ISSN 0084-6198

Algol Bulletin no. 47

AUGUST 1981

CONTENTS

LIBRARIANS please take note.

There is a microfiche enclosed with this issue. Please file it away wherever you keep microfiches, and write your catalogue number for it here:

The ALGOL BULLETIN is produced under the auspices of the Working Group on ALGOL of the International Federation for Information Processing (IFIP WG2.1, Chairman Robert B. K. Dewar, Courant Institute).

Facilities for the reproduction of the Bulletin have been provided by courtesy of the John Rylands Library, University of Manchester.

Facilities for the reproduction of the microfiche enclosed with this issue have been provided by the courtesy of the Rechenzentrum of the Ruhr-Univeritaet, Bochum.

The ALGOL BULLETIN is published approximately three times per year, at a subscription of $10 per three issues, payable in advance. Orders and remittances (made payable to IFIP) should be sent to the Editor. Payment may be made in any currency (a list of acceptable approximations in the major currencies will be sent on request), but it is the responsibility of each sender to ensure that cheques etc. are endorsed, where necessary, to conform to the currency requirements of his own country. Subscribers in countries from which the export of currency is absolutely forbidden are asked to contact the Editor, since it is not the policy of IFIP that any person should be debarred from receiving the ALGOL BULLETIN for such a reason.

The Editor of the ALGOL BULLETIN is:

Dr. C. H. Lindsey,
Department of Computer Science,
University of Manchester,
Manchester, M13 9PL,
United Kingdom.

Back numbers, when available, will be sent at $4 each. However, it is regretted that only AB32, AB34, AB35, AB36, AB38, AB39, AB40, AB41, AB42, AB43, AB45, and AB46 are currently available. The Editor would be willing to arrange for a Xerox copy of any individual paper to be made for anyone who undertook to pay for the cost of Xeroxing.

AB47.0 EDITOR'S NOTES.

First, my usual remarks about paucity of contributions, and hence the thinness of this issue. However, you do get your free plastic insert again. This time, it contains the complete Revised Report (together with the Commentaries collated from AB43.3.1 and AB44.3.1), the Sublanguage Report, and the Standard Hardware Representation. These are the three official documents, approved by I.F.I.P. Also included are the Partial Parametrization Proposal (AB39.3.1) and the Modules and Separate Compliation Proposal (AB43.3.2).

Standardization of ALGOL 68.

Following a suggestion from the ISO committee concerned with programming languages (ISO TC97/SC5), Working Group 2.1 decided, at its meeting in August 1980, to press for an International Standard on ALGOL 68. The Standard will be prepared jointly by IFIP and ISO, and I have been appointed to coordinate these activities.

The intention is to leave the Revised Report intact as the definition of the language. The Standard will refer to the Report, and will prescribe precise requirements for conforming programs, implementations and accompanying documentation. I hope to publish a working draft of the standard in a future edition of the ALGOL Bulletin.

In the meantime, there are many political problems to overcome. We have to persuade sufficient National Standards Organisations to vote, first to have the proposed standard even considered, and later to have it approved. The danger is that they may be unaware of the interest in ALGOL 68 within their respective countries, and that they will refuse to take it seriously. Here is where each of you, if you believe this to be a worthwhile endeavour, can help. Write to your National Standards Organization. Tell then that ALGOL 68 matters; that people really do use it; and that a Standard therefore ought to exist.

AB47.1 Announcements.

AB47.1.1 International Symposium on Algorithmic Languages.

This Symposium is organized by the Mathematical Centre under the auspices of IFIP TC2 as a tribute to Professor A. van Wijngaarden on the occasion of his retirement from the Mathematical Centre. Professor van Wijngaarden is well known for his contributions in the area of programming language design (ALGOL 60, ALGOL 68, two-level grammars). The Symposium is to be held from Oct. 26-29, 1981 at the Free University of Amsterdam, in The Netherlands.

Program:

H.Zemanek (IBM, Vienna): "The role of Professor van Wijngaarden in the early history of IFIP", (invited address).

A.I.Wasserman (University of California, San Francisco), R.P.van de Riet and M.L.Kersten (Free University, Amsterdam): "PLAIN, an algorithmic language for interactive information systems".

R.Schild (Landys & Gyr, Switzerland): "PORTAL, a process-oriented real-time algorithmic language".

J.D.Roberts (University of Reading): "Naming by colours: a graph-theoretic approach to distributed structure".

H.S.Warren, Jr. (IBM, Yorktown Heights): "Optimization of inductive assertions".

A.Bossavit and B.Meyer (Electricite de France, Clemart): "Methods for vector programming".

