The intrinsically exponential complexity of the circularity problem for attribute grammars

Article *in* Communications of the ACM · December 1975

DOI: 10.1145/361227.361231

3 authors:

Mehdi Jazayeri
University of Lugano
164 PUBLICATIONS **4,176** CITATIONS
SEE PROFILE

William F. Ogden
The Ohio State University
43 PUBLICATIONS **890** CITATIONS
SEE PROFILE

William C. Rounds
University of Michigan
68 PUBLICATIONS **1,694** CITATIONS
SEE PROFILE

Programming Languages
B. Wegbreit Editor

The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars

Mehdi Jazayeri
University of North Carolina
William F. Ogden
Case Western Reserve University
William C. Rounds
University of Michigan

Attribute grammars are an extension of context-free grammars devised by Knuth as a mechanism for including the semantics of a context-free language with the syntax of the language. The circularity problem for a grammar is to determine whether the semantics for all possible sentences (programs) in fact will be well defined. It is proved that this problem is, in general, computationally intractable. Specifically, it is shown that any deterministic algorithm which solves the problem must for infinitely many cases use an exponential amount of time. An improved version of Knuth's circularity testing algorithm is also given, which actually solves the problem within exponential time.

Key Words and Phrases: attribute grammars, circularity problem, context-free grammars, computational complexity, exponential time, semantics

CR Categories: 4.12, 5.25

A version of this paper was presented at the Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., Jan. 20–22, 1975.

The work of the first author was partially supported by the Advanced Reseaich Projects Agency of the Department of Defense under contract DAAB03-70-C-0024.

Author's addresses: M. Jazayeri, Department of Computer Science, University of North Carolina, Chapel Hill, NC 27514; W.F. Ogden, Department of Computer Science, Case Western Reserve University, Cleveland, OH 44106; W.C. Rounds, Department of Computer and Communications Sciences, The University of Michigan, Ann Arbor, MI 48104.

1. Introduction

A fundamental activity of computer science is to study the intrinsic computational difficulty of problems posed for computer solution. Much recent effort has been devoted to finding "natural" examples of problems which can in principle be solved by machine, but which in actual practice are not amenable to machine solution because, no matter how they are programmed, they require too much time or space. Interest has focused on a broad class of problems known as the *NP*-complete problems, which includes (among many others) the classical traveling salesman problem, for which the best known algorithms all require exponential time. (See [1, Ch. 10].). However, no one has yet succeeded in proving that these problems are intrinsically of exponential complexity and hence generally intractable. Meyer [15], on the other hand, discovered a provably intractable "natural" problem in formal logic. His problem turned out, in fact, to be of much higher than exponential complexity. The circularity problem discussed here is a much simpler problem which arises in the area of semantics of programming languages. It is one of the first provably intractable problems which concern a system that is currently employed in practice.

Attribute grammars were introduced by Knuth [11] as a method for specifying the semantics of context-free languages. The *circularity problem* for a grammar is to determine whether or not the system of purported "definitions" given by the grammar avoids circularity in all possible instances. This problem is not entirely trivial. In [11a] Knuth eventually gave a correct algorithm to solve the problem; we show here that any deterministic algorithm which solves the problem must use an amount of time exponential in the description size of the grammar in question. Specifically, if we let n denote the size of the description of a grammar, there is a constant $c > 0$ such that any correct algorithm must run for $2^{cn/\log n}$ steps, for infinitely many n's. We also give an algorithm which actually solves the problem within time 2^{dn} for a specific $d > 0$. This improves the running time of Knuth's algorithm, which takes 2^{dn^2} steps in the worst case.

Attribute grammars have been used by investigators in fields such as language design (SIMULA [17], and PL360 [6]), formal semantics [10], and natural language recognition and question-answering systems [16]. They have been used as a tool in program optimization [15] and in the theory of program correctness [7]. There are also applications to the design of parser and compiler generators [3, 12]. These applications seem to support Knuth's original content that attribute grammars are a

Fig. 1. Example of an attribute grammar.

Start symbol: A	Terminals: $\{x, y, z\}$
Nonterminals: $\{A, B, C\}$	Attributes: synthesized: $\{a_1, b_1, c_1\}$ inherited: $\{a_2, b_2, c_2, c_3\}$

Productions and semantic rules:

1. $A \rightarrow BCz$
   $a_1 = b_1 + 5$, $c_2 = 26 * c_1$
   $b_2 = f(c_1)$, $c_3 = 56$
2. $B \rightarrow Bx$
   $(b_1)^0 = (b_1)^1$, $(b_2)^1 = (b_2)^0/124$
3. $B \rightarrow x$
   $b_1 = b_2$
4. $C \rightarrow y$
   $c_1 = 4$

Notes:
(i) f is a known function.
(ii) In the second production, superscripts are used to distinguish between the attributes of the left-hand B and the right-hand B.

natural, humanly understandable method of language definition.

The rest of this paper is in three principal sections. In Section 2 we define attribute grammars, give examples, and establish our lower bound. The proof involves a demonstration that the membership problem for writing pushdown acceptors [13], which has been shown to be an inherently exponential problem by Cook [5], can be efficiently imbedded into the attribute grammar circularity problem. Section 3 contains our circularity algorithm; the technique is to construct an ordinary context-free grammar from the given grammar, such that the constructed grammar is only exponentially larger than the original, and such that (in essence) the language generated by the context-free grammar is empty if and only if the original grammar was noncircular. The "emptiness problem" for context-free grammars can be solved in polynomial time; since a polynomial applied to an exponential is still just an exponential, we obtain our upper bound. Finally, in Section 4 we summarize our results and indicate some of their practical implications.

2. Attribute Grammars: Lower Bound

2.1 Definition and Examples

An attribute grammar is a context-free grammar with the following additions. (i) In each production, each occurrence of a nonterminal has a fixed finite set of *attributes* associated with that occurrence. A fixed subset of these attributes comprises the *inherited* attributes; the others are *synthesized.* (ii) Each production $X \rightarrow \beta$ has a fixed set of *semantic rules* associated with it. These rules specify how to evaluate the synthesized attributes of the left-hand nonterminal X, and the inherited attributes of any nonterminals occurring in β, in terms of the values of other attributes occurring in the production. A semantic rule can, in a given case, express any function of attributes appearing anywhere in the given production. One can use attributes to specify context-sensitive features of a language, or to compute "meanings" of expressions in the language generated by the underlying context-free grammar. We give an example of an attribute grammar in Figure 1.

In practical applications, these attributes could denote the data type of the nonterminal, its numerical value, its location in memory, or a variety of other quantities.

