Computer Science Journal of Moldova, vol.**5**, no.3(15), 1997

Grammar flow analysis reduction to scheduling problem

K. Chebotar

Abstract

The article describes a variant of *GFA* problem connected with shared storage. It shows that this *GFA* problem may be reduced to a well known problem of scheduling theory (the Bellmann–Johnson problem for two–machine conveyer system). It proposes an effective algorithm to solve a particular case of the problem.

1 Introduction

GFA [1, 2] is a technique that offers a unified framework for describing and proving of properties of context–free grammar. *GFA* is performed on the grammar (syntax) graph, whose nodes correspond to nonterminals or productions, and whose edges are drawn according to the productions. The propagation functions are defined to assign some information to graph nodes. This information may have a very complex structure. According to the nature of these functions two kinds of flow analysis schemes are distinguished. In the bottom–up scheme, the information is assigned to the nodes bottom–up that is that the information attached to the left–hand side of a production depends on the information attached to the symbols in the right–hand side of production. In the top–down scheme information is propagated from the root top–down and reflects the dependence of the information attached to the symbols in the right–hand side of a production on the information attached to the left–hand side nonterminal but generally also on the information previously attached during the bottom–up propagation.

The specific interest for *GFA* problem appears as a result of application of attribute grammars technique to the design and implementation of language processors. Attribute grammars offer convenient tools to solve various problems at the compiler construction level. For example, such well–known problems as attribute grammar circularity test and attributes evaluation. In this context new methods were proposed to solve these particular cases which would be effective for more common properties of *GFA*. Let note some of them:

GP grammar partitioning [3], in which the grammar is decomposed into subgrammars

WS weak stability [4] which allow to skip the processing of nonterminals and productions that generates only terminal strings

SS using the definition of iteration semantic stability [5, 6] determines the evaluated information "ages", that permit to avoid redundant computations.

The article describes a variant of *GFA* problem connected with shared storage. It shows that this *GFA* problem may be reduced to well–known problem of scheduling theory (the Bellmann–Johnson problem for two–machine conveyer system). It proposes an effective algorithm to solve a particular case of this problem.

2 Notations and definitions

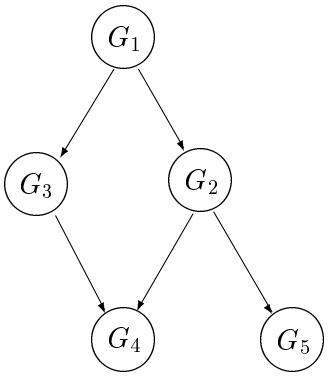
Let $G = (V_N, V_T, P, Z)$ be a context-free grammar, where V_N is the set of nonterminals, V_T is the set of terminals, P is the production set and Z is the axiom. Each production will be of the form
$p : X_0 \to X_1 X_2 \dots X_{n_p}$.

The grammar graph is a directed graph $\Gamma_G = (V_N, U_G)$ with the set of nodes V_N and the set of edges U_G defined as follows:

$$U_G = \{(X_i, X_j) \mid \exists p \in P, p : X_i \to \alpha X_j \beta,\ X_j \in V_N,\ \alpha, \beta \in (V_N \bigcup V_T)^*\}.$$

Let $K = \{G_1, G_2, \ldots, G_l\}$ be the connected components (equivalence classes) of graph Γ_G and $\Gamma_0 = (K, U_0)$ be the condensed–graph for Γ_G. Every permutation of Γ_0 nodes $\pi = (G_{i_1}, G_{i_2}, \ldots, G_{i_l})$, obtained as the result of topological sorting of Γ_0 is an admissible graph treewalk used to resolve all enumerated above *GFA* problems.

Let Π will be a set of all admissible permutations of Γ_0 nodes. For example, for the graph described in the Fig.1 the following permutations are admissible:



$$\Pi = \{\ (G_5, G_4, G_3, G_2, G_1),$$
$$(G_5, G_4, G_2, G_3, G_1),$$
$$(G_4, G_3, G_5, G_2, G_1),$$
$$(G_4, G_5, G_2, G_3, G_1),$$
$$(G_4, G_5, G_3, G_2, G_1)\}$$

Fig.1

Moreover the admissible permutation practically does not influence the effectiveness and the final result of algorithms used to solve *GFA* problems. One could try to reach more effective solutions of *GFA* problems using a choice of admissible permutation from Π, in particular more effective storage allocation. Let us show that the variant of this problem may be reduced to a well–known scheduling problem for two–machine conveyer system.

