ON ATTRIBUTE GRAMMARS WITHOUT ATTRIBUTE SYNTHESIS

Martha SIDERIS

Computer Technology Institute, Patras, Greece

Communicated by H.R. Wiethe
Received 25 April 1989

We characterize precisely the power of attribute grammars when all functions are identities (such attribute grammars are known to simulate functionless logic programs). In the general case, the problem of deciding whether a given grammar generates a given string is complete for exponential time. Even in the absence of recursion, the problem is complete for polynomial space. In grammars with bounded “width”, the problem is complete for polynomial time.

Keywords: Attribute grammar, complexity theory

1. Introduction

Attribute grammars have been proposed by Knuth [7] for the formal specification of programming languages. An attribute grammar is basically a context-free grammar, augmented by attributes associated with the nonterminal symbols, functions defining values of attributes, and conditions über attributes. An attribute grammar generates the language consisting of all strings which have a parse tree in which all semantic conditions are satisfied.

Attribute grammars have been used extensively in a wide range of applications, including compiler writing systems, natural-language parsing, pattern recognition, and others. A recent application of attribute grammars is in logic programming. In [9,2] constructions are given, which transform logic programs into attribute grammars and vice-versa. The construction of an attribute grammar from a logic program can be briefly described as follows: A nonterminal symbol of the attribute grammar corresponds to a predicate symbol of the logic program. Each nonterminal has a number of attributes equal to the arity of the corresponding predicate symbol; thus the attribute values of a nonterminal correspond to terms. The attribute grammar contains no terminal symbols, therefore the only string generated by it is the empty string. A production of the attribute grammar corresponds to a clause of the logic program. The semantic rules and conditions of each production are constructed in such a way that there is a one-to-one correspondence between the parse trees of the attribute grammar (that is, parse trees of the underlying context-free grammar with the attribute values of nodes computed according to the semantics of the corresponding production) and the proof trees of the corresponding logic program. In this way the attribute evaluation process of the attribute grammar simulates the unification process of the corresponding logic program. It is easy to see that the only semantic function needed for the simulation is the identity function $\phi(x) = x$.

Recent research has revealed some interesting and unexpected connections between attribute grammars and complexity theory. There are three natural computational problems suggested by attribute grammars:

1. First, given an attribute grammar and a string, is the string generated by the attribute grammar? We call this the generation problem; it is the subject of this paper.
(2) The second problem is related to the first, only with the grammar fixed: The question is, what is the class of languages generated by attribute grammars? We call this the characterization problem.

(3) A third problem arises in the important (e.g., for compiler applications) special case in which attribute values are strings, and the only operation allowed in attribute synthesis is concatenation. Thus, we can think that the parse tree translates the string parsed into the string produced as the attribute value of the root. The question is, what are the languages that are the ranges of such translations? In [4] it was shown that this latter class of languages is equal to the class of languages log-tape reducible to context-free languages.

For general attribute grammars, without restrictions on the complexity of the predicates and attribute functions involved, it is easy to see that the generation problem is undecidable and the characterization problem trivial, since arbitrary languages can be generated by attribute grammars, using powerful enough predicates and functions. In [3] we studied the characterization problem for an appropriately restricted class of attribute grammars. We showed that the class of languages generated by attribute grammars is closely related to the class NP of languages accepted in polynomial time by nondeterministic Turing machines.

In this paper we shall consider the generation problem (1) above for attribute grammars of a special type, which we call attribute grammars without attribute synthesis: (AGWAS): All semantic functions are projections \((\phi(x_1, \ldots, x_n) = x_i)\) or nullary functions (i.e., functions with no arguments), and all conditions are Boolean combinations of terms of the form \(x_i = x_j\). This new restriction on attribute grammars is a fairly natural one, and is motivated by the logic programming application mentioned earlier. Notice that, for this class of attribute grammars, problem (3) above is meaningless, and problem (2) is trivial, since any language generated by an AGWAS is context-free (see the proof of Theorem 1).

In this paper we give precise upper and lower bounds on the complexity of the generation problem for AGWAS: "Given an AGWAS \(AG\) and a string \(x\), is \(x\) in the language generated by \(AG\)?". For various subclasses of AGWAS, our results are as follows:

(a) General AGWAS: The problem is \(\text{EXPTIME}\)-complete (Theorem 1).

(b) AGWAS in which the underlying context-free grammar does not have recursive productions (this corresponds to logic programs without recursion): The problem is \(\text{PSPACE}\)-complete (Theorem 2).

