```
[abs "a" : abs "z"] nng stg;

proc int prog = (prog p, nng input) nng : (
  for i from abs "a" to abs "z" do stg[i] := 0 od;
  stg [abs readvar of p] := input;
  int cmd (body of p);
  int expr (writeexpr of p));

proc int cmd = (cmd cm) void :
  case cm in
    (seq se) : (int cmd (cmd1 of se); int cmd (cmd2 of se)),
    (asst as) : stg [abs lhs of as] := int expr (rhs of as),
    (loop lp) : to int expr (lim of lp) do int cmd (body of lp) od
  esac;

proc int expr = (exp ex) nng :
  case ex in
    (int) : 0,
    (var v) : stg [abs v],
    (succ su) : int expr (opd of su) + 1
  esac
```

Fig. 4 Operational definition in ALGOL 68

language definer in achieving a complete, consistent, and 'bug-free' specification, for language defining and programming are actually very similar activities in certain respects. Moreover, if the defining language is a 'mainstream' language like ALGOL 68 as opposed to a specially invented notation or a primitive scheme like pure LISP (where the lack of strong type checking is a serious disadvantage), then the definition will be more acceptable and understandable to the community of implementors and users. In this connection, it is important that the executable definition be comparable to or possibly better than a purely formal definition with respect to qualities such as readability and conciseness. In the author's opinion, this is achievable in the case of ALGOL 68 with partial parametrisation but not in the case of strict ALGOL 68 (and certainly not in the case of other 'mainstream' languages); it is to be hoped that future ALGOL 68 compilers will permit the use of this additional feature.

When viewed purely as an implementation of LOOP, the specifications in Fig. 3 are indeed strange, arcane and extremely inefficient, and the specifications using strict ALGOL 68 are even more so. The latter have been tested with the aid of an ALGOL 68 compiler; as expected, they will slowly but successfully interpret simple LOOP programs with small input values. For example, interpretation of the program given earlier for computing the function $\lambda x \,.\, 2x + 2$ with input value 2 involves about 70 function calls with a maximum recursion depth of 18. The lack of clarity and usefulness of Fig. 3 *as a processor* as opposed to a *definition* is not surprising, since denotational definitions are much less implementation oriented than operational definitions. If we had wanted the definition to constitute a clear and useful processor (probably at the expense of other uses such as proving correctness of programs), we should have taken an operational approach and written specifications such as those of **Fig. 4.**

References

ANDERSON, E. R., BELZ, F. C. and BLUM, E. K. (1976). SEMANOL (73) A Metalanguage for Programming the Semantics of Programming Languages, *Acta Informatica*, Vol. 6, pp. 109-131.

HOARE, C. A. R. and LAUER, P. E. (1974). Consistent and Complementary Formal Theories of the Semantics of Programming Languages, *Acta Informatica*, Vol. 3, pp. 135-153.

LINDSEY, C. H. (1974). Partial Parametrization, *ALGOL Bulletin*, No. 37, pp. 24-26.

LINDSEY, C. H. (1976). Specification of Partial Parametrization Proposal, *ALGOL Bulletin*, No. 39, pp. 6-9.

MOSSES, P. D. (1975). The Semantics of Semantic Equations, Proc. 3rd Symp. on Mathematical Foundations of Computer Science, Springer-Verlag Lecture Notes in Computer Science, No. 28, pp. 409-422.

MOSSES, P. D. (1976). Compiler Generation using Denotational Semantics, Proc. 5th Symp. on Mathematical Foundations of Computer Science, Springer-Verlag Lecture Notes in Computer Science, No. 45, pp. 436-441.

PAGAN, F. G. (1976). On Interpreter-Oriented Definitions of Programming Languages, *The Computer Journal*, Vol. 19, pp. 151-155.

PAGAN, F. G. (1977). ALGOL 68 as an Implementation Language for Portable Interpreters, Proc. Strathclyde ALGOL 68 Conf., *SIGPLAN Notices*, Vol. 12, No. 6, pp. 54-62.

RAYWARD-SMITH, V. J. (1977). Using Procedures in List Processing, Proc. Strathclyde ALGOL 68 Conf., *SIGPLAN Notices*, Vol. 12, No. 6, pp. 179-183.

REYNOLDS, J. C. (1972). Definitional Interpreters for Higher-Order Programming Languages, Proc. 25th ACM National Conf., pp. 717-740.

TENNENT, R. D. (1976). The Denotational Semantics of Programming Languages, *CACM*, Vol. 19, pp. 437-453.

VAN WIJNGAARDEN, A. *et al.* (1976). Revised Report on the Algorithmic Language ALGOL 68, Springer-Verlag, also in *Acta Informatica*, Vol. 5, Parts 1-3 (1975) and *SIGPLAN Notices*, Vol. 12, No. 5, pp. 1-70 (1977).

Book review

Structural Analysis and Design (Volumes 1 and 2), 1978. Infotech, £120)

The first volume is an analysis and bibliography of the processes of analysis and design. It takes the form of a series of extracts from many sources, including volume 2, linked by an editorial commentary. This format is very effective and it succeeds in demonstrating both the need for a methodology and also the schools of thought which lead to the competing methodologies. SADT, Michael Jackson Methodology, Warnier-Orr Methodology and Structured Design are each concisely and clearly described.

The second volume contains 19 invited papers, the first of which, by R. R. Brown, includes a productivity analysis of two application developments at Hughes Aircraft in 1974 and 1972/3: One of the rare published case studies. An elegant paper by S. N. Griffiths compares methodologies and finds much merit in Michael Jackson. It also takes an interesting perspective view of the current structured design scene together with some predictions for the future.

The book makes a valuable contribution to current literature, and provides a good starting point for a study of structured design. It won't be much help to the application practitioner looking for guidance in selecting and introducing a methodology in his own shop but will sustain the current debate. It is a report not so much on the state of the art, but more on the state of mind of the artists.

K. BOARDMAN (London)