On Arithmetic Expressions and Trees

R. R. Redziejowski*
IBM Nordic Laboratory, Lidingö, Sweden

A description is given of how a tree representing the evaluation of an arithmetic expression can be drawn in such a way that the number of accumulators needed for the computation can be represented in a straightforward manner. This representation reduces the choice of the best order of computation to a specific problem under the theory of graphs. An algorithm to solve this problem is presented.

KEY WORDS AND PHRASES: arithmetic expression, compiler design, graph theory, programming, storage minimization, topological ordering, tree
CR CATEGORIES: 4.12, 5.32

1. Introduction

In [1], Nakata discussed the evaluation of an arithmetic expression in an order that would minimize the number of accumulators needed for the computation. As shown below, the tree used in [1] can be modified in such a way that the required number of accumulators has a straightforward graphical representation. This representation leads to the specific problem of a "minimum width" ordering of a tree, which is the subject of the present paper.

To evaluate an expression, e.g. $((a + b) - c \cdot d)/(e(f - g))$, one has to perform a number of arithmetic operations: $A = a + b, B = c \cdot d, C = f - g, D = A - B, E = e \cdot C, F = D/E$, thus computing the partial results $A, B, \cdots, E$ and final result F. This process can be represented in the form of a tree, as in Figure 1. Each vertex of that tree represents one operation; an arc from a vertex x to a vertex y represents a partial result that is used in the operation x and computed in the operation y.

Choosing a feasible order for the operations is equivalent to a topological sorting of the tree, i.e. to placing its vertices in such a sequence that a vertex x precedes a vertex y if there is an arc from y to x. (This condition is equivalent to the requirement that each partial result be evaluated before being used.)

* Present Address: Laboratoriet for Impuls-og Cifferteknik, Danmarks Tekniske Højskole, 2800 Lyngby, Danmark

The result of such a sorting is conveniently represented as in Figure 2. In the following, we call a diagram such as Figure 2 a "lineup" of the three in Figure 1. It can be regarded as a timescale representation of the computation.

Suppose we take a "snapshot" of the computation while executing an operation x. Such a snapshot will show

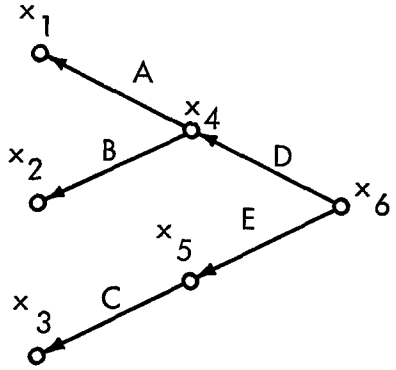


Fig. 1

some results of the previous operations being stored for further use and not participating in the operation x. It is easy to see that in Figure 2 these stored results are represented by the arcs passing over the vertex x. If the number of these arcs is w, we must have at least $w + 1$ storage elements (accumulators or core locations) at our disposal at the considered moment: w for the results already stored and one for the result of x. The number of storage elements we have to reserve for the entire computation is thus defined by the maximum number of arcs passing over a single vertex in Figure 2; we shall call this number the "width" of our "lineup." The choice of

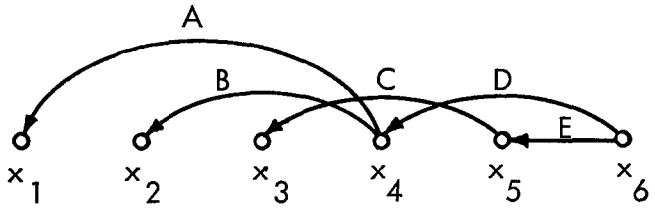


Fig. 2

the best order of operation is thus reduced to constructing a minimum width lineup for the tree in Figure 1.

In Section 3 an algorithm to accomplish this construction is suggested; Section 2 contains the definitions.

Although the trees corresponding to arithmetic expressions are always binary, the discussion is not more complicated when the general case is considered. Thus in the following, the trees are not assumed to be binary.

2. Definitions

In the following we use the concept of a (directed) *graph*, defined as an ordered pair (X, U) consisting of a nonempty set X of *vertices* and a set U of *arcs*. An arc from a vertex $x \in X$ to a vertex $y \in X$ is here denoted by $(x, y) \in U$. For $x, y \in X$ such that there exists an arc $(x, y) \in U$, we call y a *successor* of x. The reader is assumed to be familiar with the elementary concepts of a path in a graph, a circuit, etc.