An attribute grammar is said to be well-formed if the attributes associated with nonterminals at any node of any derivation tree can be evaluated using the semantic rules for the productions which make up the tree. All interesting context-free grammars generate an infinite number of trees; the problem of well-formedness is not trivially solvable. For example, considering the definition of G in Figure 1, we see that a_1 cannot be evaluated in any derivation tree unless b_1 is first evaluated. In production (2), the attribute $(b_1)^0$ depends on $(b_1)^1$, in production (3), b_1 depends on b_2, and so forth. We can follow these dependencies for a while, but the information is more easily visualized if we use dependency graphs. A *dependency graph* for a production consists of a node for each attribute mentioned for the nonterminals in the production, and a directed arc from attribute a to attribute b iff a is used in the evaluation of b. Figure 2 shows the productions, semantic rules, and dependency graphs for grammar G.

The semantic rules provide for information flow through a tree; dependency graphs help represent this flow. The synthesized attributes carry information upward, and the inherited ones carry it downward. This fact is illustrated in Figure 3 where we have associated a dependency network with a derivation tree in G by piecing together the dependency graphs associated with individual productions.

By inspecting the dependency network in Figure 3, we can tell that attribute c_1 should be evaluated before any other attributes. Now suppose there had been an arrow from b_1 to b_2 at the second level in Figure 3.

Fig. 2.

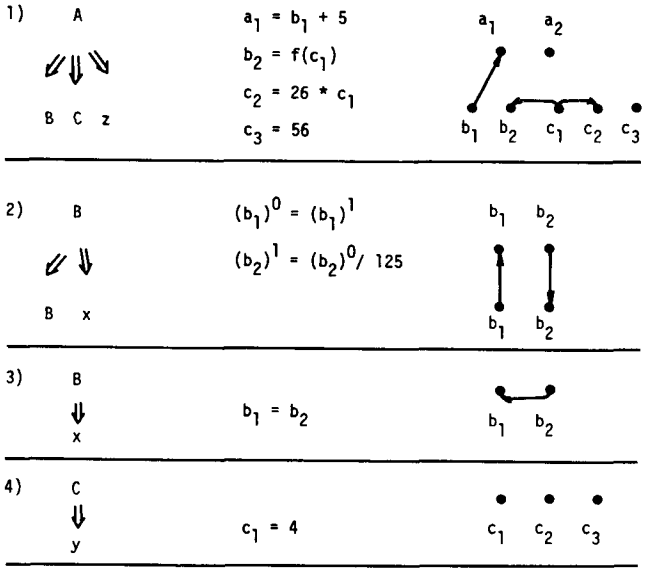


Fig. 3.

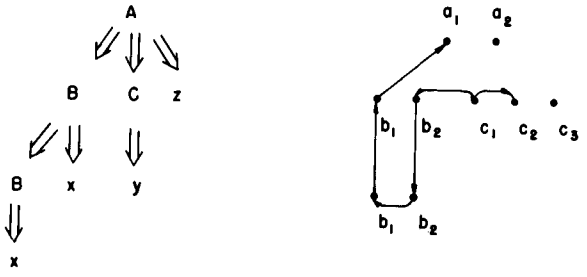
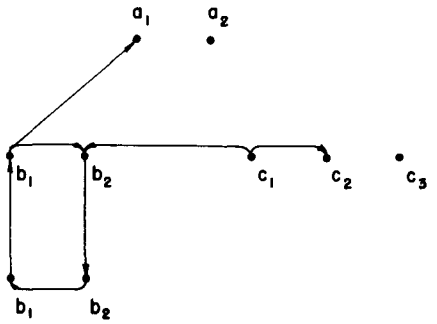


Fig. 4.



Clearly then neither b_1 nor b_2 could be evaluated; nor could a_1 for that matter. This fact is illustrated in Fig. 4.

Dependency networks exhibiting this property are called circular. An attribute grammar is circular if it can generate some circular dependency network. Circular attribute grammars are in some sense undesirable; one would like an algorithm to solve the circularity problem. In [11a] Knuth gives such an algorithm; this shows that the problem is decidable, but Knuth's algorithm in the worst case uses time 2^{dn^2}, where n is the description size of the grammar, and it appears that other investigators have felt that the problem is quite complex [18, p. 38]. We will now show that the problem is in fact intrinsically exponential.

2.2 Proof of Exponential Lower Bound

In this section we demonstrate how to simulate a writing pushdown acceptor (auxiliary pushdown automation with linear bounded input tape) using an attribute grammar. Since the membership problem for wpda's requires exponential time, we obtain a corresponding result for the circularity problem.

A writing pushdown acceptor [13] is a deterministic Turing machine with one work tape and an auxiliary pushdown stack. The input originally appears on the work tape, and the work head of the machine must never leave the squares on which the input appeared. The pushdown stack, however, may grow without bound. The membership problem for wpda's is the following. Given the description of a wpda and an input tape, determine if the wpda will accept the input (acceptance will be by final state and empty stack). We will transform the wpda membership problem to the circularity problem by giving a map G which takes a pair (M, α), where M is a wpda and α is an input tape, to an attribute grammar $G(M, \alpha)$ which will be circular iff M accepts α. The time required to construct $G(M, \alpha)$ will be $K_M \mid \alpha \mid \log \mid \alpha \mid$, where $\mid \alpha \mid$ is the length of α, and K_M depends only on M.

Given this construction, we obtain our lower bound by a standard complexity argument. Cook [5] has shown that for each $d > 0$ and each language L accepted by a Turing machine within time 2^{dn}, one can give a wpda accepting exactly L. Choose some $d_0 > 0$ and a language L accepted in time $2^d 0^n$ but not in time 2^{dn} for any $d < d_0$ (always possible by a well-known result of complexity theory; see [8, Theorem 10.11].). Let M be a wpda accepting L. For any $d < d_0$ there are infinitely many α such that any TM accepting L must run for more than $2^{d|\alpha|}$ steps; thus the procedure "form $G(M, \alpha)$; test $G(M, \alpha)$ for circularity" must take $2^{d|\alpha|}$ steps for infinitely many α. But since forming $G(M, \alpha)$ requires only $K_M \mid \alpha \mid \log \mid \alpha \mid$ steps, it follows by an easy inequality that testing an arbitrary grammar of size n for circularity must use at least $2^{dn/K_M \log N}$ steps (the grammar $G(M, \alpha)$ has size $n = K_M \mid \alpha \mid \log \mid \alpha \mid$). If we let $d = d_0/2$ and $c = d/K_M$, we can formulate our result in a concise way:

THEOREM 1. *There is a constant $c > 0$ such that any algorithm which tests an arbitrary attribute grammar of size n for circularity must run for more than $2^{cn/\log n}$ steps for infinitely many grammars.*

PROOF. By the preceding remarks, we must show how to construct $G(M, \alpha)$ in $O(\mid \alpha \mid \log \mid \alpha \mid)$ steps. We describe the construction in some detail, prove that the construction is correct, and finally analyze its complexity.

