数理解析研究所講究録
547巻 1985年 112-127

Global Storage Allocation in Attribute Evaluation

Takuya Katayama

Hisashi Sasaki

Department of Computer Science

Tokyo Institute of Technology

2-12-1 Ookayama, Meguro-ku

Tokyo 152, Japan

1. Introduction

Global storage allocation for attributes in an attribute grammar evaluator is discussed and an algorithm for determining if a given set of attribute occurences can share a common global storage is obtained. It is widely known that the key problem to be overcome for generating production quality compilers from attribute grammars is to find a proper way of allocating storages to attribute instances in derivation trees. In the most primitive case, a separate storage location is allocated to each attribute instance. This storage allocation strategy is, however, far from desired as each storage is used only once during the whole process of attribute evaluations. Several people considered the problem of economical use of storages in attribute evaluation. Saarinen [3] classified attribute occurrence into significant and nonsignificant ones and stored significant ones in a stack. Katayama [2] proposed to store every attribute instance in a stack.

A great economy in space and time could be expected by allocating a global storage to attribute instances. As semantic rules of attribute grammars are purely applicative, straightforward implementation is time and space-consuming by that it may involve costly copy operations of big data values such as symbol tables. To treat big attribute values efficiently, we have to do two things. First, find a set of attribute instances which could be allocated a common storage location. Second, change the way of accessing the data from by-value to by-update, i.e., update the data instead of changing the whole values. Farrow [1] showed the usefulness of these issues in his evaluator.

In this paper, we consider the problem of determining if a given set of attribute occurences could share a common storage location. Sethi [4] consider the problem of storage globalization for finite dags and introduced the concept of pebble game. Though his method is effective for a given finite dag, it cannot be directly applied to attribute evaluation where we have to deal with dags of attribute dependency of indefinite size. He also made a proposal for attribute grammars [5], it is not enough.

We propose here an algorithm for testing attribute globalization. In general, storage allocation strategy for attributes is largely dependent on the structure of attribute evaluator. The evaluator we are considering is constructed on the principle of assigning a procedure to each pair of nonterminal symbol and its synthesized attribute and translating the attribute grammar into a set of procedures [2]. The evaluator is recursive in nature and the problem of testing attribute globalization is reduced to the problem of production rule level, which is solved by a modified version of Sethi's algorithm for finite dags.

2. Definitions

An attribute grammar is a context free grammar G augmented with semantic rules. Formally it is defined by

(G, A, F)

where

(1) $G = (V_N, V_T, P, S)$ is a context free grammar with V_N a set of

nonterminal symbols, V_T a set of terminal symbols, P a set of production rules and S the initial symbol. In the following we assume without loss of generality that the initial symbol S never appears in the right side of any production rule.

When

$$p: X_0 \to w_0X_1w_1 \ldots w_{n-1}X_nw_n$$

is a production rule in p where $w_i \in V_T^*$ and $X_i \in V_N$, we refer the occurrence of the nonterminal symbol X_k by X(p,k) and n by n(p). Then, the rule p is expressed as

$$p: X(p,0) \to$$
$$w_0X(p,1)w_1 \ldots w_{n(p)-1}X(p,n(p))w_{n(p)}.$$

(2) A is a set of attributes. Each $X \in V_N$ has a subset A[X] of A. An element of A[X] is called an attribute of X. A[X] is a disjoint union of the set INH[X] of inherited attributes and the set SYN[X] of synhesized attributes.

When p is a production rule we say that p has an attribute occurrence a.X(p,k) if $a \in A[X(p,k)]$ and $0 \le k \le n(p)$.

(3) F is a set of semantic functions. A semantic function $f_{p,v}$ is associated with every attribute occurrence v = a.X(p,k) such that $a \in SYN[X(p,0)]$ or $a \in INH[X(p,k)]$ for $1 \le k \le n(p)$. It specifies how to compute the value of v from values of other attribute occurrences of the rule p. We denote the set of these attribute occurrences by $D_{p,v}$. It is called a dependency set of $f_{p,v}$. If $D_{p,v} = \{v_1,\ldots,v_m\}$ then $f_{p,v}$ is a mapping

$$\mathrm{domain}(a_1)\times\ldots\times\mathrm{domain}(a_m) \to \mathrm{domain}(a)$$

where $v_i = a_i.X(p,k_i)$ and domain(a) is a value domain of the attribute a in general. We express this fact by an equation

$$v = f_{p,v}(v_1,\ldots,v_m).$$

Now we define several dependency relations among attributes and related concepts.