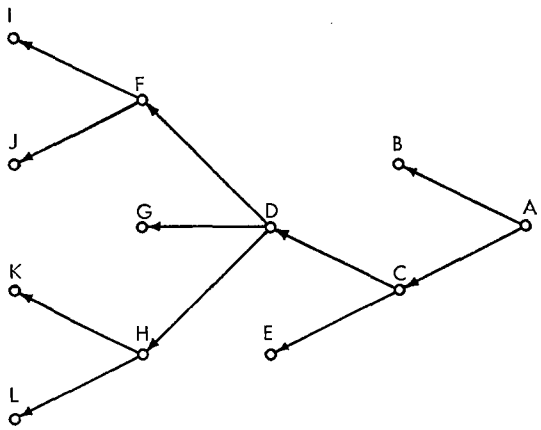
A graph (X, U) is called a *tree* if it is finite, does not contain circuits, and contains a vertex $r \in X$ such that:

(a) r is not the terminal vertex of any arc, and

(b) every vertex $x \in X$, $x \neq r$, is the terminal vertex of exactly one arc.

The vertex r having the above properties is called the *root* of the tree. A vertex $x \in X$ that is not the initial vertex of any arc is called a *leaf* of the tree; any vertex that is not a leaf is called a *node*. For a tree $G = (X, U)$ and its vertex $x \in X$, we denote by $G(x)$ a subtree consisting of x and all vertices accessible from x via a path in G. A *lineup* of a tree $G = (X, U)$ is formally defined as a sequence $\varphi = (x_1, x_2, \cdots, x_n)$ of all vertices of G such that for every arc $(x_i, x_j) \in U$ holds $j < i$. In the lineup $\varphi = (x_1, x_2, \cdots x_n)$, an arc $(x_i, x_j) \in U$ is said to *pass over* a vertex $x_k \in X$ if $j < k < i$.

For this lineup, let w_i denote the number of arcs passing over a vertex $x_i \in X$. The *width* of the lineup φ is then defined as the maximum of w_i for $i = 1, 2, \cdots, n$, and is denoted $W(\varphi)$.



Col. 1: x	Col. 2: v(x)	Col. 3: φ(x)		
L	0	L		
K	0	K		
J	0	J		
I	0	I		
H	1	K L H	K, L;	0, 1;
G	0	G		
F	1	I J F	I, J;	0, 1;
E	0	E		
D	2	I J F K L H G D	F, H, G;	1, 2, 2;
C	2	I J F K L H G D E C	D, E;	2, 1;
B	0	B		
A	2	I J F K L H G D E C B A	C, B;	2, 1;

FIG. 3

3. Algorithm

A minimum width lineup of a given tree G can be constructed by filling in a table as in Figure 3. The table is divided into 3 columns and n rows. We start by enumerating in column 1 all vertices of G. We write there, bottom to top, first the root of G, then all its successors, then successors of these successors, and so on. To each vertex we thus assign one row of the table in such a manner that if a vertex y is a successor of a vertex x, the row assigned to x lies below that assigned to y. Having filled column 1, we complete the table row by row.

Suppose we have have already completed $m \geqq 0$ rows, and are about to complete the next, the $(m + 1)$-th row. Let the vertex corresponding to this row be x. We complete the $(m + 1)$-th row by one of the following rules, depending on whether x is a leaf or node of G:

Rule 1. If x is a leaf of G, write 0 in column 2; write x in column 3.

Rule 2. If x is a node of G, find in G all successors of x; let them be $y_0, y_1, \cdots, y_k$ where $k \geqq 0$. Find in the table the rows assigned to $y_0, y_1, \cdots, y_k$; due to the way we have filled column 1, all of them must be among the m rows that have already been completed. Let $v(y_i)$, for $i = 0, 1, \cdots, k$, denote the contents of column 2 in the row assigned to y_i, and $\varphi(y_i)$ the contents of column 3 in that row. Place $y_0, y_1, \cdots, y_k$ in a sequence according to decreasing values of $v(y_i)$ for $i = 0, 1, \cdots, k$. Let this sequence be $(y_{i_0}, y_{i_1}, \cdots, y_{i_k})$; then $v(y_{i_0}) \geqq v(y_{i_1}) \geqq \cdots \geqq v(y_{i_k})$. Compute the number $p = \max(p_0, p_1, \cdots, p_k)$ where $p_j = v(y_{i_j}) + j$ for $j = 0, 1, \cdots, k$. Write p in the $(m + 1)$-th row, column 2; in column 3 write $\varphi(y_{i_0})$ followed by $\varphi(y_{i_1})$, etc., followed by $\varphi(y_{i_k})$, followed by x.