(c) AGWAS which have a bounded number of attributes per nonterminal, and in which each production of the underlying context-free grammar has a relatively short right-hand side (this corresponds to logic programs with a bounded body): The problem is \(\text{P}\)-complete (Corollary to Theorem 1).

Our lower bounds hold even in the special case in which the input string is the empty string (this latter case is of interest because of the logic programming connection mentioned above).

2. Definitions

An attribute grammar is a tuple \(AG = (G, A, d, \sigma)\), specified as follows:

(a) \(G = (N, T, R, S)\) is a context-free grammar with nonterminal set \(N\), terminal alphabet \(T\), set of productions \(R\) and start symbol \(S\).

(b) \(A\) is the set of attributes. For each nonterminal \(X\) of \(G\) we have two sets of attributes; \(\text{attr}_i(X)\) is the set of inherited attributes of \(X\) and \(\text{attr}_s(X)\) is the set of synthesized attributes of \(X\). The sets of inherited and synthesized attributes of a nonterminal are disjoint and their union over all nonterminals of \(G\) constitutes the set \(A\).

(c) \(d\) is a function assigning to each attribute \(a\) of \(A\) a domain \(d(a)\).

(d) \(\sigma\), the semantic part of \(AG\), assigns to each production \(r = X_0 \rightarrow x_0 X_1 \cdots x_{k-1} X_k x_k\) in \(R\) (where the \(X\)'s are nonterminals and the \(x\)'s are strings of terminals of \(G\)) a finite set of semantic rules. Each semantic rule is of the form
a_0 \leftarrow \phi(a_1, \ldots, a_m), \text{ where } \phi \text{ is a function and each of the } a_i \text{'s is an attribute of the nonterminals } X_j \text{ except that } a_0 \text{ may be a Boolean variable in the case that the semantic rule is a predicate. The set of semantic rules } \sigma(r) \text{ of production } r \text{ will contain}

(i) one predicate (without loss of generality);
(ii) one semantic rule for each synthesized attribute of the nonterminal symbol on the left-hand side of the production;
(iii) one semantic rule for each inherited attribute of the nonterminal symbols on the right-hand side of the production.

The attribute grammars considered in this paper have the following form: All semantic functions are projections (that is \( \phi(x_1, \ldots, x_n) = x_i \)) or nullary functions (i.e., functions with no arguments), and all predicates are Boolean combinations of terms of the form \( x_i = x_j \). We call such attribute grammars attribute grammars without attribute synthesis (AGWAS).

A parse tree for \( AG \) is a parse tree of \( G \) with the attributes of all internal nodes computed according to the semantics of the corresponding production and such that, for each application of production \( r \), the corresponding predicate evaluates to true. A string \( x \in L(G) \) is a sentence of the language generated by \( AG \), denoted \( L(AG) \), iff there is a parse tree of \( AG \) corresponding to a derivation of \( x \).

The problem we shall consider is the following: Given an attribute grammar \( AG \) without attribute synthesis and a string \( x \in \Sigma^* \) does \( AG \) generate \( x \)? We shall relate several aspects of this problem with certain well-known complexity classes [6], namely P (class of languages accepted by polynomial-time bounded Turing machines), PSPACE (class of languages accepted by polynomial-space bounded Turing machines) and EXPTIME (class of languages accepted by Turing machines operating within time bound \( 2^n \) for some \( c > 0 \)).

An alternating Turing machine [1] (ATM) is a Turing machine in which there are two kinds of states, the universal states and the existential states. Such a machine accepts, when starting from an existential state, if there is an accepting next configuration; from a universal state, if all next configuration are accepting. It was shown [6] that the class of languages accepted by polynomial-space bounded ATMs is equal to EXPTIME and the class of languages accepted by polynomial-time bounded ATMs is equal to PSPACE. In particular, it is known that the following languages are EXPTIME-complete and PSPACE-complete respectively:

\[ L_1 = \{ (M, x) : \text{ATM } M \text{ accepts input } x \text{ within space } |x| \}, \]
\[ L_2 = \{ (M, x) : \text{ATM } M \text{ accepts input } x \text{ within time } |x| \}. \]

3. The complexity of the generation problem

In this section we prove the following result.

Theorem 1. The problem of deciding whether an attribute grammar without attribute synthesis generates a string \( x \) is EXPTIME-complete.