First we choose a notation for the wpda M. Moves are of two kinds: those which manipulate the work tape, and those which manipulate the stack.

A tape move is represented in the form $\delta(q, x, Z) = (q', D, y)$. Here, x and y are work tape symbols over some alphabet Σ, and Z is a stack symbol chosen from some alphabet Γ. The meaning is "in state q, scanning work tape symbol x and with Z as the top stack symbol, print symbol y on the work tape, move the head one square in direction D, and go to state q'." A stack move is either $\delta(q, x, Z) = (q', \text{PUSH } Z')$ or $\delta(q, x, Z) = (q', \text{POP})$. The first notation means "in state q, scanning x, with top stack symbol Z, add Z' to the top of the stack and go to state q', leaving the tape unchanged," and the second means "in the same situation, pop the top symbol from the stack." There is a special $Z_0 \in \Gamma$ which marks the bottom of the stack and which is never popped from the stack; other conventions will be stated as needed in the proof.

Each move rule of the wpda will give rise to certain productions in the attribute grammar; the initial wpda configuration will give rise to the starting productions of the grammar. Throughout the construction we assume $\Sigma = \{a, b\}$.

The nonterminal symbols of $G(M, \alpha)$ are quintuples $\langle q, x, r, y, Z\rangle$ where q and r are states; $x, y \in \Sigma$; and $Z \in \Gamma$. Associated with each nonterminal will be $6 * (2n - 1)$ attributes, where n now represents the length of $\alpha = x_1 \ldots x_n$, the input to the wpda. These attributes will be used to keep track of possible tape configurations throughout a wpda computation.

A derivation tree in the grammar $G = G(M, \alpha)$ represents a computation of the wpda in the same way that a context-free derivation tree represents a computation of an ordinary pushdown automaton. Push moves are represented by binary productions of the form $X \rightarrow YZ$; tape moves by unary productions $X \rightarrow Y$; and pop moves by terminal productions $X \rightarrow t$. At a given nonterminal node in the tree, we will use the attributes to represent two wpda tape configurations: (i) the "current" configuration, consisting of the head position and tape contents at the present time, and (ii) the "predicted"configuration consisting of these same quantities at the time just before the current top stack symbol is going to be popped from the stack. In the nonterminal $\langle q, x, r, y, Z\rangle$ used at this place in the tree, q represents the "current" state, x represents the "current" symbol scanned by the head, r represents the "predicted" state, y is the "predicted" scanned square, and Z is the current (and predicted!) symbol on top of the stack.

$3 * (2n - 1)$ attributes of the nonterminal symbol are used to represent the current tape configuration; the other $3 * (2n - 1)$ are used for the predicted configuration. Let C be the set of these current attributes for a nonterminal symbol and P the set of predicted attributes. C is partitioned into $2n - 1$ *cells* $C(1), \ldots, C(j), \ldots, C(2n - 1)$. Each cell contains three attributes which we name $C(j, a)$, $C(j, b)$ and $C(j, *)$. In a circular dependency network, the fact that $C(j, *)$ participates in a circuit will imply that cell $C(j)$ represents a current tape square. It will turn out that in this case, exactly one of $C(j, a)$ and $C(j, b)$ also participates in the circuit, so that the tape square will contain either an a or a b. At a nonterminal node, the active cells in a circuit will always form a block of n contiguous cells $C(p), \ldots, C(p + n - 1)$. Such a block must always include cell $C(n)$ because there are only $2n - 1$ cells in all. Cell $C(n)$ will thus always represent the position of the head relative to the beginning of the tape; a "move left" instruction will cause a "shift right" in the active cell block, so that $C(p + 1)$ becomes the first active cell, and $C(n)$ is one space closer to the first active cell. Similar remarks apply to the P attributes; a configuration can be obtained from an active block, but instructions will cause no changes in P attributes. (For an idea of the general form of a circuit corresponding to an accepting computation, examine Figure 10.)

In each cell $C(j)$, the inherited attributes are $C(j, a)$ and $C(j, b)$; $C(j, *)$ is synthesized. In the $P(j)$ cells, the $P(j, *)$ attribute is inherited, and the other two are synthesized. In a production $X^0 \rightarrow X^1X^2$ let C^0 and P^0 be the attributes associated with X^0, C^1 and P^1 with X^1, and C^2 and P^2 with X^2.

We are now ready to associate grammar productions to move rules of the wpda. A simple example will be used to illustrate the construction.

1. Let q_0 be the initial state of M. In the initial configuration of M the tape head is at square 1 of the tape; the tape contents are $\alpha = x_1 \ldots x_n$, and the top stack symbol is Z_0. We make up productions, corresponding to this initial configuration, of the form

$$\langle S \rightarrow \langle q_0, x_1, q_f, y, Z_0\rangle\rangle.$$

Here S is a special starting nonterminal, with no attributes; q_f can be any final state, and y is any element of Σ. The semantic rules are the same for each of these productions. They refer only to attributes of the right-hand nonterminal. The connections are as follows:

$$C(n + j + 1, x_{j+1}) = C(n + j, *), \quad \text{for } j = 1 \text{ until } n - 1;$$
$$C(n, x_1) = C(2n - 1, *);$$
$$P(j, *) = P(j, a) + P(j, b), \quad \text{for } j = 1 \text{ until } 2n - 1.$$

Note: $P(j, a) + P(j, b)$ indicates that both attributes are connected to $P(j, *)$. Example: Let q_4 be a final state, and let $\alpha = aab$. Then one production would be $S \rightarrow \langle q_0, a, q_4, b, Z_0\rangle$ with the dependency graph shown in Figure 5.

In Figure 5, we have numbered the P sets in reverse order from the C sets; this will make our graph planar in the rest of the examples. Notice how *aab* is selected by the C connections; the fact that the rightmost cells $C(3)$, $C(4)$, $C(5)$ are active indicates that the head position is at the left end of the tape.

2. Consider a tape move of the form $\delta(q, x, Z) =$

Fig. 5.

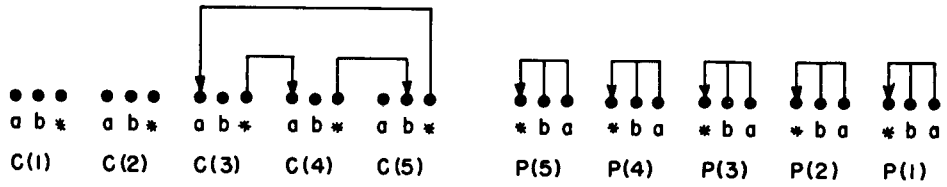


Fig. 6.