3 The optimal scheduling problem for two–machine conveyer system

Because the readers interested in GFA problems may be not acquainted with the scheduling theory, we will formulate here the classical variant

of problem: to obtain the optimal schedule for processing N objects by the two–machine conveyer system.

Let n_i (m_i) be the processing time for i–th object by the first (second) machine. It is supposed that:

- all the objects must be processed by the first machine then by the second one or only by one of them (the case $n_i = 0$ or $m_i = 0$);
- each machine can process simultaneously no more than one object;
- the same object cannot be processed by both machines simultaneously;
- the objects processing order is the same for both machines.

The problem: order the processing of the objects by conveyer in such a way that minimize the total time of all objects processing.

The scheme of such conveyer system is presented at Fig.2.

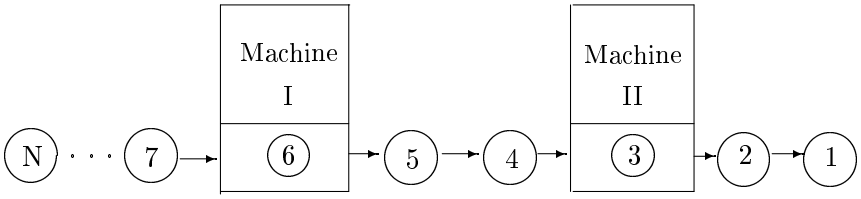


Fig.2

Lets x_i will be the wait time for arrival of the i–th object to the second machine. The typical schedule is the following:

Machine I	n_1	n_2		n_3		n_4	n_5		
Machine II	x_1	m_1	x_2	m_2	x_3	m_3	m_4	x_5	m_5

Fig.3

It is necessary to minimize $T(\pi) = \sum\limits_{j=1}^{N}(m_i + x_i)$ for all $\pi \in \Pi$. The values x_i may be expressed through n_i and m_j in the following way:

$x_1 = n_1$
$x_2 = max(n_1 + n_2 - x_1 - m_1, 0)$
$x_3 = max(n_1 + n_2 + n_3 - x_1 - x_2 - m_1 - m_2, 0)$
$\dots$
$x_J = max(\sum\limits_{i=1}^{J} n_i - \sum\limits_{i=1}^{J-1} x_i - \sum\limits_{i=1}^{J-1} m_i, 0)$

Applying the simple rules such as $\max(a, (\max(b, c)) = \max(a, b, c)$ and $a + \max(b, c) = \max(a + b, a + c)$, we obtain

$x_1 = n_1$
$x_1 + x_2 = \max(n_1 + n_2 - m_1, n_1)$
$x_1 + x_2 + x_3 = \max(n_1 + n_2 + n_3 - m_1 - m_2, n_1 + n_2 - m_1, n_1)$
$\dots$
$\sum\limits_{i=1}^{N} x_i = \max(Y(\pi, 1), Y(\pi, 2), \dots Y(\pi, N))$,

where $Y(\pi, 1) = n_1, Y(\pi, J) = \sum\limits_{i=1}^{J} n_i - \sum\limits_{i=1}^{J-1} m_i, J = 2, 3, \dots, N.$

Because in the equality $T(\pi) = \sum\limits_{i=1}^{N} x_i + \sum\limits_{i=1}^{N} m_i$. $\sum\limits_{i=1}^{N} m_i$ is constant, it is necessary to minimize $\sum\limits_{i=1}^{N} x_i$, i.e. it is necessary to find

$\min\limits_{\pi\in\Pi}\max\limits_{1\le j\le N} Y(\pi, j)$.