Proof. We first give an exponential-time algorithm for the problem. Given an attribute grammar \( AG \) without attribute synthesis, we can construct a context-free grammar \( G = (N, \Sigma, R, S) \) such that \( AG \) generates a string \( x \) iff \( G \) does. For each nonterminal \( A \) of \( AG \) with attributes \( a_1, \ldots, a_k \) and for each \( k \)-tuple of constants appearing in \( AG \), \( G \) has a new nonterminal \( A(c_1, \ldots, c_k) \). For each rule \( A \rightarrow B_1 \ldots B_m \), \( m \geq 0 \), and for each combination of attribute values \( c_1, \ldots, c_k \) for \( A, c_1^1 \ldots c_k^1 \) for \( B_1, \ldots, c_1^m, \ldots, c_k^m \) for \( B_m \) satisfying the functions and predicate associated with the rule, \( G \) contains the rule

\[ A(c_1, \ldots, c_k) \]
\[ \rightarrow B_1(c_1^1, \ldots, c_k^1), \ldots, B_m(c_1^m, \ldots, c_k^m). \]

It is immediate that \( AG \) generates \( x \) iff \( G \) does. We next test whether \( G \) generates \( x \). This can be done in time \( O(gx^3) \), where \( g \) is the length of the description of \( G \) [5]. Now the length of the description of \( G \) is at most \( n^{n^2+2} \), where \( n \) is the description of \( AG \), since for each rule of \( AG \) \( (n \text{ rules at most}) \) we construct at most \( n^n \) rules of \( G \).
(we have \( < n \) choices of constants for attributes each among \( < n \) possibilities) and each of the new rules has length at most \( n \). The upper bound follows.

For the lower bound (completeness) we shall reduce the EXPTIME-complete language \( L_1 \) (see Section 2) to the generation problem for attribute grammars without attribute synthesis. In particular, given an ATM \( M \) and an input string \( x \) we show how to construct an attribute grammar \( AG(M, x) \) such that \( M \) accepts \( x \) in place if \( AG(M, x) \) generates the empty string.

\( AG(M, x) \) contains a nonterminal for each state of \( M \), and a starting nonterminal \( S \). Each nonterminal, besides \( S \), has \( 2n \) attributes, where \( n = |x| \). One of those attributes (called \( s \)) is synthesized and takes the value \( true \) if the state corresponding to the nonterminal is accepting and \( false \) otherwise. The remaining \( 2n - 1 \) attributes (called \( c_1, c_2, \ldots, c_{2n-1} \)) are inherited and represent the contents of the tape and the position of the head as follows: The value of attribute \( c_1 \) represents the scanned symbol. The values of attributes \( c_{i_1}, \ldots, c_{i_{1-1}} \) represent the contents of the tape squares to the left of the head position, while the values of attributes \( c_{i_{n+1}}, \ldots, c_{i_{2n+1}} \) represent the contents of the tape squares to the right of the head position.

A right move of \( M \) is simulated by shifting left the attribute values (that is, \( c_i \leftarrow c_{i+1} \) for \( 1 \leq i \leq 2n - 2 \)) and a left move of \( M \) by shifting right the attribute values (that is, \( c_i \leftarrow c_{i-1} \) for \( 2 \leq i \leq 2n - 1 \)). The value of the attribute representing the scanned symbol is changed according to the corresponding value of the transition relation \( \delta \). The attributes \( c_1, \ldots, c_{n-1} \) take as initial value the blank symbol \# , while the attributes \( c_n, \ldots, c_{2n-1} \) take the corresponding elements of the input string.

The starting production of \( AG(M, x) \), \( S \rightarrow Q_0 \), corresponds to the starting configuration of \( M \). In the semantic rules of this production the values of all inherited attributes of \( Q_0 \) are initialized in the way described above. A production of the form \( Q \rightarrow e \) is constructed for each nonterminal corresponding to an accepting or rejecting state. In the semantics of this production the synthesized attribute of the left-hand nonterminal takes the value \( true \) if the corresponding state is accepting and \( false \) otherwise.

A production of \( AG(M, x) \) is constructed for each rule of the transition relation \( \delta \). This production is of the form \( Q \rightarrow Q_1Q_2 \ldots Q_m \) if the state on the left-hand side of the \( \delta \)-rule is universal and of the form \( Q \rightarrow Q_1 | Q_2 | \ldots | Q_m \) if it is existential. The semantic rules of each production guarantee that

(a) the production is applied if the scanned symbol is equal to the symbol on the left-hand side of the \( \delta \)-rule;
(b) all inherited attributes of all right-hand side nonterminals represent the tape contents and position of the head after the corresponding move of \( M \) is performed;
(c) the synthesized attribute of the right-hand side nonterminal has the value \( true \) if the corresponding state is universal and all its successors are accepting or if it is existential and at least one of its successors is accepting.