To illustrate the application of rule 2, the sequence $(y_{i_0}, y_{i_1}, \cdots, y_{i_k})$ and the numbers $p_0, p_1, \cdots, p_k$ are shown in Figure 3 to the right of each row that was completed according to that rule. For example, for $x = D$ we have $y_{i_0} = F$, $y_{i_1} = H$, $y_{i_2} = G$, and $p_0 = v(y_{i_0}) + 0 = 1$, $p_i = v(y_{i_1}) + 1 = 2$, $p_2 = v(y_{i_2}) + 2 = 2$.

We now demonstrate that after completing the table, the sequence appearing in the last row, column 3, is a minimum width lineup of G.

4. Proof of the Algorithm

Suppose the table has been completed as above for a tree $G = (X, U)$ with root r. For $x \in X$ let $v(x)$ be the number appearing in the row of the table assigned to x, column 2; let $\varphi(x)$ be the sequence appearing in the same row, column 3. The last row was assigned to the root r; for this row we have $v(r)$ and $\varphi(r)$, respectively. The following two theorems can be proved about the contents of the table.

THEOREM 1. *For every vertex $x \in X$, $\varphi(x)$ is a lineup of the subtree $G(x)$, and its width is $v(x)$.*

THEOREM 2. *For every lineup of G, its width is not less than $v(r)$.*

From Theorem 1 and the fact that $G(r) = G$, it follows immediately that the sequence $\varphi(r)$ appearing in the last row, column 3 is a lineup of G having width $v(r)$. Theorem 2 states that this width is a minimal one.

PROOF OF THEOREM 1. To prove the theorem, it is enough to demonstrate that (1) the theorem holds for every leaf of G, and (2) for every node x of G, if the theorem holds for all successors of x, it also holds for x.

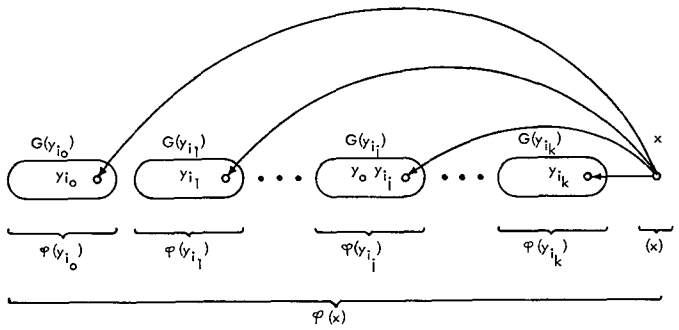


FIG. 4

That (1) is true follows from rule 1. If x is a leaf, the subtree $G(x)$ consists of a single vertex x. The sequence (x) is a lineup of such a tree and its width is 0.

That (2) is true can be seen as follows. Let x be a node of G, and let $y_{i_0}, y_{i_1}, \cdots, y_{i_k}$ be its successors, as in rule 2. Let us assume that the theorem holds for all these successors, i.e. that for $j = 0, 1, \cdots, k$, $\varphi(y_{i_j})$ is a lineup of the subtree $G(y_{i_j})$ and its width is $v(y_{i_j})$. The concatenation of $\varphi(y_{i_0})$, $\varphi(y_{i_1})$, $\cdots$, $\varphi(y_{i_k})$ and (x) in rule 2 corresponds to a construction shown in Figure 4. Having $k + 1$ trees, $G(y_{i_0})$, $G(y_{i_1})$, $\cdots$, $G(y_{i_k})$, we construct the tree $G(x)$ by adding the vertex x and $k + 1$ arcs, (x, y_{i_0}), (x, y_{i_1}), $\cdots$, (x, y_{i_k}). It is easy to see from Figure 4 that the sequence $\varphi(x)$ so obtained must be a lineup of $G(x)$. To show that the width of this lineup is $v(x)$, let us consider a vertex y belonging to the tree $G(y_{i_j})$, where $0 \geqq j \geqq k$. It is easy to see that the number of arcs passing over y is less than or equal to the number p_j as defined in rule 2: there are j arcs, (x, y_{i_0}), (x, y_{i_1}), $\cdots$, $(x, y_{i_{j-1}})$, outside the tree $G(y_{i_j})$, and as many as $v(y_{i_j})$ arcs within that tree. From this the width of $\varphi(x)$ is bound to be $\max(p_0, p_1, \cdots, p_k) = v(x)$; the theorem thus holds for the vertex x also.

