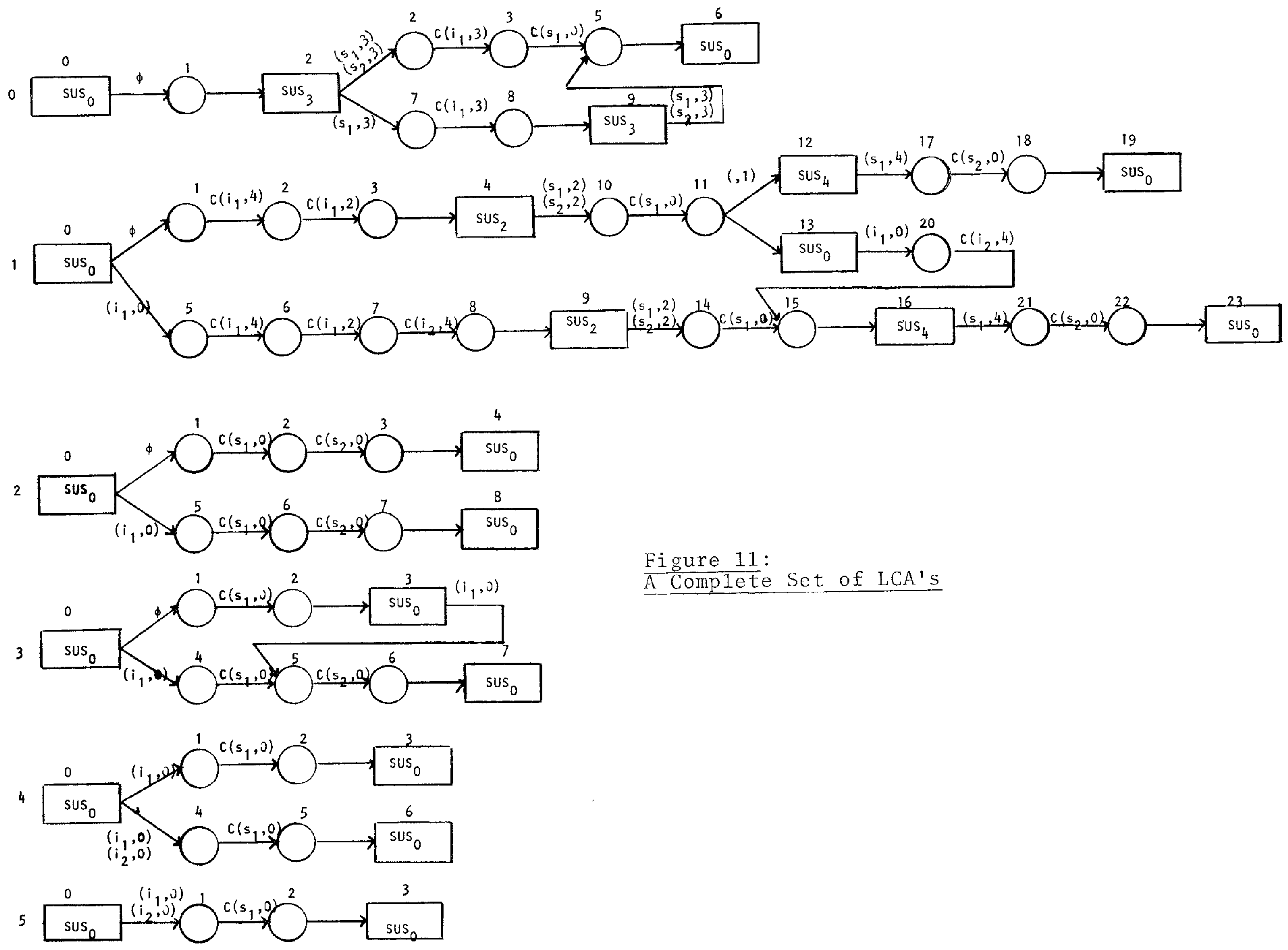