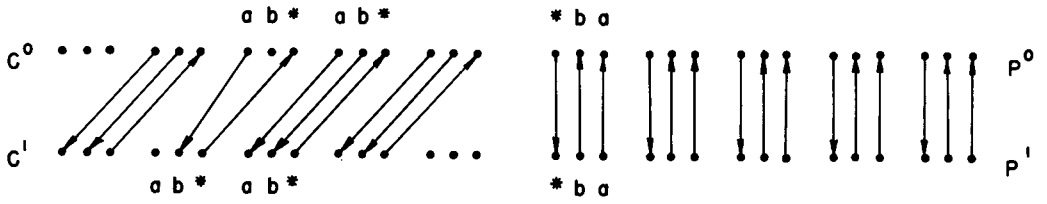
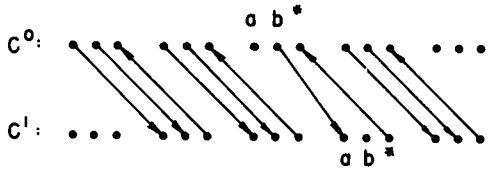


Fig. 7. (P attributes as previously.)



(q', right, y). This gives rise to productions of the form

$$\langle\langle q, x, r, w, Z\rangle \rightarrow \langle q', u, r, w, Z\rangle\rangle$$

for all $u, w \in \Sigma$, *and* $r \in Q$. The dependency graph is the same for each of these productions:

$$\begin{aligned}
C^1(j, v) &= C^0(j+1, v) \text{ for each } v \in \Sigma, \text{ and} \\
&\quad \text{for } j = 1, \ldots, n-2, n, \ldots, 2n-2; \\
C^1(n-1, y) &= C^0(n, x); \\
C^0(j+1, *) &= C^1(i, *), \text{ for } j = 1 \text{ until } 2n-2; \\
P^1(j, *) &= P^0(j, *), \text{ for } j = 1 \text{ until } 2n-1; \\
P^0(j, v) &= P^1(j, v), \\
&\quad \text{for } j = 1 \text{ until } 2n-1, \text{ and all } v \in \Sigma.
\end{aligned}$$

This is a "move right" instruction, so the active cells "shift left" to maintain the proper head versus tape position. This is illustrated by the next example, where the move rule is $\delta(q_0, a, Z_0) = (q_8, \text{right}, b)$. A possible production for this rule would be $\langle q_0, a, q_4, b, Z_0\rangle \rightarrow q_8, a, q_4, b, Z_0\rangle$, and the graph in Figure 6 illustrates the corresponding dependencies.

The linkage of $C^0(3, a)$ to $C^1(2, b)$ indicates that the square to the left of the scanned square now has b printed on it; the printing is not reflected by the nonterminal. If we had used the production

$$\langle\langle q_0, a, q_4, b, Z_0\rangle \rightarrow \langle q_8, b, q_4, b, Z_0\rangle\rangle$$

which (incorrectly) asserts that b is on the new scanned square, there would be a break in the circuit when the production corresponding to the next move is applied. The "predicted" configuration is not affected by this instruction; identity connections in the P sets reflect this fact.

3. Similarly, a "move left" instruction will give rise to a "shift right" dependency graph. We omit the formal description of production and semantic rules, but give an example: if $\delta(q_3, b, Z_1) = (q_6, \text{left}, a)$, we would have a production $\langle q_3, b, q_6, b, Z_1\rangle \rightarrow \langle q_6, b, q_6, b, Z_1\rangle$ and the graph in Figure 7.

In this and in further examples, q_6 will be a "pop the stack" move, so the "predicted" state and scanned symbol should be q_6 and b, which agree with the "current" q_6 and b. Again, incorrect selection of a production will either block the derivation or break a circuit.

4. Push moves of the form $\delta(q, x, Z) = (q', \text{PUSH } Z')$ give rise to binary productions as follows. For each $u, w, y \in \Sigma$; $s, t, r \in Q$, such that $\delta(s, u, Z') = (t, \text{POP})$, we have a production of the form

$$\langle\langle q, x, r, w, Z\rangle \rightarrow \langle q', x, s, u, Z'\rangle\langle t, u, r, w, Z\rangle\rangle$$

with associated dependency rules

$$\begin{aligned}
C^1(j, v) &= C^0(j, v); & P^0(j, v) &= P^2(j, v); \\
C^0(j, *) &= C^1(j, *); & C^2(j, v) &= P^1(j, v); \\
P^2(j, *) &= P^0(j, *); & P^1(j, *) &= C^2(j, *);
\end{aligned}$$

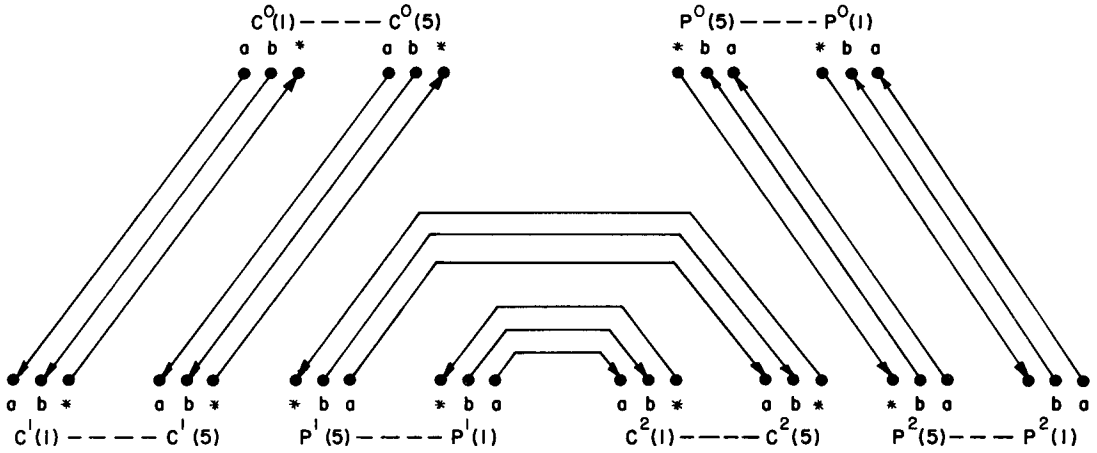
for each $v \in \Sigma$, and for all $j = 1, \ldots, 2n-1$. An example is the move $\delta(q_8, a, Z_0) = (q_{15}, \text{PUSH } Z_1)$ which gives rise to the production

$$\langle\langle q_8, a, q_4, b, Z_0\rangle \rightarrow \langle q_{15}, a, q_6, Z_1\rangle\langle q_9, b, q_4, b, Z_0\rangle\rangle$$

and the dependency graph shown in Figure 8. Assume $\delta(q_6, b, Z_1) = (q_9, \text{POP})$.

