

Corrections to Stone's Compiler Procedures

Key Words and Phrases: compilation, one-pass compilation, parallel processor

CR Categories: 4.12

EDITOR:

I would like to point out some corrections to the ALGOL procedure in the appendix to the article by Harold S. Stone, "One-Pass Compilation of Arithmetic Expressions for a Parallel Processor" [*Comm. ACM* 10, 4 (Apr. 1967), 220-223].

In line 10: *treelevel := 1;*
should be *treelevel := 0;*

In line 11: *for treelevel + 1 while . . .*
should be *for treelevel := treelevel + 1 while . . .*

In line 14: *minus := false;*
should be *minus := false*

In line 24: *output ('+')*
should be *oustring (1, '+')*

In line 32: *. . . while item = equiv('↑') do*
should be *. . . while item = equiv('↑') do*

In line 37: *if divide then outstring(1, '/')*
should be *if divide then outstring(1, '/');*

In line 38: *divide := false;*
should be *divide := false*

Although the above errors are most likely typographical in origin, there is an omission which makes the program recurse to infinite depth upon encountering '(' in the input string. When procedure "primary" is evoked and encounters '(' it then evokes procedure "expression" without updating the position of the pointer in the input string. Hence when "primary" is again evoked, and encounters '(', the process is repeated.

To correct this, I inserted a call to procedure "scan" between lines 61 and 62, which also eliminates the need to position the input pointer at the first character prior to calling procedure "expression" initially. The algorithm was then coded in Raytheon 520 (recursive) Real-Time FORTRAN, and produced the correct answers to the examples given.

JEFFREY L. DEVEBER
38 High Street
Cambridge, MA 02138

The Go To Statement Reconsidered

Key Words and Phrases: go to, jump instruction, branch instruction, program intelligibility, program sequencing, algorithm progress, algorithm analysis

CR Categories: 4.20

EDITOR:

I was taken aback by Dijkstra's attack on the go to statement, which is an obviously useful and desirable statement [*Comm. ACM* 11, 3 (Mar. 1968), 147-148]. I reject the implication that the "quality" of a program is directly correlated with a minimal use of go to and similar statements.

It is not clear to me that there is a real motivation for the attack. I do not see, for example, that our "intellectual powers are better geared to master static relations" rather than dynamic ones.

There is a discussion of a coordinate system or "characteristic indices" for the progress of an algorithm. The use of this coordinate system is not detailed, so it is not clear what properties it should have. In any case, it is noted that it is not interesting to limit oneself to algorithms where the progress is described simply

by pointing to the last statement executed. It is proposed to allow "repetition clauses" and procedure calls. The progress of the algorithm is then "characterized by a (mixed) sequence of textual and/or dynamic indices." I interpret this to mean that all procedures, all statements, and all iteration loops are named. The sequence of "characteristic indices" consists of a list (in the proper order) of (1) all statements where procedures are called and the corresponding procedure name, (2) all iteration loops entered along with the number of repetitions. If procedures are called in an iteration loop, then this call is meshed into the sequence between the repetitions of the iteration loop.

This sequence of "characteristic indices" implies an elaborate machinery for its construction. It requires names and variables that do not naturally occur in the algorithm. In complicated algorithms, both the construction and interpretation of this sequence are not trivial. Since such an index is only interesting for complicated algorithms, I visualize that the construction and/or interpretation would be done automatically.

It is stated that if one allows the go to statement, then the construction of the sequence of "characteristic indices" becomes very difficult. It seems to me that the addition of the go to would require only a modest extension of the machinery already implied by the use of procedures and repetition clauses. One need merely note which go to's one encountered and where they went.

It is stated that if go to's are allowed then one can uniquely describe the progress of an algorithm by means of a counter of the number of actions taken. This is not true in any sense that I can find.

It is claimed that the go to statement is, in some sense, programmer dependent while the other statements are not. I am unable to understand this reasoning.

I find the emotional tone of this attack as disquieting as the "scientific" analysis. How many poor, innocent, novice programmers will feel guilty when their sinful use of go to is flailed in this letter?

JOHN R. RICE
Purdue University
Computer Sciences
Mathematical Sciences Building
Lafayette, IN 47907

A Reply by E. W. Dijkstra

EDITOR:

I hope that the following answer to colleague John R. Rice will clear up his misunderstanding.

The construction of a mechanism that, along with the computation asked for by the programmer, keeps up to date the sequence of characterizing indices requires very little additional machinery, e.g. one stack. The operations to be performed on this initially empty stack are:

- (1) at procedure call the return address is stacked;
- (2) at procedure return the top element of the stack is unstacked (the unstacked element is the corresponding return address);
- (3) at repetition clause entry a counter with a standard initial value (0 or 1, say) is stacked;
- (4) after computation of the Boolean expression controlling a repetition clause the corresponding counter is the top element of the stack; if the Boolean indicates that control remains in the loop the top element of the stack is increased by 1, if control leaves the loop this top element is unstacked.