PROOF OF THEOREM 2. The theorem obviously holds for $v(r) = 0$. Assume $v(r) > 0$ and suppose there exists a lineup φ of G having a width $W(\varphi) < v(r)$. For every node x of G do the following: Find the sequence $(y_{i_0}, y_{i_1}, \cdots, y_{i_k})$ and the numbers $p_0, p_1, \cdots, p_k$ computed when completing the row assigned to x according to rule 2; then find the smallest j, $0 \geqq j \geqq k$, such that $p_j = v(x)$, and mark the arcs (x, y_{i_0}), (x, y_{i_1}), $\cdots$, (x, y_{i_j}). For example, if $x = D$ in Figure 3, then $j = 1$ and we mark the arcs (D, F) and (D, H). The result of such a marking for the tree and table in Figure 3 is shown in Figure 5. One can easily see that (1) every node of G is the initial vertex of at least one marked arc, and (2) for every marked arc $(x, y) \in U$, x is the initial vertex of at least $q + 1$ marked arcs, where $q = v(x) - v(y)$.

Now mark all leaves of G accessible from the root r via the marked arcs; from (1) it follows that there must be at least one such leaf. Let $z \in X$ be the marked leaf that appears farthest to the right in φ; let μ be the path from r to z (obviously, μ must consist entirely of marked arcs). Take an arc $u = (x, y) \in U$ belonging to μ. Let $v(x) - v(y) = q$; according to (2), x is then the initial vertex of at least $q + 1$ marked arcs. If $q > 0$, let $u_1, u_2, \cdots, u_q$ be the marked arcs beginning at x and distinct from u. For $i = 1, 2, \cdots, q$ set out from x via the arc u_i and follow the marked arcs as far as possible (for illustration see Figure 6). The path μ_i so traversed must end in a marked leaf, and thus to the left of z in the sequence φ (z was so chosen). Since φ is a lineup of G, x must be to the right of z in φ, and thus the path μ_i must contain an arc u_i' passing over z. This gives q arcs passing over z: $u_1', u_2', \cdots, u_q'$. By taking in turn all arcs in μ as the arc u above, one can find in this way $v(r) - v(z) = v(r)$ arcs passing over z, which contradicts assumption that $W(\varphi) < v(r)$. (Notice that it could be proved, in a similar way, that for every $x \in X$, $v(x)$ is a minimal width for all possible lineups of the subtree $G(x)$.)

