Faults in functions, in ALGOL and FORTRAN

I. D. Hill

Medical Research Council, 242 Pentonville Road, London N1 9LB

When a fault is found during the evaluation of a function, usually because of an invalid argument, what action should be taken? The advantages and disadvantages of various possibilities are discussed.

(Received August 1969, Revised November 1970)

Suppose we wish to write a procedure to evaluate a function, A, with argument n, and only certain values of n are permitted. For the sake of argument, let us say that n must be zero or positive. It will usually be desirable to include, in the procedure, a test for $n \geqslant 0$. The question is, if this test fails what action should be taken?

There are several possibilities, each with its advantages and disadvantages. In examining these I shall pay particular attention to ALGOL 60 and ANSI FORTRAN. The users of other languages may like to consider whether the same arguments apply or not.

The first point to be considered is whether the language, and the compiler, concerned permit functions to have side effects.

Side effects prohibited

If side effects are not allowed there would seem to be only two possible actions:

(*a*) to give the function an impossible result. For example, if the function is necessarily positive, a result of -1 might be used to indicate incorrect usage. This has the disadvantage that the user must be aware of the possibility and always test for it. This may be easy to remember the first time one uses the function, and when one has just read its documentation but it is very easily forgotten on later occasions.

I have known a case where this method was used, in calling from magnetic tape the heights of a number of men, -1 being returned if the information was not available. Forgetting to test the results led to two men with heights of -1 being used in the ensuing analysis.

Testing for the impossible result is simple enough, if remembered, provided that the function is called in a simple context, such as

$$x := A(n)$$

but the advantage of a function over a subroutine call is that it can be used within an expression, and if the expression is at all complicated, the impossible result may be very difficult to test for.

Furthermore, this method is not always possible. For some functions, any numerical value whatever can be a valid result, and no impossible value is available for a fault indication;

(*b*) to abandon the functional approach altogether and use a subroutine call instead i.e. instead of

real procedure *A*(*n*) or FUNCTION A(N)

to resort to

procedure *A*(*n*, *result*) or SUBROUTINE A(N, RESULT)

This has the advantage that all compilers (I sincerely hope) will allow side effects in this context, thus permitting the use of any of the methods (*c*)-(*h*) listed below, but the disadvantage that calling A must now always be a self-contained 'statement'. Not being able to include the call in an expression is certainly a handicap in program writing.

Side effects permitted

If side effects are allowed, a number of actions become possible, as follows:

(*c*) to print a message and terminate the job. This is the method usually adopted for standard functions available without user declaration, '*sqrt*(*n*)' for example will generally produce this result if $n < 0$. It is a bit drastic to recommend as a normal device for user-defined functions, where recovery from the fault would often be possible if the job were not terminated. In ALGOL 60 there is also the difficulty that there is no method of terminating, such as the 'STOP' statement of FORTRAN;

(*d*) to give the procedure an extra parameter, of Boolean (or LOGICAL) type, which, as a side-effect, is set to **true** if the call was successful, or to **false** otherwise. This has the disadvantage of method (*a*) that the user has to remember to make the test, but the advantages that the existence of the parameter reminds him that the test is necessary, and that even if the function is used in a complicated expression, the Boolean parameter is still there to be tested after the calculation of the expression has been completed. However the work of completing the calculation after a fault has occurred, will usually be wasted, and may even cause some other fault;

(*e*) to have an extra parameter as in (*d*) above, but of integer type. This has the advantage that if more than one type of fault is possible, different integer results can be given to indicate which one occurred. Otherwise the advantages and disadvantages are as for (*d*). This method is recommended as editorial policy for algorithms published in Applied Statistics (Working Party on Statistical Computing, 1968);

(*f*) to have a Boolean procedure (or LOGICAL FUNCTION), giving the result of the validity test as the main output, while setting the required function value as a side effect. This rather backhand method does at least force the user to make the fault test. It has the disadvantages that it looks perverse to the human user, and that the required result is no longer in functional form and cannot be included in an expression;

(*g*) in ALGOL, to give the function a label parameter, and jump to that label if faulty usage occurs. This has the advantages that the user is forced to set a label to go to, and forced to think about the action he is going to take if he arrives there. Furthermore, although the function can still be used as part of an expression, useless calculations in evaluating the expression will be abandoned as soon as the fault is found.

It has the disadvantages that (i) the ALGOL Report (Naur *et al.*, 1963) is not entirely clear on whether this is allowed or not (Knuth, 1967) and (ii) even if the Report can be read as allowing it, some compilers (e.g. Burroughs) do not, while others (e.g. Atlas), while apparently allowing it, fail to cancel temporary storage being used in evaluating the expression of which the function is part. This can be disastrous if many such error exits are taken within one program.

In ANSI FORTRAN, this method is not available since labels are not allowed as dummy arguments, and an assigned GO TO can go only to labels within the current subprogram, and cannot execute a return to the calling program.

In some non-standard implementations of FORTRAN, however, there is less restriction and the effect can be achieved by, for example, the

RETURN J

statement of XDS FORTRAN, where J is a dummy argument to which a label has been assigned;

(*h*) to give the function a procedure (or SUBROUTINE) parameter. This forces the user to give an actual procedure to be performed in case of failure, and may be allowed by compilers that disallow jumping out to a label. It is sometimes suggested that calling a procedure overcomes some of the difficulties associated with jumping out to a label (e.g. Herriot, 1968) but if the user can now jump out of the procedure, all the difficulties are still there, in a more pernicious form as they are more heavily disguised.

If one does not jump out of the procedure, it means returning to the original function, where one no longer wishes to be, and eventually continuing with useless calculations.

Conclusion

In my opinion, by far the most satisfactory method, where available, is that designated (*g*) above—to give the function a label parameter—in that the user cannot forget to make the test, in the absence of a fault the result is available in functional form, yet in the event of a fault useless calculations are discontinued.

The difficulties are not so much troubles with the method, as troubles with languages and compilers which have not allowed for the method. I hope that one day it will be generally agreed that this is a normal and respectable method of dealing with faults in functions, and that language writers and compiler writers should be expected to write their languages and compilers accordingly.

I may seem inconsistent in this choice, since in general I agree with Dijkstra (1968) that 'the goto statement . . . is too much an invitation to make a mess of one's program' and I always try to avoid the use of labels wherever reasonably practicable. However, in the case being discussed, the program is already a mess—a failure has occurred, and some exceptional step is needed. The shock of a 'goto', jumping away from the scene of action, to start picking up the pieces elsewhere, seems to me to be just what is required, in emphasising that this is an unusual, and unhappy, event.

Acknowledgement

I acknowledge some enlightening discussion with Peter Hammersley following an earlier draft of this paper.

References

DIJKSTRA, E. W. (1968). Go To Statement Considered Harmful, *CACM*, Vol. 11, pp. 147-148.
HERRIOT, J. G. (1968). Editor's note (within Certification of Algorithm 299), *CACM*, Vol. 11, p. 271.
KNUTH, D. E. (1967). The Remaining Trouble Spots in ALGOL 60, *CACM*, Vol. 10, pp. 611-618.
NAUR, P., *et al.* (1963). Revised Report on the Algorithmic Language ALGOL 60, *Comp. J.*, Vol. 5, pp. 349-365.
Working Party on Statistical Computing (1968). The Construction and Description of Algorithms. *Appl. Statist.*, Vol. 17, pp. 175-179.