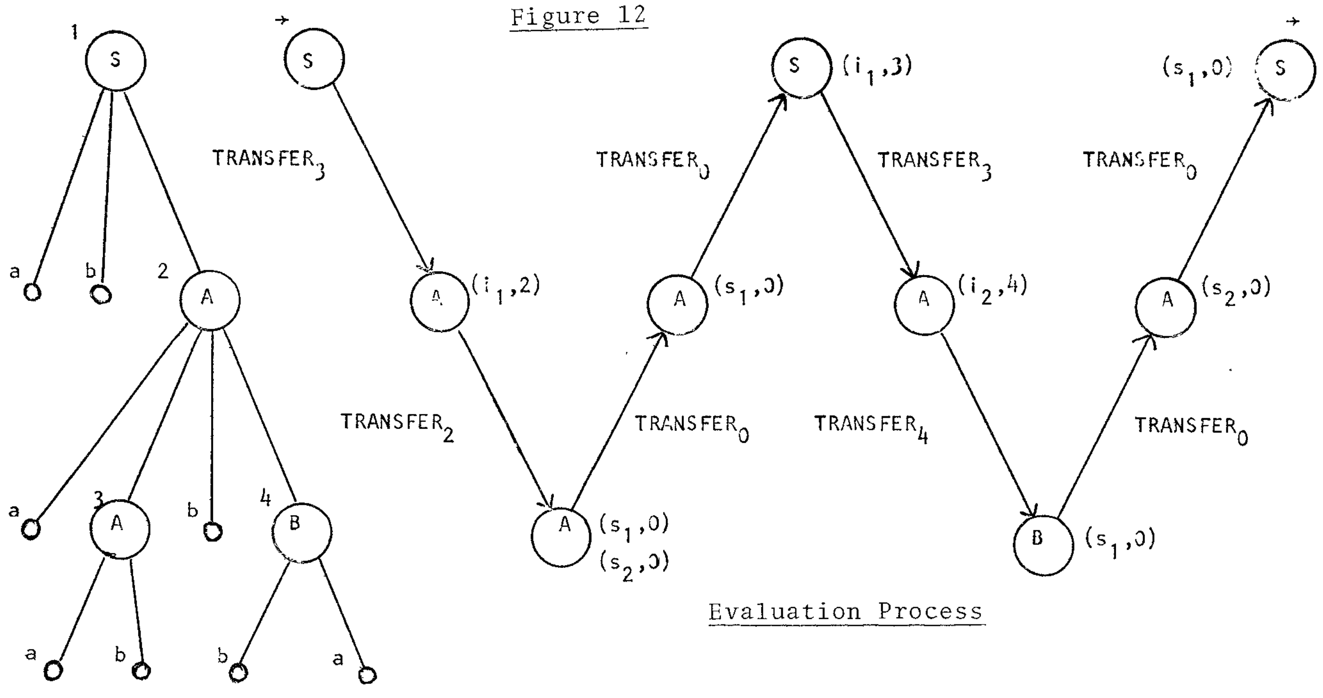