P.Klint (Mathematical Centre, Amsterdam): "Formal language definitions can be made practical".

J.Backus (IBM, San Jose): "Is computer science based on the wrong fundamental concept of "program"? An extended concept", (invited address).

L.G.L.T.Meertens (Mathematical Centre, Amsterdam): "Issues in the design of a beginners' programming language".

D.Grune (Mathematical Centre, Amsterdam): "From VW-grammar to ALEPH".

M.Broy, P.Pepper and M.Wirsing (Technical University of Munich): "On design principles for programming languages: An algebraic approach".

J.Darlington (Imperial College, London): "Structured descriptions of algorithm derivations", (invited address).

M.Sato and M.Hagiya (University of Tokyo): "HYPERLISP".

D.de Champeaux and J.de Bruin (University of Amsterdam): "Symbolic evaluation of LISP functions with side effects for verification".

P.Naur (University of Copenhagen): "Aad van Wijngaarden's contributions to ALGOL 60", (invited address).

M.M.Fokkinga (Technical University of Twente): "On the notion of strong typing".

H.B.M.Jonkers (Mathematical Centre, Amsterdam): "Abstract storage structures".

J.C.Reynolds (Syracuse University): "The essence of ALGOL", (invited address).

R.Kuiper (Mathematical Centre, Amsterdam): "An operational semantics for nondeterminism equivalent to a denotational one".

O.Grumberg, N.Francez, J.A.Makowsky (Technion, Haifa) and W.P. de Roever (University of Utrecht): "A proof rule for fair termination of guarded commands".

W.M.Turski (University of Warsaw): ALGOL 68 revisited twelve years later, or: from Aad to Ada", (invited address).

A full social program has also been arranged.

Full details concerning registration, fees and hotel accomodation may be obtained from:

Mrs.S.J.Kuipers,
Mathematical Centre,
Postbus 4079,
1009 AB Amsterdam,
The Netherlands.

AB47.1.2 ALGOL 68 Implementation - FLACC.

Some recent adjustments in the pricing structures for the FLACC system may be of interest.

These changes are primarily aimed at helping those interested in Algol 68 by making a high-quality implememtation generally available. We particularly wish to help people who want to learn and to use Algol 68, but who cannot justify the FLACC lease price.

We propose therefore, to distribute an unsupported version of FLACC. The frozen version is designated "FLACC V1.4U", and is available to universities and technical schools for a one-time charge of C$1500. This price does not include maintenance or update services, nor is there any provision for an acceptance period.

For complete licensing information, please write to:

Sigrid Fritz,
Chion Corporation,
Box 4942,
Edmonton, Alberta,
CANADA T6E 5G8.

AB47.1.3 International Conference on ALGOL 68 - Proceedings.

The Proceedings of the International Conference on ALGOL 68, held at Bochum on March 30-31 1981, under the sponsorship of the WG2.1 Subcommittee on ALGOL 68 Support and of the Rechenzentrum der Ruhr-Universitaet, can be obtained from:

Mathematical Centre,
Postbus 4079,
1009 AB Amsterdam,
The Netherlands.

Table of Contents:

On the design of an abstract machine for a portable ALGOL 68 compiler, by L.G.L.T.Meertens.

An implementation of modular compilation in ALGOL 68 (invited lecture), by G.J.Finnie & M.C.Thomas.

Programming languages for a course in data structures, by V.J.Rayward-Smith.

Context-free grammars and derivation trees in ALGOL 68, by V.Linnemann.

An ALGOL 68 prelude for the implementation of test generation algorithms, by S.D.Butland.

A programming system for interval arithmetic in ALGOL 68, by G.Guenther & G.Marquardt.

Teaching with ALGOL 68, in Manchester (invited lecture), by C.H.Lindsey.

The price of the proceedings, published as Mathematical Centre Tract 134, is Dfl 29.40 (including VAT). Foreign payments are subject to an additional Dfl 6.50 per remittance to cover bank, postal and handling charges. Forwarding of the publication(s) ordered from abroad will take place on receipt of remittance. Payment should be made in Dutch currency or its equivalent.

AB47.1.4 An Axiomatic Semantics for Expression Languages.

This is the Ph.D. Thesis of P.A.Pritchard, as submitted to the Australian National University. Subject to availability, copies may be obtained by writing to:

Professor P.A. Pritchard,
Department of Computer Science,
Cornell University,
Ithaca, New York 14853,
U.S.A.

The work is closely related to that on ALGOL 68 by Richard Schwartz (see AB44.1.5).