(1) Let p be a production rule. A dependency graph DG_p for the production rule p, which gives dependency relationship among attribute occurrences of p, is defined by

$$DG_p = (DV_p, DE_p)$$

where the vertex set DV_p is the set of all attribute occurrences of p and the edge set DE_p is the set of dependency pairs for p. Formally

$$DV_p = \{a.X(p,k) \mid 0 \le k \le n(p) \text{ and } a \in A[X(p,k)]\}$$

$$DE_p = \{(v1,v2) \mid v1 \in D_{p,v2}\}.$$

(2) When a derivation tree T is given, a dependency graph DG[T] for the derivation tree T which represents dependencies among attributes of nodes in T is defined. DG[T] is obtained by pasting DG_p's together according to the syntactic structure of T. Let p be the production rule applid at the root of T and T_k the k-th subtree of T. DG[T] is recursively constructed from DG_p, $DG[T_1]$, ..., $DG[T_{n(p)}]$ in the following way.

$$DG[T] = (DV_T, DE_T)$$

where

$$DV_T = DV'_p \cup \cup_{k=1}^{n(p)} DV[T_k]$$

$$DE_T = DE'_p \cup \cup_{k=1}^{n(p)} DE[T_k]$$

and $DG'_p = (DV'_p, DE'_p)$ is the graph obtained from DG_p by replacing every attribute occurrence a.X(p,k) in the production rule p by the corresponging attribute instance a.X(p,k).r(k) in the tree T, where r(k) is the root node of T_k. We assume r(0) denotes the root of T.

$$DV'_p = \{a.X(p,k).r(k) \mid a.X(p,k) \in DV_p\}$$

$$DE'_p = \{(a.X(p,k).r(k), b.X(p,j).r(j)) \mid (a.X(p,k), b.X(p,j)) \in DE_p\}$$

(3) When r(0) is the root node labeled by $X \in V_N$ of a derivation tree T and $s \in SYN[X]$, we define DG[s.X,T], a subgraph of DG[T], by removing vertices and edges which are not located on any path leading to s.X.r(0).

(4) Let T be a derivation tree with the root labeled by $X \in V_N$. DG[T] determines IO graph IO[X,T] of X with respect to T. It gives how synthesized attributes of X depend on other attributes of X through the derivation tree T. That is,

$$IO[X,T] = (A[X], E_{IO}[T])$$

where an edge $(a,s) \in E_{IO}[T] \subset A[X] \times SYN[X]$ exists iff DG[T] has a path from v_a to v_s, where v_a and v_s are vertices for attributes a and s of the root X of T respectively, and this means that the attrbute a is reqired to evaluate the synthesized attribute s.

For general attribute grammars, X may have finitely many IO graphs IO_1, ..., IO_N where $IO_k = (A[X], E_k)$. Superposing these IO_k's results in the superposed IO graph

$$IO[X] = (A[X], E_{IO}), \text{ where } E_{IO} = \cup_{k=1}^{N} E_k.$$

(5) For a production rule p, its augmented dependency graph is defined by

$$DG_p^* = (DV_p^*, DE_p^*)$$

where

$$DV_p^* = DV_p,$$

$$DE_p^* = DE_p \cup \{(a.X(p,k), b.X(p,k)) \mid (a,b) \text{ is an edge of } IO[X(p,k)] \text{ for } 1 \le k \le n(p)\}.$$

DG_p^* represents dependency relations among attribute occurrences of p, which is given partly by semantic functions and partly by derivation trees.

(6) An attribute grammar is said absolutely non circular iff DG_p^* contains no cycle for any production rule p.

3. An Attribute Grammar Evaluator

Here we briefly sketch an attribute grammar evaluator which we consider in this paper [2]. Let X be a nonterminal symbol of an absolutely noncircular attribute grammar G and s a synthesized attributes of X. We associate a procedure

$$R_{X,s}(u_1, \ldots, u_m, T; v)$$

with each pair (X,s), where $u_1, \ldots, u_m$ (abbreviated by $\vec{u}$) are parameters corresponding to the inherited attributes in I = $\{i \mid (i,s) \in IO[X]\}$ and v is a parameter for s. T is a parameter for derivation tree. Prameters to the left (right) of ";" are input (output) parameters. This procedure is intended to evaluate the synthesized attributes s when supplied with the values of inherited attributes in I and a derivation tree T.