Figure 11:
A Complete Set of LCA's

Figure 12



Evaluation Process

Parse Tree

Starting from the root, control moves down to node 2 (due to entering state 3 in LCA A_0), then down to node 3; for each node visited, the attributes evaluated at the node are indicated in the figure; e.g. at node 3, s_1 and s_2 are evaluated and then control moves up back to the root. After evaluating the inherited attribute i_1 of node 2, control moves down again, this time to node 4; after evaluating s_1 for node 4, then s_2 of node 2, control returns to the root, and evaluation is complete. We see that every visit to every node in the tree produces at least one new attribute value.

To ensure that step 3(i) in Algorithm 2 can always be carried out, the set of LCA's must satisfy the following condition.

The Closure Condition: Whenever at runtime, control is transferred to an LCA A_p with transmitted set T, the resumed SUSPEND state in A_p must have an exit labelled by T.

The above Closure Condition assures us that Algorithm 2 will always terminate upon entering a final SUS_0 state of the root LCA A_{p_0}.

Theorem 1: If the set of LCA's for attribute grammar G satisfies the Closure Condition, then Algorithm 2 terminates for each semantic tree T after executing $O(|T|)$ elementary operations (where $|T|$ indicates the size of T).

The Completion Attribute: To guarantee that the above evaluation aglorithm will result in the complete evaluation of every semantic tree, a slight modification of the attribute grammar has to be made. A new dummy synthesized attribute, called the completion attribute, is added to each nonterminal. For each production p: $X_0 \to X_1 \dots X_{n_p}$, the completion attribute of X_0 is defined in terms of all inherited attributes of X_0 and of all synthesized attributes (including the completion attribute) of all sons X_k, $1 \le k \le n_p$.

An attribute grammar to which a completion attribute as above has been added for each nonterminal will be called an augmented attribute grammar. In an augmented grammar, the completion attribute is dummy in the sense that it is not computed at runtime. However, this attribute enables us to construct such translators in which the complete evaluation of every semantic tree will be enforced. For this reason we add to the definition of LCA's the following condition:

Completion Condition: For each terminal state $SUS_0\ [A]^{p_0}$ of the root LCA A_{p_0}, the set A must contain the completion attribute of the start symbol S.

Theorem 2: For every augmented attribute grammar G and for every set of LCA's for G which satisfies both the Closure Condition and the Completion Condition, Algorithm 2 fully (and correctly) evaluates each semantic tree of G in time proportional to the size of the tree.

THE CONSTRUCTION OF A TRANSLATOR FOR A GIVEN ATTRIBUTE GRAMMAR

In this section we describe informally how a complete set of LCA's satisfying the Closure Condition and the Completion Condition can be systematically built for each non-circular attribute grammar.

The LCA's are constructed in parallel, state by state and transition by transition, until all of them are complete. The construction is based on simulation of all possible situations that can arise at runtime.

Before starting the actual construction of LCA's, the set of characteristic graphs C_X for each nonterminal X is constructed. After this preliminary computation the construction process is initialized by defining for each production an LCA consisting of the initial state alone. As the algorithm proceeds the LCA's are expanded by adding new states and new transitions in a specific order, so as to ensure that the Closure Condition will be always satisfied. For this reason, whenever we add to LCA A_p a $TRANSFER_k$ transition leading into state $SUS_k[A]$, we must be able to identify all SUSPEND states in all LCA's, which at runtime might be reactivated as a consequence of executing this $TRANSFER_k$ instruction; each of these SUSPEND states must have an exit labelled by the transmitted set $T = A/k$.

Definition: For every non-initial SUSPEND state $s = SUS_k[A]^p$ in LCA A_p, define the set REACTIVATE(s) to be the set of all SUSPEND states in all LCA's to which, under certain conditions at runtime, control might be transferred as consequence of suspending LCA A_p in state $SUS_k[A]$. The states in $REACTIVATE(SUS_k[A]^p)$ may belong to any LCA $A_{p'}$, which, in some semantic tree, interacts with LCA A_p as its "k-th neighbour". Therefore if $k \neq 0$ then p[k] must coincide with p'[0] and if k = 0 then p[0] must coincide with one of the sons of p'.

The next definition enables us to keep track of the states in which the neighbour LCA's were left after the "most recent" visit.

Definition: For each exit E[s,T] from a SUSPEND state $s = SUS_k[A]^p$, define the set SOURCE(E[s,T]) to be the set of all SUSPEND states $SUS_{k'}[A']^{p'}$ in all LCA's, which at runtime may pass control to LCA A_p and cause it to be reactivated in state s and take the exit E[s,T]. Clearly SOURCE(E[s,T]) will consist only of states of the form $SUS_{k'}[A']^{p'}$ whose transmitted set A'/k' coincides with T. Moreover, if k = 0, then p'[k'] = p[0], while if $k \neq 0$ then p[k] = p'[0].

Thus each SUSPEND state s is associated with its REACTIVATE set, and each transition E[s,T] leaving a SUSPEND state s is associated with its SOURCE set SOURCE(E[s,T]). These sets are kept in memory and updated during the construction process.