This graph reflects the fact that the tape configuration after the completion of the "pop" move q_6 should become the current configuration for the nonterminal $\langle q_9, b, q_4, b, Z_0\rangle$; when Z_1 is popped from the stack by q_6, the next state is q_9 and Z_0 reappears on the stack. The "predicted" tape configuration on the left branch is returned in the P^1 attributes, and the connection of

Fig. 8. (Attributes not shown are similarly connected.)



P^1 to C^2 attributes means that this configuration becomes the "current" configuration for state q_9.

5. Now we consider "pop" moves of the form $\delta(q, x, Z) = (q', \text{POP})$. Such a move gives rise to the single production

$\langle\langle q, x, q, x, Z\rangle \rightarrow t\rangle$

where t is a terminal symbol, and attribute connections for the nonterminal symbol $\langle q, x, q, x, Z\rangle$ of the form

$C^0(j, *) = P^0(j, *)$, all j;
$P^0(j, v) = C^0(j, v)$, all $j \neq n$, and $v \in \Sigma$;
$P^0(n, x) = C^0(n, x)$.

As an example, the rule $\delta(q_6, b, Z_1) = (q_9, \text{POP})$ gives us the production $\langle q_6, b, q_6, b, Z\rangle \rightarrow t$ with the connections shown in Figure 9.

The transition to state q_9 is not explicit in this production; it should have been correctly predicted at "split time" by the production which simulated pushing Z_1 on the stack. In part 4 we correctly selected that production.

6. Similarly, for each final state q_f, we have productions

$\langle\langle q_f, x, q_f, x, Z_0\rangle \rightarrow t\rangle$

for each $x \in \Sigma$; the attribute connections are the same as in part 5. In our example, q_4 is a final state, so a production might be $\langle q_4, b, q_4, b, Z_0\rangle \rightarrow t$.

Figure 10 shows the circuit associated with an entire accepting computation by the wpda using the move rules described in the foregoing examples. Again, the order of the P cells is reversed to make the graph planar. Only arcs which contribute to the circuit are shown explicitly.

This completes the construction. We must show that it is correct: $G(M, \alpha)$ is circular iff M accepts α. We do not give the complete correctness proof. It is relatively straightforward to show that if M accepts α then there is a circuit in $G(M, \alpha)$. We give details of the converse proof, which is more difficult.

Define two attributes of a nonterminal to be *adjacent* if they occur in consecutive cells $C(j)$ and $C(i + 1)$ (or in consecutive P cells.). A block of adjacent attributes is a set of m attributes chosen from cells $C(j), \ldots, C(j + m - 1)$. A block is *alphabetic* if each attribute is $C(j, a)$ or $C(j, b)$.

An alphabetic block of size n associated with a nonterminal $\langle q, x, r, y, Z\rangle$ is said to *represent* a *current configuration* of the wpda if the attributes are C attributes, and if the "scanned" attribute in $C(n)$ is x. [A block of n attributes, as we have said, must always include an attribute in $C(n)$.] Similarly, an alphabetic block of n P-attributes *represents* a *predicted configuration* if the attribute in $P(n)$ is y. The state and top stack symbol of a wpda configuration can be obtained from the nonterminal, and the head position from the relative position of $C(n)$ in the block.

A partial derivation tree in G is a tree whose root is some nonterminal $\langle q, x, r, y, Z\rangle$ and whose leaves are all terminal symbols. A dependency network can be associated with each such tree. The following properties are proved by induction on the height of partial derivation trees.

Property 1 (uniqueness and back (*) connectivity). In any partial tree, let $C(j, w)$ be an alphabetic attribute of the root nonterminal, which is connected by some network path to another attribute of the root nonterminal. Then this other attribute must be $P(k, u)$ for some k and u; and $P(k,u)$ is the unique attribute of the

Fig. 9.

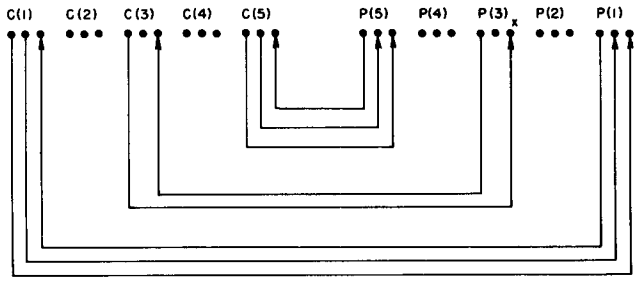
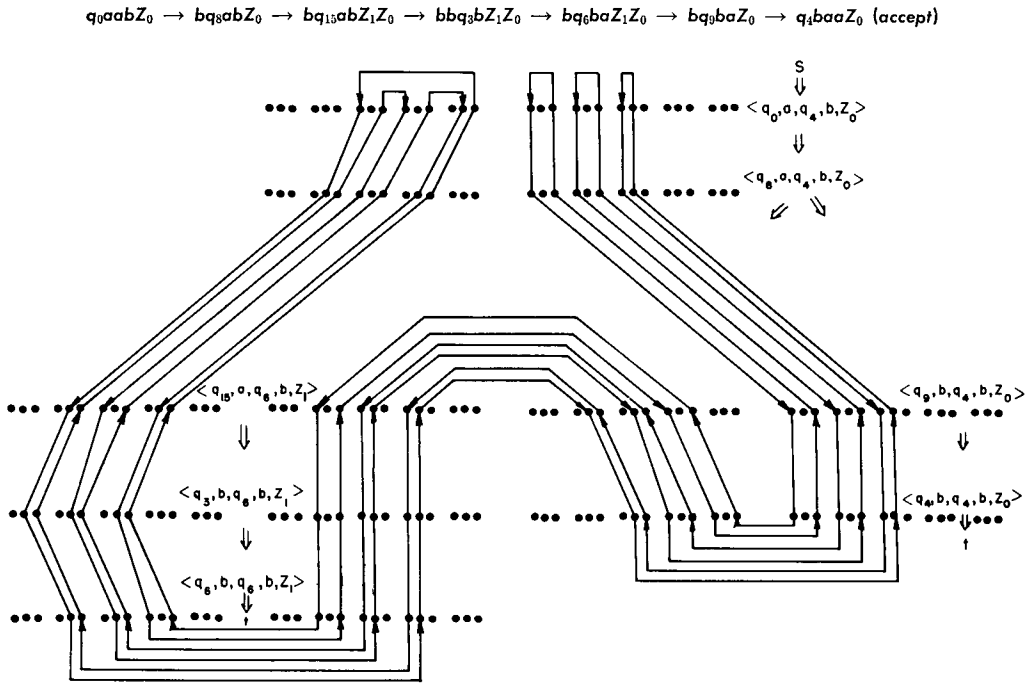


Fig. 10. Circular dependency network for an accepting computation.



root to which $C(j, w)$ is connected. Also, in this case, $P(k, *)$ is connected to $C(j,*)$ and to no other root attribute.