The contents of this stack with the current value of the order counter on top of it is a representation of the "(mixed) sequence of textual and/or dynamic indices." In an implementation the above could be used at great advantage, for instance to report when a run-time error has occurred. In the case of a lengthy com-

(Please turn to page 541)

been most concerned with total utilization of only core and central processor time. In the present case we also need a neat dovetailing of I/O requirements, which in turn requires accurate knowledge of the personality of the job. Penny [5] has pointed out the difficulty of obtaining such information in an open shop scientific computing environment, but there are periods, even in open shop installations, when "production" jobs predominate. The characteristics of such jobs can be determined and used to modify the scheduling algorithm. The Brookhaven National Laboratory has taken some tentative steps in this direction in an effort to relieve disk congestion [6].

The second method (improving the predictive ability of the priority assigner) offers a little more promise, primarily because the present algorithm is based on the whole history of the job, instead of only its recent history. Many jobs, for instance, observe the pattern of heavy I/O at the beginning, followed by a more or less compute-bound period, and end with a burst of output. Such jobs acquire a high priority during the initial input phase, then they lock out other, currently more I/O-bound jobs while the initial effect wears off, and finally they are themselves locked out when trying to output their results. By considering only recent history, the priority assigner will more accurately reflect the current character of each job.

The major cause of operator delays in the Chippewa system is the assignment of tape drives to active programs. On some systems (the ATLAS system, for instance [7]), great pains are taken to assign tapes before granting access to central memory and the central processor. In Chippewa, tapes must be assigned through the control point just like any other system resource, and the control point remains idle from the time the request is posted until the operator completes the assignment. There are two practical approaches to the problem: to emulate the ATLAS system and preassign all tapes; or to roll out the job during the delay, so that its core may be freed for a more productive program. The course adopted by a given installation depends upon the local job mix, of course: an installation which averages one tape per job, thirty jobs in the input queue, and six tape drives, for instance, is unlikely to gain much from the ATLAS method. This choice (between preassignment and roll-out) should be carefully considered, for reduction of dead time is the most promising of the three possibilities for immediate system improvement.

Conclusions. The treatment described here was based on the principle that the object of the scheduling algorithm(s) for multiprogramming systems should be to maximize the number of simultaneous processes. In applying it to the Chippewa system, a 33 percent increase (from 1.5 to 1.98) in simultaneous processes achieved a doubling of throughput. The particular steps by which these results were obtained (increasing the number of jobs in core and dynamically assigning priorities on the basis of I/O-boundedness) were—and in general will be—easy to implement. Furthermore, their efficacy is not limited to the

Chippewa system: Penny [5] has shown them effective for any multiprogramming system with one level of storage.

Acknowledgments. The author thanks the following people who, wittingly or not, have contributed to the ideas presented here: the programming staffs at CERN and the Brookhaven National Laboratory, especially Charles Symons of the former and Les Lawrence of the latter. He also thanks Jeremy Knight of the Lawrence Radiation Laboratory, Berkeley, California, who has implemented much of what has been described; and the referees for directing his attention to [5].

RECEIVED SEPTEMBER, 1967; REVISED JANUARY, 1968

REFERENCES

1. 6600 Chippewa Operating System. Pub. No. 60124500, Control Data Corp., Apr., 1965.
2. Chippewa Operating System Reference Manual. Pub. No. 60134400, Control Data Corp., Dec., 1965.
3. STEVENS, D. F. System evaluation on the Control Data 6600. Rep. UCRL-17893, Lawrence Radiation Lab., Oct. 1967.
4. CODD, E. F. Multiprogram scheduling. *Comm. ACM* 3, 6 (June 1960), 347-350 (Parts I and II); *Comm. ACM* 3, 7 (July 1960), 413-418. (Parts III and IV).
5. PENNY, J. P. An analysis, both theoretical and by simulation, of a time-shared computer system. *Comput. J.* 9, 1 (1966), 53-59.
6. LAWRENCE, L. L. Private communication.
7. HOWARTH, D. J., JONES, P. D., AND WYLD, M. T. The ATLAS scheduling system. Proc. AFIPS 1963 Spring Joint Comput. Conf., Vol. 23, Spartan Books, New York, pp. 59-67.

Letters to the Editor—continued from page 538

(A Reply by E. W. Dijkstra)

putation the suggestion made by Rice to "note which `go to`'s one encountered and where they went" easily leads to an unacceptably long record as it has all the unattractive features of a trace.

In my letter "Go To Statement Considered Harmful" I did not stress the possibility of actually implementing this additional mechanism, for even without it I would advocate to abstain from using `go to` statements as their abolishment reduces the programmer's intellectual effort to continue to understand what he is doing and what the machine will do under control of his program: once one has been trained to control the sequencing via procedures, alternative clauses, and repetitive clauses, exclusively, one soon discovers that programming has become much easier. My letter gave an explanation of this phenomenon by analyzing some of the demands that the effort of programming makes upon the human mind: the dynamical and textual indices have been presented primarily as a thinking aid.

I have not stressed this very explicitly, and if this omission has caused his misunderstanding, I offer him—and all other puzzled readers—my sincere apology, as I also do for the misprint on the last line but one on page 147, where "progress" should be read for "process."

EDSGER W. DIJKSTRA
*Department of Mathematics
 Technological University
 Eindhoven, The Netherlands*