A central role in the construction process is played by the queue Q. Q consists of couples of the form:

$$(E[s^p,T],s^{p'}),$$

where s^p and $s^{p'}$ are SUSPEND states in A_p and $A_{p'}$ respectively and $E[s^p,T]$ is an exit from s^p labelled by T.

The meaning of such a couple appearing in Q is that exit $E[s^p,T]$ has to be created in A_p (unless already there), and state $s^{p'}$ must be added to SOURCE ($E[s^p,T]$). Note that by definition of the SOURCE set it follows that T must be precisely the transmitted set of $s^{p'}$.

The construction process is based on retrieval and processing of couples from Q, one at a time. The processing of a couple ($E[s^p,T],s^{p'}$) consists of creating the exit $E[s^p,T]$ and adding $s^{p'}$ to its SOURCE set. The exit will lead to an appropriate active state [A U T * k] which must also be added if absent. This new active state will then be "developed", which in turn may cause the addition of some new couples to Q. The process goes on until eventually Q remains empty, at which point construction is completed.

Developing an Active State. Recall that an active state $[A]^p$ may have three types of exits: a single CALL(a,k) transition, a single unconditional TRANSFER$_k$ transition, or a set of conditional TRANSFER transitions. In developing the active state $[A]^p$, we first try to construct a CALL(a,k) transition out of this state, for some attribute occurrence (a,k) of production p s.t. $(a,k) \notin A$. Such a transition can be created only if the semantic function $f^p_{(a,k)}$ is ready to evaluate, i.e. depends only on attribute occurrences in A. If such an exit is indeed created, it leads into another active state [A U {(a,k)}] and we proceed to develop this new state (unless this state already existed before in A_p).

In case no CALL(a,k) exit out of state $[A]^p$ can be created, we attempt to construct an unconditional TRANSFER$_k$ exit for some $k = 1,\dots,n_p$. If this turns out to be impossible as well, a set of conditional TRANSFER transitions will be constructed.

Processing a New SUSPEND STATE. When a new SUSPEND state s is created, it is not "developed" in the usual sense, i.e. no exits from it are constructed right away. Instead, the REACTIVATE set of s is computed, giving rise to a set of couples to be entered into the queue Q. These couples represent the need to construct new exits (labelled with the transmitted set of s) from all SUSPEND states in REACTIVATE(s). It follows that exits from SUSPEND states are created only via retrieving and processing couples from the queue Q, and different exits from the same SUSPEND state are created at different times.

The Construction Algorithm uses a subroutine Develop ($[A]^p$) for developing an active state. After computing the characteristic graphs and initializing the queue Q and the LCA's, a loop for processing the queue Q is entered. Unfortunately, a full presentation of the Construction Algorithm requires quite a few additional technical definitions and is therefore omitted here. The reader is referred to [C&H] for a complete formal description of the algorithm, including a proof of the following theorem:

Theorem 3: For every augmented non-circular attribute grammar, the Construction Algorithm terminates with a set of LCA's which have the structure described above and which satisfy both the Closure Condition and the Completion Condition.

MAIN THEOREM: For every augmented non-circular attribute grammar, there can be constructed a translator, based on a set of LCA's, which will perform the complete evaluation of each semantic tree (using Algorithm 2) in time proportional to the size of the tree and in a near-optimal fashion.

RELATION TO PREVIOUS WORK: The first implementation of attribute grammars is due to Fang [F], who used parallel processes, one for each semantic function.

A deterministic approach was first developed by Lewis, Rosenkrantz and Stearns [L&R&S1] and by Bochman [B], who introduced an algorithm which traverses the tree in a depth-first left-to-right fashion, performing evaluation of all attributes in a single pass. Because of this restriction, the class of attribute grammars for which this method applies (named "L-attributed" in [L&R&S1,2]) is rather limited. To increase the class of attribute grammars that can be efficiently evaluated, Bochman proposed to allow evaluation to occur in several left-to-right passes such that on each pass the attributes evaluated by previous passes can also be used.