Property 2 (adjacency preservation). If two adjacent C attributes are connected as above to P attributes, then the P attributes are adjacent.

Both these properties are assertions about the form of dependency networks and do not refer to computations; they are easy to establish by induction. The next property gives us a fact about configurations.

Property 3. If a block of root attributes representing a current configuration is connected to a block of predicted root attributes, then that predicted block represents a predicted configuration. Furthermore, the current configuration will yield the predicted configuration, by means of the wpda moves, in such a way that for every intermediate configuration, the stack will extend or equal the current stack, and the stack will have the same contents in the predicted and current configurations.

Fig. 11. Size information for constructed grammar $G(M, \alpha)$.

Attribute grammar constituent set	Number of objects in this set	Length of description of each object
Nonterminals	$q^2 * s^2 * p$	$\log(q^2 * s^2 * p)$
Attributes per nonterminal	$2 * (s + 1) * (2n - 1)$	$\log(2 * (s + 1) * (2n - 1))$
Productions (without dependencies)	$\leq k * s^3 * q$	$\leq 3 \log(q^2 * s^2 * p)$
dependency graphs	$\leq k * s^3 * q$	$\leq 3 * 2 * (s + 1) * (2n - 1) * \log(2*(s + 1) * (2n - 1))$

(Notice that properties 1 and 2 guarantee that the hypothesis of property 3 is fulfilled in only one way.) Property 3 also follows by induction on partial derivation trees; the basis of such an induction is established by considering height 1 trees $X \rightarrow t$, and then proving the result for an arbitrary tree assuming it for all proper subtrees. We omit the details.

To complete the proof, we show that if the grammar has a dependency network with a circuit, then there is an accepting computation by the machine. For a loop to exist, an initial production $S \rightarrow \langle q_0 , x, q_f , y, Z_0\rangle$ must be involved, because only in this production are synthesized attributes connected to inherited ones. At least one C attribute, say $C(j, x_p)$, of the $\langle q_0 , x, q_f , y, Z_0\rangle$ nonterminal must participate in the circuit. $C(j, x_p)$ must be connected through the tree to some other root attribute, hence (by uniqueness in property 1) to $P(k, *)$ back $(*)$ connectivity implies that $P(k, *)$ is connected to $C(j + 1, x_{p+1})$. Repeating this argument shows that all C attributes linked together in the initial production must participate in the loop. This means that the initial configuration forms a block of size n; by property 3, the initial configuration yields a final configuration, as required.

Having established that the construction works as desired, we turn to the analysis of its complexity. Let n be the length of the tape α. We will show that the length of $G(M, \alpha)$ is bounded by $K_M n \log n$, where K_M depends only on the wpda M. The table in Figure 11 presents descriptive size information about the constructed grammar. In the table, q is the number of wpda states, p is the cardinality of Γ, s is the cardinality of Σ, n is the length of α, and k is the number of move rules of M.

The length of $G(M, \alpha)$ satisfies

$$\begin{aligned} |\, G(M, \alpha)\,| &\leq (\text{number of attributed productions}) \\ &\quad * (\text{length of each production}) \\ &\leq (ks^3 * q) * (3 \log (q^2 s^2 p) + 3 * 2(s+1) \\ &\quad * (2n-1) * \log (2(s+1)(2n-1))) \end{aligned}$$

Because k, s, q, and p depend only on the machine M, we have

$$|\, G(M, \alpha)\,| \leq K_M n \log n$$

as we wanted. Since $G(M, \alpha)$ can be constructed in time proportional to its length, Theorem 1 is proved.

3. Upper Bound on Complexity of the Circularity Problem

This section gives an algorithm to test an arbitrary attribute grammar for circularity. It is closely related to the algorithm of Knuth [11a] but it improves the time bound to 2^{dn}, where n is the grammar size.

In brief, our algorithm produces a context-free grammar G' of size $2^{d_1 n}$ from an attribute grammar G of size n. G' is then tested for the property "$L(G') = \varnothing$." This property can be determined in time $O(|\, G' \,|^3)$ so that the whole process takes time $O((2^{d_1 n})^3) = O(2^{dn})$. Technically, it is more convenient to perform an algorithm called "useless production removal" on G' instead of the emptiness test; the two algorithms, however, require essentially the same time [2, p. 144].

Here is a general outline of the method for constructing the context-free grammar G' from G.

1. Generate nonterminal symbols of G'. These are pairs $\langle X, D\rangle$ where X is a nonterminal of G, and D is a set of pairs of attribute names from G. The number of these symbols is small enough so that they can all be generated in exponential time.

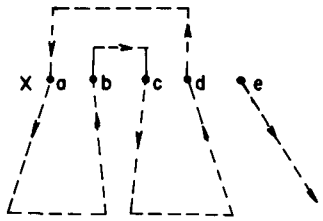
2. Using the generated nonterminal symbols, make up productions of G' corresponding to productions of G. This is done on a production-by-production basis, stepping through G productions. Again only exponential time is required.

3. Test each new production for admissibility; each new production must reflect in its $\langle X, D\rangle$ symbols certain connectivity relations in the dependency graph of its underlying G production. This test can be performed, for each new production, in time polynomial in $|\, G\,|$. Let G' consist of the productions found to be admissible.

4. Perform the useless production removal algorithm mentioned above, on G'. Test to see if certain productions remain in G'; if so output "circular"; else output "well-formed."

The improvement in our algorithm over Knuth's comes in parts 1 and 2. The observation which makes the algorithm possible is the following. If there is a circuit in some dependency network for grammar G, then there is a *simple* circuit: one which goes through each attribute node at most once. We only concern ourselves with finding a simple circuit; this reduces the number of D sets, which are sets of pairs, from a possible 2^{m^2} to $2^{m \log m}$, where there are m attributes mentioned in G. We illustrate the idea in Figure 12. Suppose X is a nonterminal node in some G derivation tree, and X has attributes which participate in a simple circuit in the dependency network. These attributes must occur in pairs as shown.

Fig. 12.