The procedure $R_{X,s}(\vec{u},T;v)$ is constructed in the following manner. First we introduce variable symbols for attribute occurences. However, for the sake of convenience, the same symbols are used for attribute occurences and variables which correspond to them. We consider they are local variables of the procedure. $R_{X,s}$ is of the following form:

procedure $R_{X,s}(\vec{u};v)$

case production(T) **of**

p_1: $H_{p1,s}$

p_2: $H_{p2,s}$

...

...

end

end

where $\vec{u}$, v and T are reference parameters. p_1, p_2, ... are productions with left side symbol X. The procedure $R_{X,s}$ determines the production rule p applied at the root of T and it perform a sequence $H_{p,s}$ of statements to compute the values of attribute occurences in p.

The sequence $H_{p,s}$ is constructed in the following steps where we put X_0=X.

(1) Construct the augmented dependency graph DG_p^*.

(2) Remove from DG_p^* vertices and edges which are not located on any path leading to $s.X_0$. Denote the resulting graph by

$DG_p^*[s] = (V,E)$.

(3) To each attribute occurences

$x \in V' = V-\{i.X_0 | i \in INH[X_0]\}$

assign a statement st[x] for evaluating x as follows.

Case 1: If $x = i.X_k$ for some $i \in INH[X_k]$ and k=1, ... ,n or $x = s.X_0(=v)$ for the attribute $s \in SYN[X_0]$, then st[x] is the assignment statement

$x \leftarrow f_{p,x}(z_1, \ldots, z_r)$

where $f_{p,x}$ is the semantic function for x and $D_{p,x} = \{z_1, \ldots, z_r\}$ is the dependency set for f.

Case 2: If $x = t.X_k$ for some $t \in SYN[X_k]$ and k = 1, ... ,n then st[x] is the procedure call statement

call $R_{Xk,t}(w_1, \ldots, w_h, T_k; x)$

where w_1, ..., w_h are attribute occurences on which $t.X_k$ is dependent. T_k is the k-th subtree of T.

(4) Let $x_1, x_2, \ldots, x_N$ be element in V' which are listed according to the topological ordering determined by E, i.e., if $(x_a, x_b) \in E$ then $a<b$. Then $H_{p,s}$ is the sequence of statements

$$st[x_1],\ st[x_2],\ \ldots,\ st[x_N].$$

So far we have only considered construction of a procedure for a paticular nonterminal. The construction of evaluator over the entire G is:

(1) We first construct the procedure $R_{S,s}$ for the initial symbol S and its synthesized attribute s by the algorithm we have stated. The body of $R_{S,s}$ may contain calls of other procedures $R_{X,s}$'s and they are constructed in the same way;

(2) Repeat this process until no more new procedure appears;

(3) Arrange these procedures and add statements to input the values of inherited attributes of S to call the procedure $R_{S,s}$ and to output the values of S completes the construction of the evaluator for G. Note that, when a derivation tree is evaluated by this evaluator, the value of every attribute instance is stored in the stack of activation records or procedure calls.

4. Pebbling for Attribute Grammar

Sethi introduced pebbling for dags as a model for computation using global storages [4]. It is essentially to serialize computations so that values in global storages may not be erroneously lost. Let D=(V,E) be a finite dag whose nodes are labeled by storage locations in a set L. We denote the storage allocation function by

$$g : V \to L,$$

i.e., g(v) is the storage allocated to a node v. Pebbling for D is a sequence

$$v_1, v_2, \ldots, v_n$$

of elements $v_1, \ldots, v_n$ of V such that the following conditions are satisfied.

(1) if $(v_i,v_j) \in E$ then $i<j$.

(2) for each $i = 1, \ldots, n$, if there exists j such that $i<j$ and $(v_i,v_j) \in E$, then there is not k satisfying $i<k<j$ and $g(v_k) = g(v_i)$.

Sethi gave an algorithm for deciding if there exists a pebbling for a given (D,g,L).