Jazayeri [J1], observing that not all programming language features are amenable to evaluation from left to right, extended Bochman's method by introducing the "Alternating Semantic Evaluator" that alternately makes left-to-right, then right-to-left passes. Jazayeri showed that certain left-recursive situations could be evaluated by a single right-to-left pass, even though no fixed number of left-to-right passes was sufficient. Kennedy and Warren [K&W] noted that Jazayeri's extension still leaves many attribute grammars which cannot be evaluated by any fixed number of alternating passes. They exhibited an example of an attribute grammar with a left-recursive rule $B \rightarrow Bb$, such that the first visit to a B-node son cannot be made until during the second visit to its B-node father. No method of evaluation in passes can handle such a grammar, for which "nested passes" are required. Kennedy and Warren were the first to develop a deterministic approach in which the traversal order is not determined a priori, but is tailored to given attribute grammars by analyzing their dependency constraints. Their "treewalk evaluator" works like a recursive routine with a tree node to visit as parameter; while at a node, the evaluator may evaluate semantic functions or call itself recursively to visit sons. Their construction works only for a restricted class of attribute grammars - the "absolutely noncircular" grammars. For a grammar in this class, the evaluator's action at a node need not depend on the structure of the node's subtrees. In our terminology, the absolutely noncircular attribute grammars are the ones for which our translator construction will yield LCA's without any conditional transfer instructions. For such grammars the evaluation algorithm can be significantly simplified by eliminating altogether the first phase, because in this case the characteristic graphs of the subtrees need not be computed. In [Wa] Warren introduced a general model for deterministic evaluation of attribute grammars, called the "coroutine evaluator", and developed general methods for constructing such evaluators. However, both the "treewalk evaluator" and the "coroutine evaluator", are not near-optimal, because they may wander in the tree making a lot of futile visits to subtrees before finally reaching a node where a new attribute value can be produced.

In this paper, following the deterministic approach suggested in [K&W] and [W], we have presented a general construction of a near-optimal translator for any non-circulator attribute grammar. By introducing the characteristic graph as the main tool for analysing (and representing) dependencies among attribute occurrences in a semantic tree, we have been able to obtain a near-optimal evaluation strategy which takes into account the structure of the subtrees of the node being processed. The translators introduced in this paper will usually be smaller than the ones constructed in [W], due to some reduction techniques used here (implicitly) which produce minimal LCA's (as opposed to the tree shaped evaluators in [W] which tend to be redundant). Because of this and due to their being near optimal, our translators will be by far more efficient.

COMPLEXITY ISSUES: The translators constructed in this paper all work in linear time w.r.t. the size of the parse tree, provided that one unit of time is charged for the evaluation of a semantic function, and assuming a random access memory. In fact, as was noted in [L&R&S1], one could always produce a linear-time evaluation strategy for each individual parse tree by analysing the dependencies in the tree at runtime; one would then construct the compound dependency graph of the tree and perform a topological sort on that graph. By the method presented here the dependency analysis is done once and for all for each grammar during the translator construction, thus saving us a considerable runtime overhead.

As for the time complexity of the translator construction, admittedly, it may grow exponentially with the size of the grammar. In fact, this is unavoidable in view of the inherent exponential complexity of the circularity problem [J&O&R], as our construction will also detect circularity. The size of the translators may also be exponential, as the set of characteristic graphs may be of exponential cardinality (which is precisely what accounts for the exponentiality of Knuth's circularity test).

A similar situation occurs with respect to parser construction. For instance, it is a well known fact that LR(k) parsers can have number of states which is exponential with the size of the grammar. Moreover, the problem of determining whether an arbitrary context-free grammar is LR(k) (with k unspecified) was shown to be NP-complete when k is expressed in unary (complete for non-deterministic exponential time when k is expressed in binary) [H&S&U].

Nevertheless, both the parser and the translator may be worth constructing once and for all for each attribute grammar, to be later on jointly used for the efficient implementation of the entire translation process. Furthermore, both can be generated automatically when an attribute grammar specifying a programming language and its translation is given as input to a compiler generating system.