ABSTRACT

This thesis addresses the problem of giving Hoare-style axiomatic definition of the semantics of expression-oriented block-structured programming languages. This problem is tackled per medium of an exemplary expression language El which caters for the manipulation of both l-values and r-values.

A notational extension is presented which allows the effects of state-changing expressions to be naturally described, and a formal Hoare-style program logic D is then given which defines the partial correctness semantics of El.

Proofs of the consistency and completeness (in the sense of Cook) of D are obtained by a novel method involving a translation of El programs in an underlying statement-oriented language. This method enables clear comparisons to be made of the two styles of programming language.

It is shown that efficient syntax-driven program verification is possible

for El in both of the major styles, viz. backward substitution and symbolic execution, but that the latter style is more natural when l-values are manipulated.

Finally, the above mentioned work on El is related to and compared with other work in the literature concerned with "side-effects" in conventional languages, and Schwartz's closely related work on ALGOL 68 is examined in some detail.

AB47.1.5 An ALGOL 68 Indenter - Correction.

The following correction should be made to fix a bug in the ALGOL 68 indenter given in AB46.4.4. The bug only affects programs which contain an exit as a constituent of a serial-clause which is enclosed between brief delimiters.

AB46 p.34 "STATE=MIDDLER"+10

```
# THEN GAP => AND (CLAUSE<>EXIT) THEN GAP #
```

AB47.1.6 Book Review : Intermediate Language for Graphics.

by P.J.W.ten Hagen et al.
Mathematical Centre Tracts 130. ISBN: 90 6196 204 8

This text contains the definition of a special purpose data description language for pictures, the Intermediate Picture Language (ILP). The authors envisage ILP emdebbed in a high level programming language to provide, for example, variables, conditionals and loop constructs. All the parameters to ILP are constants. No details of this embedding are given. ILP is a high level plotfile and again no advice is offered on mapping ILP onto a plotfile. They do not envisage a FORTRAN implementation, and having eliminated the constraints such a language would imply has enabled the six authors to create a graphics language pleasant to read.

The essential construct of ILP is:

```
<picture>:: PICT (<dim>) <pname> <PE>
<PE>      :: {<picture element>} | {WITH <aname> DRAW <pname>}
```

The picture once defined and stored is subsequently referred to via its name <pname>. <dim> specifies the dimensions of the coordinates used in the picture elements.

Dimensions >4 seem to be unnecessary; (-7 is also syntactically correct)! The picture elements (lines, text, connected lines or curves) must all be of this dimension (unless a subspace has been selected). The authors do not indicate actions in the event of a conflict between the picture elements and the picture dimension. The simplest picture is a collection of picture elements. This simple picture may be invoked as part of another picture using the WITH <aname> DRAW <pname>. <aname> is the name of an attribute pack, a collection of attributes to be applied to <pname>. Attributes include line style, colour and width. More interesting attributes are the transformation control, specified either in basic operations such as translate, rotate, scale, or perspective projection or through a complete (affine or homogeneous) transformation matrix. The number of parameters required is determined by the current dimension specified as part of the picture header.

Since pictures invoke other pictures, mixing rules for the attributes are

necessary; Chapter 3 deals with attribute concatenation rules and semantics of ILP. Coordinate transformation matricies are multiplied together and, for example, for line style patterns the latest style invoked is used. The feature of ILP to be able to present the same picture on different output devices with differing attributes is ideal in a graphics environment.

The fourth and final chapter of the book is devoted to a discussion of the group's design goals and how far they have been able to achieve them. ILP was implemented in 1979 as a compiler and interpreter. The compiler checks the syntax and produces an efficient coding of the programs (a plotfile?) for the interpreter which drives the drawing machine. 60k bytes on a PDP11/45 for a plotfile spooler (be it so versatile) seems rather large!

Appendices one and two contain the syntax rules in BNF, though a book on graphics should have produced a graphic representation of them as for example in books on ALGOL68, and Pascal. The page numbers indicated in the index for the appendices are also one short! The last appendix is an example program, drawing a house and makes full use of ILP to produce a pythagoras tree for the curtains.

W.T.Hewitt,
Computer Graphics Unit, University of Manchester.

AB47.2 Letters to the Editor.

AB47.2.1 ALGOL 68 Syntax Chart on Microfiche.

135, East MAIN,
Apt. R6,
Westboro Ma. 01581,
U.S.A.

16th. Jan. 81.