The simple algorithm is known, proposed by S.M. Johnson, to resolve this problem. Some complexities may appear when any restrictions are formulated to the set of admissible objects permutation Π. For example, the objects may be interconnected by an acyclic directed graph. In this case every topological sort of graph nodes will be an admissible permutation.

4 The optimal storage allocation

Let $\Gamma_0 = (K, U_0)$ be the condensed graph for the grammar graph Γ_G, $K = \{G_1, G_2, \ldots, G_l\}$. Applying the method, looking the same like shown in [3] to each node G_i one can evaluate n_i,the volume of storage necessary to test the circularity of subgrammar G_i, and m_i,the volume of storage which becomes free after G_i testing.

Algorithm A_0 (topological ordering of the Γ_0 nodes)

$i := 1$;
repeat
- give the rank i to all terminal nodes G_j of U_0 (no edge exists from it), $\alpha(G_j) := i$;
- discard them from Γ_0, along with the edges that enter it;
- $i := i + 1$;

until $\Gamma_0 = \oslash$

Algorithm A_1 (topological ordering of grammar nonterminals)

init : **foreach** $G_i \in U_0$ **do**
  foreach $X_j \in G_i$ **do**
    $\beta(X_j) := \alpha(G_i)$
  endfor
endfor;
ordering : **foreach** $(X_i, X_j) \in \Gamma_0$ & $X_i \in G_1$& $X_j \in G_2$&$G_1 \neq G_2$ **do**

$\beta(X_j) = max(\beta(X_j), \alpha(G_1))$
endfor.

Therefore, the priority $\beta(X_j) = k$ shows that after subgrammar G_i ($\alpha(G_i) = k$) circularity testing all the storage, occupied by relations of dependency for nonterminal X_j may be made free. This information permits for all node subgrammars G_i of graph Γ_0 to calculate (maybe approximatively) characteristics n_i, the volume of storage necessary to keep the dependencies during circularity test for subgrammars G_i and m_i, the storage made free immediately after G_i testing.

Let $\pi = (G_1, G_2, \ldots, G_l)$ be an arbitrary admissible permutation of Γ_0 nodes. For subgrammar G_1 testing it is necessary to allocate the storage which volume is equal to n_i. To test G_2 it is necessary to allocate the storage of volume equal to $n_1 - m_1 + n_2$. If we denote by $Y(\pi, G_J)$ the volume of storage necessary for circularity test of G_J then we obtain:

$$Y(\pi, G_1) = n_1$$

$$Y(\pi, G_2) = n_1 - m_1 + n_2$$

$$\ldots$$

$$Y(\pi, G_J) = \sum_{i=1}^{J} n_i - \sum_{i=1}^{J-1} m_i$$

The total volume of storage necessary for the test of all nodes of graph Γ_0 is equal to $F(\pi) = \max\limits_{1 \le j \le l} Y(\pi, G_j)$.

Changing the permutation π the problem of optimal storage allocation is reduced to find the $\min\limits_{\pi \in \Pi} F(\pi)$, i.e. to the problem to find optimal schedule for two-machine conveyer system. To solve this problem one can use any known at present algorithm of scheduling theory. If on the values of n_i and m_i to put restrictions $n_i \ge m_i$ then a simple and effective algorithm to solve this problem may be presented. The variant of such a problem will be described in the next chapter.

5 An algorithm to solve the optimal storage allocation problem

Let $\Gamma_0 = (K, U_0)$,be the directed acyclic graph with $K = \{G_1, G_2, \ldots, G_l\}$, $n_i \geq m_i, 1 \leq i \leq l$. We denote $A(G_i)$ the set of Γ_0 nodes reachable from node G_i, i.e. $A(G_i) = \{G_j \mid (G_i, G_j) \in U_0^*\}$ and $\Gamma_A(G_i) = (A)G_i), U_0(G_i))$ - the subgraph of Γ_0, generated by the set $A(G_i)$.