Now consider the pebbling for attribute grammars. Let T be a derivation tree of an attribute grammar G. We consider to assign global storage to each attribute instance in DG[T], the dependency graph of attribute instances in T, and to find a pebbling for (DG[T],g,L). The storage allocation function g is

$$g : DV_T \to L$$

where DV_T is the nodes of DG[T]. DV_T is considered as a subset of $A \times V_N \times node(T)$, where node(T) is the set of nodes of the derivation tree T. In this formulation, a storage allocated to an attribute occurence a.X of a rule p may depend on a particular node it is applied and the same attribute occurence may be assigned different storages at different nodes of T. This is not desirable to us, as we are going to perform attribute evaluation by the evaluator stated in the previous section and its structure is independent of specific derivation trees. What we require about our storage allocation function is that it allocates a fixed storage to each attribute occurence where it appears in T. So, we consider g as

$$g : A \times V_N \to L$$

and pebbling is to be performed under g^* for each T.

$$g^* : A \times V_N \times \mathrm{node}(T) \to L$$

where

$$g^*(a,X,v) = g(a,X) \text{ for any } v \in \mathrm{node}(T).$$

That is, every node of DG[T] with nonterminal symbol X and its attribute $a \in A[X]$ is given a common storage location g(a,X). A pebbling for (DG[T],g^*,L) defines an order of evaluating attribute instances in T using global storages L. This pebbling, however, is not enough for our purpose as we are interested in a particular class of evaluators which imposes another restriction on the order in which attribute instances are evaluated.

5. Recursive Pebbling

Here we formulate a pebbling for attribute grammars which takes the structure of our attribute evaluator into consideration. Let T be a derivation tree. In the pebbling for (DG[T],g^*,L) considered in the privious section, attribute instances are evaluated in an order specified only by (1) dependency relation DG[T] and (2) legal use of global storage. However, the order is also restricted by the recursive nature of the evaluator. Let X be a nonterminal symbol and s a synthesized attribute of X. Consider to evaluate an instance of s in DG[T]. The evaluator calls a procedure $R_{X,s}$ in which it selects a production rule p with left hand symbol X and executes a sequence of statements. They are either to evaluate an inherited attribute i of right hand symbol Y or to evaluate a synthesized attribute t of a right hand symbol Z. When t is to be evaluated, the corresponding procedure $R_{Z,t}$ is called.

From the above description of the evaluator, we can see what restriction should be posed on the order of attribute evluation. That is, when the evaluator begins to evaluate an instance of attribute t, it is tied down to the task until it is finisfed. During the task, any other attribute instance cannot be evaluated unless it is necessary for evaluating t. This suggests the following definition.

[Definition] Recursive Pebbling

Let T be a derivation tree with root node labeled by $X \in V_N$ and $p: X \to w_0 X_1 w_1 X_2 \ldots X_n w_n$, $X_i \in V_N$, $w_i \in V_T{}^*$ be a production rule applied there. For $s \in SYN[X]$, consider the dependency graph DG[s.X.T] a subgraph of DG[T] specialized for s.X and pebbling C[s.X,T] for (DG[s.X,T],g^*,L). Suppose $r.X_i$, $t.X_j$, ... are synthesized attribute occurences of p on which s.X is dependent in DG_p^*, and T_i, T_j, ... are subtrees of T with root nodes X_i, X_j, ... $\in V_N$. Let $C[r.X_i,T_i]$, $C[t.X_j,T_j]$, ... be pebblings for $(DG[r.X_i,T_i],g,L)$, $(DG[t.X_j,T_j],g,L)$, ... Then, C[s.X,T] is a recursive pebbling iff

(1) C[s.X,T] is a pebbling for (DG[s.X,T],g^*,L) in the sence of Sethi,

(2) $C[r.X_i,T_i]$, $C[t.X_j,T_j]$, ... are recursive pebblings, and,

(3) C[s.X,T] contains $C[r.X_i,T_i]$, $C[t.X_j,T_j]$, ... as subsequences.

The next definition gives the condition that evaluation of attributes can be performed by our evaluator using storages L and a storage allocation function g. Note that attribute occurences which are not specified by g is allocated a storage in the stack of activation records of procedure calls.

[Definition] (g,L)-evaluatability of attribute grammars

Let G be an attribute grammar. G is (g,L)-evaluatable iff the following (1) and (2) are satisfied for any $X \in V_N$, $s \in SYN[X]$ and a derivation tree with root X.

(1) there exists a recursive pebbling for $(DG[s.X,T], g^*, L)$, and

(2) $C[s.X,T]'$ is independent of T, where $C[s.X,T]'$ is obtained from $C[s.X,T]$ by replacing (a) subsequences $C[r.X_i,T_i]$, $C[t.X_j,T_j]$, ... by single symbols $r.X_i$, $t.X_j$, ... and (b) any other element $a.X_k.v$ by $a.X_k$.