In particular, for each rule of the transition relation \( \delta \)

\[ \delta(q, \sigma) = \{(q_1, \sigma_1, D_1), \ldots, (q_m, \sigma_m, D_m)\} \]

where the \( q \)-s are states, the \( \sigma \)-s are tape symbols, and \( D \in \{left, right\} \), the corresponding production has the form

1. \( Q \rightarrow Q_1Q_2 \ldots Q_m \) if \( q \) is a universal state,
2. \( Q \rightarrow Q_1 | Q_2 | \ldots | Q_m \) if \( q \) is an existential state.

The semantic rules of all productions of the form (1) and (2) contain the predicate \( Q.c_{|x|} = \sigma \) (the notation \( Q.c_{|x|} \) is used to indicate that attribute \( c_{|x|} \) belongs to nonterminal \( Q \)). The value of the synthesized attribute of the left-hand nonterminal is given by the semantic rule

\[ Q.s \leftarrow Q_1.s \land Q_2.s \land \cdots \land Q_m.s \]

for productions of type (1) and by the semantic rule

\[ Q.s \leftarrow Q_i.s \quad (i = 1, \ldots, m) \]

for each alternative of productions of type (2).

The semantic rules for the inherited attributes of right-hand side nonterminals depend on the
value of the corresponding $D_i$. If the value of $D_i = \text{left}$, the semantic rules have the form $Q_i.c_j \leftarrow Q_i.c_{j-1}$ where $i = 1, \ldots, m$ and $j = 2, \ldots, \lfloor |x| \rfloor, (\lfloor |x| + 2 \rfloor, \ldots, (2 |x| - 1))$, and $Q_i.c_{|x|+1} \leftarrow \sigma_i$. If the value of $D_i = \text{right}$, the semantic rules have the form $Q_i.c_j \leftarrow Q_i.c_{j+1}$ where $j = 1, \ldots, \lfloor |x| - 2 \rfloor, |x|, \ldots, 2 |x| - 2$ and $Q_i.c_{|x|-1} \leftarrow \sigma_i$.

It is easy to see that the attribute grammar so constructed is an attribute grammar without attribute synthesis of length equal to $c |x|$, where $c$ depends on the length of $M$, and that it generates the empty string iff input $x$ is accepted by $M$ in place.

If, as is usually the case, each rule has a relatively short right-hand side and each nonterminal has a small number of attributes, the complexity of the generation problem is reduced substantially and the problem is in $P$.

**Corollary.** The generation problem for attribute grammars without attribute synthesis in which each rule has a right-hand side of length at most $k$ and each nonterminal has at most $l$ attributes is in $P$.

**Proof.** The calculation for the upper bound in the proof of Theorem 1 yields the time bound $O(n^{(k+1)/l})$.

In some sense, the corollary is the best possible result, because of the following fact: One can show that the generation problem for context-free grammars (with no attributes) with at most two symbols at the right-hand side of each rule is $P$-complete. In proof, we reduce the **circuit value problem** [6] to it. We are given a Boolean circuit of **or** and **and** gates and its inputs, and we are asked if a particular gate, the **output** gate is **true**. From this we can construct a context-free grammar $G$ as follows: For each gate $g$ we have a nonterminal. If $g$ is an **or** gate, $g = f \lor h$, we have in $G$ the rules $g \rightarrow f \mid h$. If $g$ is an **and** gate, $g = f \land h$, we have the rule $g \rightarrow fg$. Finally, if $g$ is a **true** input, we have the rule $g \rightarrow e$. Our initial nonterminal is the output gate. It is clear that $G$ generates the empty string iff the value of the output gate is **true**.

**4. The case of no recursion**

When the underlying grammar has no recursion (that is, there is no derivation of the form $A \rightarrow^* \alpha A \beta$), the complexity of the generation problem is reduced to polynomial space. Note that, in the logic programming application mentioned in the introduction, the case of no recursion corresponds to logic programs that are equivalent to relational database queries [10].

**Theorem 2.** The generation problem for attribute grammars without attribute synthesis which have no recursion is $\text{PSPACE}$-complete.