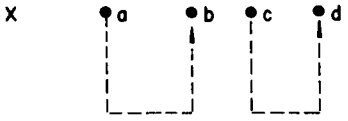
The portions of the loop which lie below X in the tree are abbreviated by the pairs (a, b) and (c, d); since the loop is simple, the pairs specify a 1-1 partial correspondence $a \rightarrow b$ and $c \rightarrow d$, between inherited and synthesized attributes. Such a set D specifying a partial 1-1 correspondence between attributes associated with a particular $X \in G$ will be called a *simple* set; the number of simple D's is at most $2^{cm \log m}$.

Our algorithm will now be specified in more detail; we analyze its complexity at the end of the section.

1. (Generate nonterminals.) Let P be a production of G, and let X be a nonterminal occurrence in P. Let $A(X, P)$ be the set of attributes associated with this X occurrence in P. For each P and occurrence of X in P, add the nonterminals $\langle X, D\rangle$ to G', where D is an arbitrary simple subset of all ordered pairs with elements in $A(X, P)$. The set $D = \varnothing$ is always allowed.

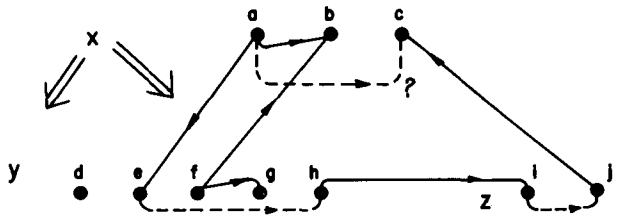
2. (Make up potential productions.) Let N' be the set of nonterminals just generated; let Σ be the terminals of G, let N be the nonterminals of G, and let h be the map from $N' \cup \Sigma$ to $N \cup \Sigma$ which sends $\langle X, D\rangle \in N'$ to X and $t \in \Sigma$ to itself. For each production P in G, add all productions $Y_0 \rightarrow Y_1 \ldots Y_k$ to G' such that $Y_i \in N' \cup \Sigma$, P is $h(Y_0) \rightarrow h(Y_1) \ldots h(Y_k)$, and such that if $Y_i = \langle X_i, D_i\rangle$, then all attributes mentioned in D_i are members of $A(X, P)$.

3. (Test for admissibility.) We intend an $\langle X, D\rangle$ symbol, where, for example, $D = \{(a, b), (c, d)\}$, to derive a terminal string in G' iff (i) X can derive a terminal string in G, and (ii) in the associated network, there is a simple path from a to b and c to d:



If $\langle X, D\rangle$ is the left side of some production, and $\langle X_i, D_i\rangle$ are the nonterminals on the right side, then we can test $\langle X, D\rangle$ for properties (i) and (ii) provided we

Fig. 13.



assume that the nonterminals $\langle X_i, D_i\rangle$ already have these two properties. Let us illustrate the test with several examples. Consider the production and dependency graph indicated by the solid arrows in Figure 13. The production

$$\langle\langle X, (a, c)\rangle \rightarrow \langle Y, (e, h)\rangle\langle Z, (i, j)\rangle\rangle$$

(illustrated by the intermittent arrows in Figure 13) and the production

$$\langle\langle X, (a, c)\rangle \rightarrow \langle Y, (d, f), (e, h)\rangle\langle Z, (i, j)\rangle\rangle$$

are both admissible because of a possible simple path $a \rightarrow e \rightarrow \cdots \rightarrow h \rightarrow i \rightarrow \cdots \rightarrow j \rightarrow c$. The production

$$\langle\langle X, (a, c)\rangle \rightarrow \langle Y, (e, f), (h, g)\rangle\langle Z, (i, j)\rangle\rangle$$

is admissible because of the path $a \rightarrow e \rightarrow \cdots \rightarrow f \rightarrow g \rightarrow \cdots \rightarrow h \rightarrow i \rightarrow \cdots j \rightarrow c$. Notice that the set $\{(e, f)(e, h)\}$ is not simple and not allowed as a D set. The production

$$\langle\langle X, (a, c)\rangle \rightarrow \langle Y, (e, f)\rangle\langle Z, (i, j)\rangle\rangle$$

is not admissible because the $g \rightarrow h$ connection is missing. The production

$$\langle X, (a, b)\rangle \rightarrow \langle Y, (d, h)\rangle\langle Z, \varnothing\rangle$$

is, however, admissible because of the direct connection $a \rightarrow b$.

To give the formal definition of admissibility we introduce some preliminary notation. Assume that no attributes are associated with terminal symbols. Let π be a production in G', let $u(\pi)$ be the underlying G production, and let $\gamma(u(\pi))$ be the associated dependency graph. Let e and f be attributes occurring in arbitrary nonterminals $\langle Y, E\rangle$ and $\langle Z, F\rangle$ in π; these nonterminals may be the same and may occur on the left side of production π. We say that e is π-related to f iff either (i) there is a local path from e to f in $\gamma(u(\pi))$ or else (ii) (e, f) is a pair in the set E (or the set F), and the nonterminal $\langle Y, E\rangle$ (or $\langle Z, F\rangle$) occurs on the right side of π. Let π^+ be the transitive closure of the π-relation. The production π with left hand side $\langle X, D\rangle$ is *admissible* if either $D_i' = \varnothing$ or for each pair (a, b) in D, a is π^+-related to b.

Part 3 of the algorithm requires us to check each π for admissibility and to eliminate all nonadmissible productions from G'.

It now follows by induction on partial derivation trees that $\langle X, D\rangle$ can derive a terminal string in the new G' iff (i) X can derive such a string in G and (ii) for each (a, b) in D there is a simple path from a to b in the dependency network.

Let the starting nonterminals of G' be of the form $\langle S, D\rangle$ where S is the starting nonterminal of G and $\langle S, D\rangle$ is the left side of some (admissible) production of G'.

4. (Remove useless productions.) To finish the algorithm we must identify those G' productions which could represent the "cap" of a loop in G. A production such as $\langle X(a, c)\rangle \rightarrow \langle Y(e, h)\rangle\langle Z(i, j)\rangle$ might be such a "cap" provided in the underlying dependency graph there were a local path from c to a. As another example, $\langle X, \varnothing\rangle \rightarrow \langle Y(g, h)\rangle\langle Z(i, j)\rangle$ would be a cap if there were connections $j \rightarrow g$ and $h \rightarrow i$ in the production; this would allow for a loop $g \rightarrow \cdots \rightarrow h \rightarrow i \cdots \rightarrow j \rightarrow g$ in a dependency network. The formal definition reflects these examples: a production π is a *cap* if there exists at least one attribute mentioned in π which is π^+-related to itself.