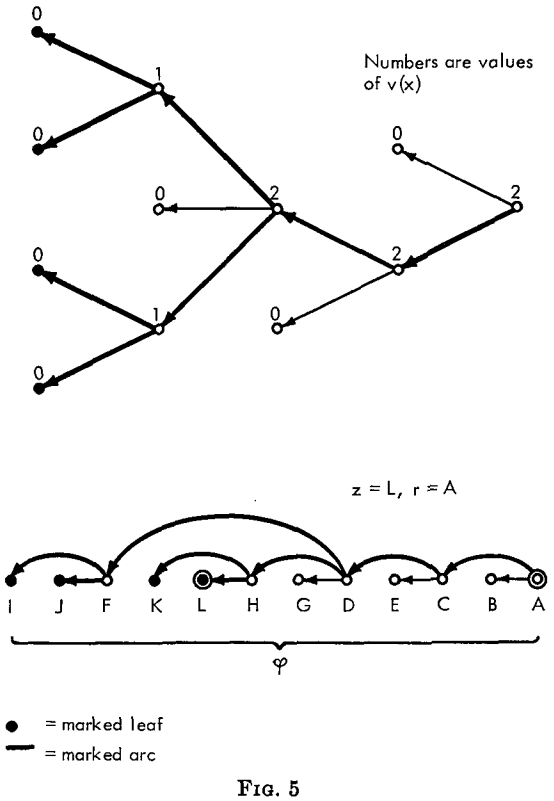


FIG. 5

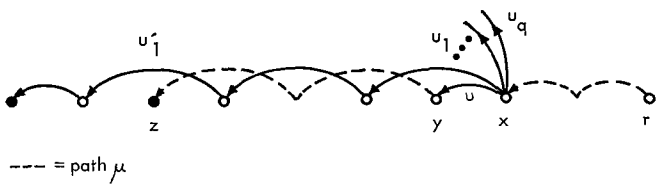


FIG. 6

5. Discussion

If the tree G represents a computation as described in the introduction, the various objects discussed in Sections 2 through 4 have the following interpretation. A leaf of G represents an operation performed entirely on initial data, and a node of G an operation having at least one partial result as its argument. The root of G represents the operation of computing the final result. A lineup φ of G represents a permissible order of executing the operations; it can be looked upon as a kind of program for the computation represented by G. The width $W(\varphi)$ represents the number of storage cells required to execute that program, minus one. For an operation x, the subtree $G(x)$ represents a part of the computation that consists of x and all operations needed to evaluate the arguments of x. Theorem 1 states that the sequence $\varphi(x)$ appearing in the row of the table assigned to x is a valid program for the "partial" computation $G(x)$, and that the number $v(x)$ appearing in the same row is the number of storage cells required to execute this program, minus one. The algorithm presented here consists of constructing such "partial" programs. We first construct the most elementary ones by using rule 1; then we combine them according to rule 2 in order to obtain those which are more and more complex.

The sequences $\varphi(y_{ij})$ appearing in rule 2 represent the programs for evaluating the arguments of the operation x. When combining them in one program $\varphi(x)$, we order them according to decreasing storage requirements. This principle is identical with the principle suggested in [1] and our algorithm is basically the same as the algorithm described there. The proof stated in Section 4 may thus be regarded as a formal proof of the algorithm given in [1].

RECEIVED MARCH, 1968; REVISED AUGUST, 1968

REFERENCES

1. NAKATA, I. On compiling algorithms for arithmetic expressions. *Comm. ACM 10*, 8 (Aug. 1967), 492–494.

Letters to the Editor

Simple Procedures That Lose Precision

EDITOR:

A number of readers have written concerning the article by G. D. Miller "An Algorithm for the Probability of the Union of a Large Number of Events" [*Comm. ACM 11*, 9 (Sept. 1968), 630–631], pointing out the well-known simplification:

$$P\left(\bigcup_{\alpha=1}^{n} E_\alpha\right) = 1 - P\left(\bigcap_{\alpha=1}^{n} \bar{E}_\alpha\right)$$

and the trivial computation it implies.

However, computers perform rational arithmetic of limited precision. For small event probabilities, therefore, the simple procedure loses precision rapidly. It should have been pointed out in the paper that increased precision is the principal merit of the algorithm. Both the referee and myself would like to apologize to all who may have been misled for not asking the author to include a few words about precision.

ROBERT M. MCCLURE, Editor
Programming Techniques Department, CACM

A Glossary of Computer Science

EDITOR:

I feel a much needed addition to the editions of *Communications of the ACM* would be a glossary of computer science terms. This glossary could logically be divided into two parts. The first part would contain those words and terms which have come into standard usage and also have a single meaning. This part would be printed toward the front of each edition (preferably) or at least once each year. The second part would be an extension of the "key words and phrases" which precedes the main text of each article. This part would define those terms which are particular to the article or which are being used with a meaning different from the standard usage (i.e. the main glossary).

STEPHEN LOWE
Litton Industries
P.O. Box 7601
Van Nuys, CA 91409

An ACM Meeting in Chicago?

EDITOR:

Recent events have demonstrated the imprudence of holding a meeting in Chicago. The American Psychological Association and the American Sociological Association have recognized Chicago's dangers, and they have canceled projected meetings in that city.

With a view to protecting its membership, the ACM should follow suit. The ACM Conference of 1971 should be relocated.

ROBERT R. FENICHEL
MIT, 545 Technology Square
Cambridge, MA 02139

Shortening the Mask Querying Routine in Peekabit

Key Words and Phrases: peekaboo, superimposed coding, natural language searching, text searching, information compaction, computer search technique
CR Categories: 3.74

EDITOR:

R. G. Glasser, professor of computer science and physics at the University of Maryland, has pointed out to me (9/25/68) that the mask querying routine in PEEKABIT [*Comm. ACM 11* (Sept. 1968), 595–598] can be shortened by about 20 percent time-wise if the master mask is stored on the master tape in complemented form (there is no reason in our programs why it could not be) and queried as follows:

```
CAL   CPM+4,1     First word of question mask
ANA   KPCOM       First word of complemented master mask
TNZ   NOTIN       If not zero, one or more words not present
CAL   CPM+5,1     Second word, question mask
ANA   KPCOM+1
TNZ   NOTIN
etc.
```

Total program running time would not be reduced 20 percent, of course, but Professor Glasser's suggested coding is much better and should effect some overall saving.

FRED C. HUTTON
Computing Technology Center
Union Carbide Corporation
Nuclear Division
Oak Ridge, TN 37830