**Proof.** For the upper bound we give a non deterministic parsing algorithm, which decides if a string is generated by an attribute grammar without attribute synthesis using polynomial space. The algorithm is a top-down, left-to-right parser, which stores the path from the root of the tree to the node being parsed (a path linear in the length of the attribute grammar since there is no recursion). At each step the algorithm

(a) guesses the grammar symbol corresponding to the next node of the tree along with a combination of all its attribute values and the production number according to which it will be analyzed and pushes the symbol onto the stack;

(b) if the symbol is a terminal, it is matched with the next input string symbol; if the symbol is a nonterminal, the algorithm checks if the attribute values are compatible with the semantic rules of the production according to which the nonterminal will be analyzed;

(c) when all successors of a nonterminal have been analyzed, the nonterminal is popped from the stack. The algorithm terminates when the last nonterminal is popped.

The size of the stack is equal to the depth of the tree, which is at most equal to the number of nonterminals in the case of no recursion. The only problem is that some attribute values needed for the computation of the semantic functions and predicates may belong to nonterminals that are not in the path from the root of the tree to the node being parsed (e.g., values of attributes that belong to successors or brothers of the current
node). We can solve this problem as follows: We transform the attribute grammar into a new one in which the left-hand side nonterminal of every production has a new attribute for every attribute belonging to a nonterminal of the right-hand side. All semantic rules are transformed in such a way that only attributes of the left-hand side nonterminal appear in the rule. Copy rules are added ensuring that the corresponding attributes of the left and right-hand side nonterminals have the same value. (Those new semantic rules copy the values of inherited attributes from the left-hand side of the production to the right and the values of synthesized attributes from right to left). With this modification, the top-down parser described above works.

For the lower bound we shall reduce the PSPACE-complete language $L_2$ (see Section 2) to the generation problem for attribute grammars without attribute synthesis in which the underlying grammar has no recursion. In particular, given an ATM $M$ and an input string $x$, we show how to construct an attribute grammar $AG(M, x)$ such that $M$ accepts $x$ within time $|x|$ iff $AG(M, x)$ generates the empty string.

The reduction is similar with the one described in the previous section. The only problem stems from the fact that the same state may be repeated in a computation of the ATM, while the same nonterminal is not allowed to participate twice in a derivation. To solve this problem, we include a nonterminal for each state and each step of the ATM (we can do that in polynomial time since the total number of steps is equal to the length of the parsed string). We therefore use $|x|$ nonterminals to represent a state, while in the previous construction we used only one. Each nonterminal $Q_t$ represents a state $Q$ at a specific time $t$, $t = 1, \ldots, |x|$, of the computation. We also construct productions for each rule of the transition relation $\delta$. The productions corresponding to the rule

$$\delta(q, \sigma) = \{(q_1, \sigma_1, D_1), \ldots, (q_m, \sigma_m, D_m)\}$$

have the form described in the previous construction, that is

$$Q_t \rightarrow Q_{1,t+1} Q_{2,t+1} \cdots Q_{m,t+1}$$

if $q$ is a universal state;

$$Q_t \rightarrow Q_{1,t+1} | Q_{2,t+1} | \cdots | Q_{m,t+1}$$

if $q$ is a universal state, $t = 0, 1, \ldots, |x| - 1$. The semantic rules of those productions are exactly the same with those described in the previous section. We also construct, as before, empty productions for the nonterminals corresponding to accepting and rejecting states.

It is clear that the constructed attribute grammar has an underlying context-free grammar with no recursion and generates the empty string iff ATM $M$ accepts input $x$ within time $|x|$. □

5. Discussion

The three results of this paper (Theorem 1, Theorem 2, and the Corollary) shed a new light on the power of attribute grammars as general models of computation. Perhaps the most surprising results are on the one hand the surprising high computational power of the attribute grammar formalism (Theorem 1), and on the other hand the relative little loss of computational power (from exponential time to polynomial space) that accompanies the loss of recursion in the grammar (compare Theorems 1 and 2). Theorem 1 no doubt corresponds to the well-known fact [8] that functionless logic programs can express all queries computable in polynomial time (with the additional exponential due to the fact, intuitively, that the program grammar is a part of the input, and the upper bound is generalized to an arbitrary string $x$, instead of the empty string). Notice that the proof of this stronger result is new and rather simple.

It would be interesting to determine how crucial the absence is of attribute synthesis in the upper bounds of Theorems 1 and 2. In other words, are these results also true for classes of attribute grammars for which a modest amount of attribute synthesis (such as concatenation) is allowed?
References