The condition (2) in the above states that the order in which the attributes of the root node of T and its immediate descendant nodes are evaluated is independent of pebblings for subtrees T_i.

6. An Algorithm for testing (g,L)-evaluatability

When an attribute grammar G is given, there are, in general, infinitely many derivation trees and we cannot test its (g,L)-evaluatability by directly resorting to its definition. We can, however, test it production-rule-wise as stated blow. For each production rule $p: X \rightarrow w_0 X_1 w_1 X_2 \ldots X_n w_n$, define an extended dependency graph DG_p°

$$DG_p^\circ = (DV_p^\circ, DE_p^\circ)$$

where

$$DV_p^\circ = DV_p \cup \{[s.X_k] \mid k=1,\ldots,n,\ s \in SYN[X_k]\}$$

$$DE_p^\circ = DE_p \cup \{(i.X_k,[s.X_k]) \mid k=1,\ldots,n,\ (i,s)\in IO[X_k]\}$$
$$\cup \{([s.X_k],s.X_k) \mid k=1,\ldots,n,\ s\in SYN[X_k]\}.$$

The special symbol $[s.X_k]$ is introduced to represent a computation for s in the subtree T_k with root node X_k.

Now consider a pebbling on DG_p°. The storage assignment function g specifies a storage allocated to each node in DV_p. For the node $[s.X_k]$, we allocate a set of storages which may be used in the possible computations in derivation trees T_k which follow X_k. In general, multiple storages may be used in T_k, so we have to allocate a subset $USE[s.X_k]$ of L to $[s.X_k]$. Define $g^\circ : DV_p^\circ \to 2^L$,

$$g^\circ(v) = \{g(v)\} \quad \text{if } v\in DV_p$$
$$= USE[s.X_k] \text{ if } v=[s.X_k].$$

There is an iteration algorithm for determinig $USE[s.X_k]$. The pebbling for (DG_p°,g°,L) is defined similarly as the usual pebbling except that (1) multiple storage locations may be allocated to a single node and (2) there is an additional requirement that some nodes are placed adjacent in this case. We call it an extended pebbling. It is defined as a sequence

$$v_1,\ v_2,\ \ldots,\ v_n$$

of nodes $v_1, \ldots, v_n$ of DG_p° such that

(1) if $(v_i,v_j)\in DE_p^\circ$ then $i<j$

(2) for each $i=1, \ldots, n$, if there exists j such that $i<j$ and $(v_i,v_j)\in DE_p^\circ$ then there is not k satisfying $i<k<j$ and $g^\circ(v_k)\cap g^\circ(v_i)\neq\emptyset$

(3) if $v_i=[s.X_k]$ then $v_{i+1}=s.X_k$.

The condition (3) in the above is introduced to state the property of our evaluator that it is tied down to computation of an attribute until it is completed. We can prove the next lemma.

[Lemma]

A modified version of Sethi's algorithm can determine if there exists an extended pebbling for $(DG^{\circ}_{p}, g^{\circ}, L)$.

Now we give the main result of our paper. It states that (g,L)-evaluatability of an attribute grammar can be reduced to extended pebblings of dependency graphs of production rules.

[Theorem]

For an attribute grammar G, a set L of storages and a storage allocation function $g: A \times V_N \rightarrow L$, G is (g,L)-evaluatable iff there is an extended pebbling for $(DG^{\circ}_{p}, g^{\circ}, L)$ for any production rule p.

7. Conclusion

Global storage allocation in attribute evaluation is studied and a decision algorithm is given to test whether, for a given attribute grammar G, it is possible to construct an attribute grammar evaluator for G which stores values of attribure instances in storages L under storage allocation function g.

References

[1] Farrow,R., LINGUIST-86 Yet Another Translator Writing System Based on Attribute Grammars. Proceeding of the SIGPLAN'82 symposium on compiler construction, 160-171, June 1982.

[2] Katayama,T., Translation of Attribute Grammar into Procedures. ACM Transaction on Programming Languages and Ststems, Vol.6, No3, July 1984.

[3] Saarinen,M., On Constructing efficient evaluators for attribute grammars. Lecture Notes in Computer Science, 62, Springer-Verlag, 1978, 382-396.

[4] Sethi,R., Pebble games for studing storage sharing. Theoritical Computer Science Vol.19, No.1, 69-84, July 1982.

[5] Sethi,R., The glogal storage needs of a subcomputation. Eleventh Annual ACM Symposium on Principle of Programming Languages, 148-157, January 1984.