Algorithm A_2 (construction of optimal permutation π^*)

$i := 1$.

repeat

- Select the arbitrary node G_i^0 of graph Γ_0 for which $m_i = \max\limits_{G_j \in U_0} m_j$. This node we denote as the leader.
- Construct the graph $\Gamma_A(G_i^0) = (A(G_i^0), U_0(G_i^0))$, generated by the set of nodes $A(G_i)$. Let $\pi_i = (G_i^1, G_i^2, \ldots, G_i^{k_i}, G_i^0)$ be an arbitrary admissible permutation of $\Gamma_A(G_i^0)$ nodes.
- Discard from the graph Γ_0 the subgraph $\Gamma_A(G_i^0)$ and all the edges (G_1, G_2) entering the nodes of subgraph $\Gamma_A(G_i^0)$, i.e. $G_2 \in A(G_{i_0}), G_1 \notin A(G_i^0)$. Let us denote the graph obtained as the result of this transformation as Γ_0^1.
- $\Gamma_0 := \Gamma_0^1, i := i + 1$.

until $\Gamma_0 = \oslash$.

Because at every step of algorithm we discard at least one node

of graph Γ_0, at the finite number of steps r, $1 \leq r \leq l$, the algorithm finishes. Let $\pi^* = (\pi_1, \pi_2, \ldots, \pi_r) = (G_1^1, G_1^2, \ldots, G_1^{k_1}, G_1^0, G_2^1, G_2^2, \ldots, G_2^{k_2}, G_2^0, \ldots, G_r^1, G_r^2, \ldots, G_r^{k_r}, G_r^0)$.

Lemma 1. $\pi \in \Pi$.

The demonstration is concluded directly from the definition of $A(G_i)$ and the construction of π^*.

Let $\pi_i = (G_i^1, G_i^2, \ldots, G_i^{k_i}, G_i^0), 1 \leq i \leq r$. The following lemma holds:

Lemma 2. $\max(Y(\pi^*, G_i^1), Y(\pi^*, G_i^2), \ldots, Y(\pi^*, G_i^{k_i}),$ $Y(\pi^*, G_i^0)) = Y(\pi^*, G_i^0)$.

Proof. For arbitrary $j, 1 \leq j \leq n_i$, let us denote:

$d(G_i^j) = n(G_i^j) - m(G_i^j), D_i = \sum\limits_{s=0}^{k_i} d(G_i^s)$.

$Y(\pi^*, G_i^j) = \sum\limits_{s=1}^{i-1} D_s + \sum\limits_{s=1}^{j-1} d(G_i^s) + n(G_i^j)$,

$Y(\pi^*, G_i^0) = \sum\limits_{s=1}^{i-1} D_s + \sum\limits_{s=1}^{j-1} d(G_i^s) + n(G_i^j) - m(G_i^j) + \sum\limits_{s=j+1}^{k_i} d(G_i^s) + n(G_i^0)$.

Therefore $Y(\pi^*, G_i^0) - Y(\pi^*, G_i^j) = \sum\limits_{s=j+1}^{k_i} d(G_i^s) + n(G_i^0) - m(G_i^j)$.

Because $\sum\limits_{s=j+1}^{k_i} d(G_i^s) \geq 0, m(G_i^0) \geq m(G_i^j), n(G_i^0) \geq m(G_i^0)$ then $Y(\pi^*, G_i^0) \geq Y(\pi^*, G_i^j)$.

The Lemma 2 shows that the maximal value of all $Y(\pi^*, G_i^j)$ must be found among values $Y(\pi^*, G_i^0)$, i.e. $F(\pi^*) = \max\limits_{1 \leq i \leq r} Y(\pi^*, G_i^0)$, and that the values of all $Y(\pi^*, G_i^0)$ does not depend on the order of nodes $G_i^1, G_i^2, \ldots, G_i^{k_i}$ in the permutation π_i.