In order for G to be circular, some admissible G'-cap must be "reachable" from a start nonterminal of G' and that cap must derive a terminal string. Formally, a production π is *useless* in a grammar if for every complete derivation $S \Rightarrow^* w$, where $w \in \Sigma^*$, π is never used in that derivation. The final step in our algorithm is to remove all useless productions from G'. (For a description of this process, see [2, Ch. 2].) If any caps are left, then the grammar G is circular; otherwise G is well formed.

This completes the description of our algorithm, and now we consider its complexity.

We assume that all our attribute grammars are somehow described in a fixed (say binary) alphabet. This means that to name m attributes requires $m \log m$ space.

For the complexity analysis, we notice that part 1 of our algorithm can be performed simultaneously with part 2 since both parts are carried out on a production-by-production basis. We will therefore show that part 2 requires only exponential time. First we show that the descripton of grammar G itself must occupy a certain amount of space. Let P be a production of G; let k_P be the number of nonterminals in P, and let M_P be the number of distinct attributes mentioned in the dependency graph of P. The space S_P required in grammar G to write down production P and its dependency graph therefore satisfies

$$\langle S_P \geq k_P \log k_P + M_P \log M_P\rangle.$$

Next, let π_P be a production of G' whose underlying production is P. The length $l(\pi_P)$ of π_P satisfies

$$\langle l(\pi_P) \leq C + k_P \log k_P + 2M_P \log M_P\rangle$$

because the $\langle X_i, D_i\rangle$ pairs in π are constructed from the nonterminals X_i in G, and each D_i mentions at most $2M_P$ attributes. Each π_P can be constructed in time

proportional to its length and there are at most $2^{l(\pi_p)}$ productions π_P. The time T to construct all the π productions thus satisfies

$$\langle T \leq \sum_{P \in G} 2^{c_1 l(\pi_P)} \leq \sum_{P \in G} 2^{2c_1 S_P} \rangle$$

where C_1 is a fixed constant not depending on G.

But the description size n of G satisfies

$$\langle n \geq \sum_{P \in G} S_P \rangle$$

Therefore, since $S_P \geq 1$,

$$\langle T \leq \sum_{P \in G} 2^{2c_1 S_P} \leq 2^{2c_1 \Sigma S_P} \leq 2^{2c_1 n} \rangle$$

The admissibility test can be performed in time $q(n)$ for each production, where q is a polynomial not dependong on G. The time to construct G' is thus bounded by $q(n) * 2^{2c_1 n}$, which is still exponential, and since the useless production removal is a polynomial-time process, (see [2, page 146]) the whole algorithm takes time at most 2^{dn}. We summarize the arguments in Theorem 2.

THEOREM 2. *There is a constant $d > 0$ such that an arbitrary attribute grammar of size n can be tested for circularity in time bounded by 2^{dn}.*

Putting Theorems 1 and 2 together, we have the main result of the paper: the time $T(n)$ necessary to solve the circularity problem satisfies

$$\langle 2^{cn/\log n} \leq T(n) \leq 2^{dn} \rangle$$

where c and d are fixed positive numbers.

4. Conclusions

We have established that the circularity problem for attribute grammars is of intrinsically exponential complexity. Such problems are generally regarded as intractable; no deterministic algorithm can solve the problem within practical time limits. (It is an open question of automata theory whether or not a polynomial space bound is sufficient for the problem.) If we wanted to build a compiler generator for a class of attribute grammars defining programming languages, which would reject circular attribute grammars, then we should be willing to impose restrictions on the way attributes are defined in the grammars. Such restrictions have been suggested previously [3, 4, 12]. Some less restrictive conditions are given in [9] and will be discussed in a future paper.

Some theoretical questions are also suggested by the results in this paper. Perhaps the most tantalizing is: can the log n factor in the lower bound be removed? We conjecture that it can be.

Acknowledgments. We would like to thank K.G. Walter for stimulating our interest in attribute grammars, H.B. Hunt for sharing many insights into computational complexity, and V. Pratt for his comments on earlier versions of our proofs.

References

1. Aho, A.V., Hopcroft, J., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. Aho, A.V., and Ullman, J.D. *The Theory of Parsing, Translation and Compiling, Vol. 1.* Prentice-Hall, Englewood Cliffs, N.J., 1972.
3. Aho, A.V., and Ullman, J.D. Translations on a context-free grammar. *Inform. Contr. 19,* 5 (Dec. 1971), 439–475.
4. Bochmann, G.V. Semantics evaluated from left to right. *Comm. ACM.* To appear.
5. Cook, S.A. Characterization of pushdown machines in terms of time-bounded computers. *J. ACM 18,* 1 (Jan. 1971), 4–18.
6. Dreisbach, T.W. A declarative semantic definition of PL360. UCLA-ENG-7289, Computer Languages Group, Computer Sci. Dep., UCLA, 1972.
7. Gerhart, S. Correctness-preserving program transformations. Proc. Second SIGACT-SIGPLAN Symp. on Principles of Programming Languages, Palo Alto, Jan. 1975, pp. 54–66
8. Hopcroft, J., and Ullman, J.D. *Formal Languages and their Relation to Automata.* Addison-Wesley, Reading, Mass. 1969.
9. Jazayeri, M. On attribute grammars and the semantic specification of programming languages. Ph.D. Th., Case Western Reserve U., 1974. (Jennings Computing Center Rep. 1159).
10. Knuth, D.E. Examples of formal semantics. *Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics, Vol. 188,* Springer-Verlag, New York, 1971.
11. Knuth, D.E. Semantics of context free languages. *Math. Syst. Th. 2* (1968), pp. 127–145; correction to article in *Math. Syst. Th. 5* (1971), p. 95.
12. Lewis, P.M., Rosenkrantz, D. and Stearns, R.E. Attributed translations. Proc. Fifth Ann. ACM Symp. on Theory of Computing, Austin, Tex., 1973, pp. 160–171.
13. Mager, G. Writing pushdown acceptors. *J. Comp. Syst. Sci.* 3 (1969), 276–318.
14. Meyer, A.R. Weak monadic second order theory of successor is not elementary recursive. Project MAC Rep., MIT, 1972.
15. Neel, D. and Amirchahy, M. Semantic attributes and improvement of generated code. Proc. ACM Nat. Comput. Conf., San Diego, 1974, pp. 1–10.
16. Petrick, S.R. Semantic interpretation in the REQUEST system. IBM Res. Rep. RC 4457, July, 1973.
17. Wilner, W.T. Declarative semantic definition. Rep. STAN-CS-233-71, Computer Sci. Dep., Stanford U., 1971.
18. Wilner, W.T. Formal semantic definition using synthesized and inherited attributes. In *Formal Semantics of Programming Languages, Courant Computer Science Symposium 2,* R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972.