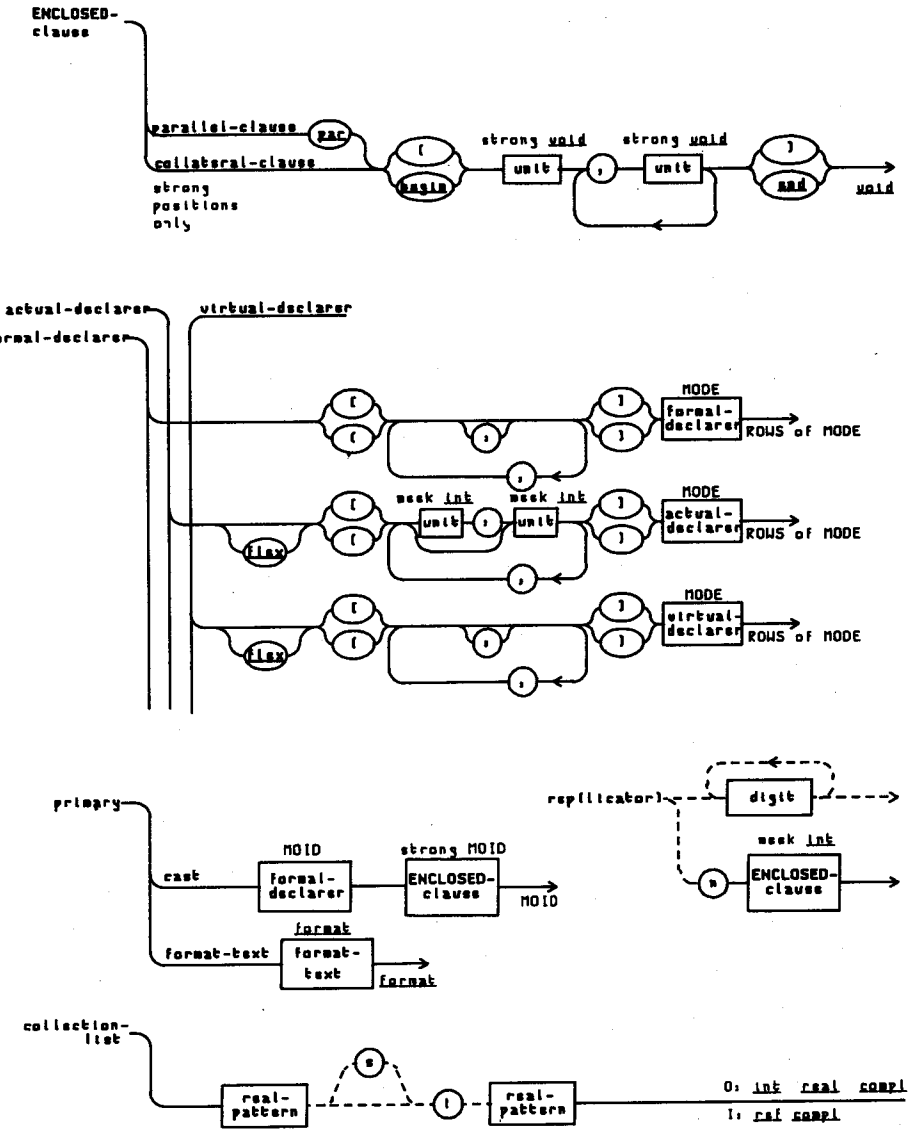
Dr. Lindsey.

I recently received Algol Bulletin 46. I believe the following are errors in the syntax charts on the enclosed microfiche. Parallel and collateral clauses are missing, casts and format texts are not listed among the units, the possible suppression in a complex frame is not shown, pragments after the insertion of a boolean pattern are allowed, and only a single digit is allowed in a replicator. I hope this arrives in time to be helpful.

Keith Wright.

Reply by C.H.Lindsey:

I plead guilty on all counts. Also some further omissions in rowers were pointed out by Mr.G.Kaniuk. The casts and the format-texts were accidents during editing. The others are mostly features which I have never used, and certainly never teach. The diagrams below show how the offending items should have looked. I have prepared a corrected version of the microfiche. If any reader would like one, please send me a fiche-sized envelope addressed to yourself (readers in the U.K. are invited to supply a second-class stamp also).



AB47.2.2 On the Efficiency of ALGOL 68 Transput.

Katholieke Hogeschool Tilburg,
Postbus 90153, 5000 LE Tilburg,
The Netherlands.

In our computer centre we use for writing application programs mainly ALGOL68 for about 6 years (first ALGOL68R, since two years the ICL 2900 compiler). I think we can say that we are quite happy with the language and the compilers. Only transput is not exactly what we need.

One of the intentions of the design of ALGOL68 was that it "allows the programmer to specify programs which we can be run efficiently on present day computers" [1].

I think that this does not always apply to the ALGOL68 transput; some features are not efficient, and this might be due to the definition.