If at some step i for the nodes G_p and G_q we have $m_p = m_q$ and this is the maximal value among the all graph nodes then in the correspondence with the algorithm A_2 the following choices of leader node are possible:

1) $G_i^0 = G_p$, and at the next $(i+1)$-th step $G_{i+1}^0 = G_q$ is selected, $G_q \notin A(G_p)$;

2) $G_i^0 = G_q$, and at the next $(i+1)$-th step $G_{i+1}^0 = G_p$ is selected, $G_p \notin A(G_q)$;

3) $G_i^0 = G_p$,$G_q \in A(G_p)$;

4) $G_i^0 = G_p$,$G_p \in A(G_q)$;

For these cases we obtain the following permutations:

1) $\pi^i = (G_i^1, G_i^2, \dots G_i^{k_i}, G_i^0)$,
$\pi_{i+1} = (G_{i+1}^1, G_{i+1}^2, \dots, G_{i+1}^{k_{i+1}}, G_{i+1}^0)$,
$G_i^0 = G_p, G_{i+1}^0 = G_q$. Let us denote the final permutation by π_1^*.

2) $\pi^i = (G_i^1, G_i^2, \dots G_i^{k_i}, G_i^0)$,
$\pi_{i+1} = (G_{i+1}^1, G_{i+1}^2, \dots G_{i+1}^{k_{i+1}}, G_{i+1}^0)$,
$G_i^0 = G_q, G_{i+1}^0 = G_p$. Let us denote the final permutation by π_2^*.

3) $\pi^i = (G_i^1, G_i^2, \dots, G_i^s, \dots, G_i^{k_i}, G_i^0)$, $G_i^0 = G_p, G_i^s = G_q$. Let us denote the final permutation by π_3^*.

4) $\pi^i = (G_i^1, G_i^2, \dots, G_i^s, \dots, G_i^{k_i}, G_i^0)$, $G_i^0 = G_q, G_i^s = G_p$. Let us denote the final permutation by π_4^*.

Lemma 3. $Y(\pi_1^*, G_{i+1}^0) = Y(\pi_2^*, G_{i+1}^0) = Y(\pi_3^*, G_i^0) = Y(\pi_4^*, G_i^0)$.

Proof. First let us show that $Y(\pi_1^*, G_{i+1}^0) \geq Y(\pi_1^*, G_i^0)$.
$Y(\pi_1^*, G_{i+1}^0) - Y(\pi_1^*, G_i^0) = \sum_{s=j}^{k_{i+1}} d(G_{i+1}^s) + n(G_q) - m(G_p) \geq 0$ because $n(G_q) - m(G_p) = n(G_q) - m(G_q) \geq 0$. Analogously $Y(\pi_2^*, G_{i+1}^0) \geq Y(\pi_2^*, G_i^0)$. Taking into account that $A(G_p) \cup A(G_q)$ is the same set for all the cases 1)–4), let us show that $Y(\pi_1^*, G_{i+1}^0) = Y(\pi_2^*, G_{i+1}^0)$.
$Y(\pi_1^*, G_{i+1}^0) - Y(\pi_2^*, G_{i+1}^0) = n(G_p) - m(G_p) + n(G_q) - n(G_q) + m(G_q) -$

$n(G_p) = 0$. In the same manner $Y(\pi_1^*, G_{i+1}^0) - Y(\pi_3^*, G_i^0) = n(G_p) - m(G_p) + n(G_q) - n(G_q) + m(G_q) - n(G_p) = 0$.

The rest of equalities is proved in the same way.

It is easy to generalize the Lemma 3 for several nodes with the same characteristics m_i. The Lemma 3 shows that the value of $Y(\pi^*, G_i^0)$ for the last of such nodes in π^* does not depend on the order of choice of leader nodes and is the maximal among the values Y for all preceding leader nodes.

Theorem. $F(\pi^*) = \min\limits_{\pi \in \Pi} F(\pi)$.

Proof. In correspondence with the Lemmae 2 and 3 it is necessary to search the maximal value of Y in leader nodes $G_1^0, G_2^0, \ldots, G_r^0$ of permutation π^*. Let π will be the arbitrary permutation from Π, $\pi = (G_{i_1}, G_{i_2}, \ldots, G_{i_l}), \pi \neq \pi^*$. To prove the Theorem it is sufficient to show that for every leader node G_j^0 of permutation π^*, in π there exist the node G_{i_k}, such that $Y(\pi, G_{i_k}) \geq Y(\pi^*, G_j^0)$.

