On Iterative Constructs

DAVID LORGE PARNAS
Queen's University

The wheel is repeatedly reinvented because it is a good idea. Perhaps Anson's "A Generalized Iterative Construct and Its Semantics" [1] confirms that "A Generalized Control Structure and Its Formal Definition" [2], and the earlier "An Alternative Control Structure and its Formal Definition" [3] presented good ideas. However, there are several misstatements in [1] that should be corrected.

(1) As Anson points out, [2] contained definitions of constructs equivalent to both DO TERM and DO UPON. However, he is incorrect when he suggests that it emphasized DO TERM because of efficiency considerations. By writing "There is a pragmatic justification for either definition! ", I made it clear that that was not the reason for my choice. DO TERM has two, quite different, advantages.

   (a) DO TERM is more general. An implementation of DO TERM may, in fact, be DO UPON if desired. Further, a programmer using DO TERM can achieve the effects of DO UPON by choosing his guards accordingly.

   (b) DO TERM, like Dijkstra's **do od**, eases the verification of programs by maintaining independence of guarded commands. The verification procedure for such constructs as **do od** and DO TERM is (1) verify that the union of the guards is true in all states where the program will be invoked; (2) verify that each guarded command, on its own, will do no wrong. For DO UPON the second step is complicated by the need to consider the terminating commands in the list when considering an iterating command.

(2) Anson argues that the semantics of DO TERM are more complex. The minor syntactic difference between his two definitions is a consequence of the clumsiness of **wp** semantics. In the relational semantics used in [2], the change from DO UPON to DO TERM meant the addition of one simple definition.

(3) As Mills' [4] has explained, programmers should not be deriving the semantics of their programs from the text as Anson's analysis suggests. We do not write programs arbitrarily and then try to determine their semantics. Instead,

Author's address: Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6.

programmers should be verifying that the program they have written has the semantics that they set out to achieve. Fortunately, this verification is much easier than the inductive derivation of semantics described in [1]. As explained above, verification is easier for DO TERM than DO UPON.

(4) Anson suggests that a stronger weakest-precondition "seems to imply a weaker construct." On the contrary, DO TERM can describe algorithms that cannot be described with DO UPON.

(5) Anson also suggests that in DO TERM termination is more difficult to obtain. Programmers can obtain the behavior that they want in either. With DO TERM the guards may be longer. For those that want to reduce the length of the guards, [2] offered a third alternative, a deterministic construct. This construct forced left-to-right consideration of the commands. This alternative has the verification disadvantages of DO UPON (the guarded command semantics are not independent), but, by putting the terminating commands first, one can achieve everything that Anson values in DO UPON. In fact, with the deterministic construct, one can often achieve guards that are shorter than they would be with DO UPON. DO UPON seems to be a compromise between DO TERM and the deterministic construct, a compromise with some of the disadvantages of both extremes and the advantages of neither.

(6) Anson has not provided the full semantics of the constructs in question. It has been known for many years (e.g., Majster [5]) that **wp** alone does not define the semantics of a program. Two programs with the same **wp** can differ in their behavior in important ways. To provide a complete semantics of the constructs one must define both **wp** and **wlp**. That was one of the reasons for using a relational semantics in [2] and [3].

When I wrote [2] I deliberately chose DO TERM over DO UPON because I felt that the simplicity of verification compensated for the longer guards. I also valued the ability to describe the algorithms that cannot be described with DO UPON. I continue to prefer the syntax used in [2]. I believe that readers who consider the facts above will make the same choice.

The discussion of these issues is made a bit academic by the four-year delay between Anson's submission of his paper (which apparently coincided with the publication of [2]) and the publication of [1]. In that time a generalization of both schemes has been published as a Technical Report [6] and has been submitted for publication. In this generalization the decision about whether a command is iterating or terminating can be made during execution, and the semantics must be that of DO TERM. Further generalizations make the semantics of the constructs more practical, since side-effects are accurately treated in all cases. A method for reducing the length of guards and avoiding duplicated subexpressions is also provided.

REFERENCES

1. ANSON, E. A generalized iterative construct and its semantics. *ACM Trans. Program. Lang. Syst.* 9, 4 (Oct. 1987), 567–581.

2. Parnas, D. L. A generalized control structure and its formal definition. *Commun. ACM 26*, 8 (Aug. 1983).
3. Parnas, D. L. An alternative control structure and its formal definition. IBM Tech. Rep. TR FSD-81-0012, IBM Corp., Bethesda, Md., Apr. 1981.
4. Mills, H. D. The new math of computer programming. *Commun. ACM 18*, 1 (Jan. 1975).
5. Majster-Cederbaum. A simple relation between relational and predicate transformer semantics for non deterministic programs. *Inf. Process. Lett. 11*, 4, 5 (Dec. 12, 1980).
6. Parnas, D. L. Less restrictive constructs for structured programs. Tech. Rep., Queen's Univ., Kingston, Ont., Sept. 1986.