REFERENCES:

[A&U1] Aho, A.V. and Ullman J.D. "Properties of Syntax Directed Translations". J. Computer Systems Sci., No. 3, pp. 319-364, (1965).

[A&U2] Aho, A.V. and Ullman, J.D. The Theory of Parsing, Translation, and Compiling, Vol. 2, Prentice-Hall, Englewood Cliffs, N.J. (1973).

[A&U3] Aho, A.V. and Ullman, J.D. "Translations on a Context Free Grammar". Inform. Contr. 19,5, pp. 439-475, (1971).

[B] Bochman, J.V. "Semantic Evaluation from Left to Right". Comm. of the Acut. Vol. 19, No.2, pp. 55-62, (1976).

[C&H] Cohen, R. and Harr, E. "Automatic Generation of Near-Optimal Linear-Time Translators for Non-Circular Attribute Grammars", Technical Report #120, Dept. of Computer Science, Technion, Israel, March 1978.

[D] Dreisbach, T.A. A Declarative Semantic Definition of PL360. UCLA-7289, Computer Science Dept. UCLA (1972).

[F] Fang, I. FOLDS, a Declarative Formal Language Definition System. STAN-72-329, Computer Science Dept., Stanford University (1972).

[G] Gerhart, S. "Correctness - Preserving Program Transformations". Proc. Second SIGACT-SIGPLAN Symp. on Principles of Programming Languages, Palo Alto, pp. 54-66 (1975).

[J1] Jazayeri, M. On Attribute Grammars and the Semantic Specification of Programming Languages. Ph.D. Thesis, Computer and Inf. Sci. Dept. Case Western Reserve University (1974).

[J2] Jazayeri, M. Live Variable Analysis, attribute Grammars, and Program Optimization: Draft, Dept. of Comp. Sci., University of N. Carolina, Chapel Hill, N.C. (1975).

[J&O&R] Jazayeri, M., Ogden, W.F. and Rounds, W.C. "The Intrinsically Dxponential Complexity of the Circularity Problem for Attribute Grammars". Comm. of the ACM, Vol. 18, No.12, pp. 697-706 (1975).

[K1] Knuth, D.E. "Semantics of Context Free Languages". Math. Systems Theory J., No. 2, pp. 127-145 (1968).

[K2] Knuth, D.E. "Semantics of Context Free Languages: Correction". Math. Systems Theory J., No. 5, p. 95 (1971).

[K3] Knuth, D.E. "Examples of Formal Semantics", Symp. on Semantics of Algorithm Lanuages. Lecture notes in Mathematics, Vo. 188, Springer-Verlag, New York (1971).

[K&W] Kennedy, K. and Warren, S.K. Automatic Generation of Efficient Evaluations for Attribute Grammars. Proc. of the 3rd ACM Symp. on POPL.

[L&R&S1] Lewis, P.M., Rosenkranz, D.J. and Stearns, R.E. "Attributed Translations". J. of Comp. and System Sciences, vol. 9, No. 3, pp. 279-307 (1974).

[L&R&S2] Lewis, P.M., Rosenkranz, D.J. and Stearns, R.E. Compiler Design Theory. Addison Wesley (1976).

[L&S] Lewis, P.M. and Stearns, R.E. "Syntax directed Translations", JACM, vol. 15, No. 3, pp. 654-683 (1968).

[N&A] Neel, D. and Armichahy, M. "Removal of Invariant Statements from Nested Loops in a Single Effective Compiler Pass". SIGPLAN Notices, vol. 10, No. 3, pp. 87-96 (1975).

REFERENCES (cont'd)

[P] Petrick, S.R. Semantic Interpretation in the REQUEST System. IBM Res. Report, RC-4457 (1973).

[W] Warren, S.K. The Coroutine Model of Attribute Grammary Evaluation. Ph.D. Thesis, Rice University, Houston, Texas (1976).

[H&S&U] Hunt III, H.B. Szymanski, T.G. and Ullman, J.D. "On the Complexity of LR(k) Testing". CACM, vol. 18, No. 12, pp. 707-726 (1975).