Let us take now the minimal segment of $(G_{i_1}, G_{i_2}, \ldots, G_{i_k})$ of permutation π, which includes all the leader nodes $(G_1^0, G_2^0, \ldots, G_j^0)$ of permutation π^*. This segment must be finished with one of the leader nodes $G_0^1, G_0^2, \ldots, G_0^j$. Let us denote this node by $G_{i_k} = G_t^0, 1 \leq t \leq j$. This results from the definition of Π, $A(G_0^i)$ and from the construction of π^*. If $G_t^0 = G_j^o$ then obviously $Y(\pi, G_{i_k}) \geq Y(\pi^*, G_0^j)$, because the segment $(G_{i_1}, G_{i_2}, \ldots, G_{i_k})$ besides all elements of permutations $\pi_1, \pi_2, \ldots, \pi_j$, may contain another elements.

If $G_t^0 \neq G_j^0$ then let us take the difference $Y(\pi, G_{i_k}) - Y(\pi^*, G_j^0) = n(G_j^0) - m(G_j^0) + n(G_t^0) + D_\pi - n(G_j^0) - m(G_j^0) + D_\pi \geq 0$, because $n(G_t^0) \geq m(G_t^0) \geq m(G_j^0)$. By D_π we denoted the summa of diffferences $d(G_{i_s})$ of $(G_{i_1}, G_{i_2}, \ldots, G_{i_k})$ segment elements not belonging to the segment $\pi_1, \pi_2, \ldots, \pi_j$, $D_\pi \geq 0$. The Theorem is proved.

Obtained permutation π^* may be used in the implementations of algorithms **GP, WS, SS** for the effective storage allocation.

6 Conclusions

The main steps in construction of the permutation π^* are:
– sorting the graph Γ_0 nodes in descendent order of values m_i;
– extraction of the subgraphs $\Gamma(A_i)$;
– construction of the permutations π_i.

Therefore, the general complexity of algorithm is $O(\log_2 l+q)$ where l, q are the numbers of nodes and edges of graph Γ_0.

The proposed algorithm may be used also to solve the variant of scheduling problem for two–machine conveyer system on the oriented acyclic graph. The problem formulated bellow may be also solved by this algorithm.

Let us assume that the complex of interactive problems(events) is solved and there is an acyclic directed graph $D = (N, U)$, expressing the relations between the problems (events). For the i-th problem it is known the total number of resources n_i, necessary for the problem execution and total number of resources m_i, made free after its execution. Among the all admissible schedules we need to find that one, which minimizes the total number of resources, necessary to resolve the whole complex.

References

[1] U. Möncke, R.Wilhelm "Grammar Flow Analysis". Attribute Grammars, Applications and Systems. Proceedings of International Summer School SAGA, Prague, Chehoslovakia, June 4–13, 1991, pp 151-186.

[2] M.Jourdan, D. Parigot. "Techniques for Improving Grammar Flow Analysis". European Symp. on Programming (ESOP 90),Copenhagen, N.Jones, ed., pp. 240-255, Lect. Notes in Comp. Sci, 432, Springer - Verlag, New-York- Heiderberg-Berlin, May 1990.

[3] K. Chebotar. "Some Modifications of Knuth's Algorithm for Verifying Cyclicity of Attribute Grammars". Progr. and Computer Software, 7, (Jan. 1981), pp. 58-61.

[4] P. Deransart, M. Jourdan, B. Lorho. "Speeding up Circularity Tests for Attribute Grammars", Acta Informatica, 21 (Dec. 1984), pp. 375-391.

[5] M.Jourdan, D. Parigot. "More on Speeding up Circularity Tests for Attribute Grammars". Rapport RR-828, INRIA, Rocquencourt, Apr. 1988.

[6] K. Chebotar. "The Design and Analysis of Algorithms for Attributed TWS Construction". PhD thesis, Computing Center, Russian Academy of Sciences, Russian Academy of Sciences, Moscow, 1984 (in Russian).

K.Chebotar
Institute of Mathematics,
Academy of Sciences of Moldova,
5 Academiei str., Kishinev,
MD 2028, Moldova
phone: (373–2) 738073
e–mail: chebotar@math.moldova.su

Received 3 June, 1997