1. - In Tilburg we found that character transput in ALGOL68 is slow, when compared with FORTRAN.
   - Reading 1000 records gives 5.5 seconds, FORTRAN 2.9, on an ICL 2960 computer.
   - Printing reals is also slower in ALGOL68 than in FORTRAN (5.4 seconds for a [1:66,1:5] real, compared with 1.0).
   - Some measurements on formatted transput seem to indicate that formatted transput is slow compared with FORTRAN.

2. Transput of the CDC ALGOL68 compiler is also slower than FORTRAN [2, table III and IV]. I do not know figures of other implementations, but I am quite interested.

3. A different definition of some aspects of the transput might come closer to out needs.
   - The transput is not record oriented enough.
   - Reading strings to the end of a line (the normal case in terminal applications) is slow because of the possibility of 'make term'. If 'make term' was not possible, reading strings could be implemented with more efficiency.
   - Routines like 'whole', 'fixed' and 'float' are very useful. The reverse functions (making a number out of a string) are hidden for the user. Although they are easy to program, these functions should have been defined.
   - Formatted transput is hardly used in Tilburg (somebody said it is a language on its own; we do not want to teach or use two languages); the needed functions can easily be defined as character and string operators and routines.

The question is left what other users of ALGOL68 think of the transput. I am sure that implementors would not like redefinition, but on the other hand, the usability of ALGOL68 could be improved.

Sincerely yours,
Joop Coumou.

P.S. In your recent article (3) there is the statement "The indenter is written in PASCAL (-... it needs to run efficiently...)". I do not know whether this is due to transput, but any how the readers of the ALGOL Bulletin could be interested in the arguments why PASCAL is more efficient for this application.

1. A. van Wijngaarden et.al. Revised Report on the Algorithmic language ALGOL 68.

2. H.J. Boom and E.de Jong. A critical comparison of Several Programming Language Implementations, Software Practice and Experience 10(1980) p435-473.

3. C.H.Lindsey. An ALGOL68 Indenter, AB46.4.4.

Editor's reply:

It may just be that the implementation of your ALGOL 68 transput is badly written. Certainly, it is known that the transput in the CDC compiler was the minimum that would guarantee correct operation, regardless of efficiency. An implementation based on the "Hansput" model (see AB44.1.1) could be expected to do much better. Indeed, an implementation of an early version of the Hansput (see AB46.4.2) on a TR440 actually ran faster than the FORTRAN implementation on that machine (but perhaps it was just a badly written FORTRAN implementation).

I think 'make term' can be acceptably efficient. In my own ALGOL 68S implementation (see AB47.3.3) we followed the Hansput model in the main, but implemented the terminating strings as bit-maps (PASCAL sets, actually), and placed the detailed inspection of them at the lowest possible level in the input primitives, so as to avoid procedure calls on a one-per-character basis. The same primitives were used for input of numbers by making, for example, the terminating string to be 'all the character set except for the digits'. However, my ALGOL 68S implementation, intended primarily for teaching, makes no claims to be efficient at run time (it is largely interpretive), which is why I wrote my indenter in PASCAL.

I agree that the ALGOL 68 transput is not record oriented enough, but what is missing is a completely separate keyed "record transput" facility, which ought to exist in parallel with the present "character transput" and "binary transput". Indeed Mr.Coumou also sent me some information on such a package for "indexed sequential" transput, which has been written at Tilburg. It takes the form of an ALGOL 68RS Album, and operates on ICL VME/B indexed sequential files as understood by, for example, the ICL 2900 COBOL compiler. I am sure that this system could be made available to other users of ALGOL 68 on 2900s.

AB47.3 Working Papers.

AB47.3.1 Errata to "A Modules and Separate Compilation Facility for ALGOL 68".

The following errata were authorized by the ALGOL 68 Sub-Committee of IFIP WG2.1 on May 12 1981. They are to be applied to:

"A Modules and Separate Compilation Facility for ALGOL 68" by
C.H.Lindsey and H.J.Boom,

as published in the ALGOL Bulletin AB43.3.2
and also as Mathematisch Centrum Report IW 105/78.

Errors in Formal Definition.

4.9.1.b+1 # DECSETY invoked => DECSETY INKSETY invoked #

{4.9.1.b+3 is ambiguous; the rule may succeed with any TAU1 such that TAU1 TALLY = TAU.}

1.2.3.R # TAU :: MU. => TAU :: MUum. #

3.6.1.h+3,h+5 # {h,i} => {h,i,-} #

Errors in Implementation Methods.

2.5+9 # HOLE => NEST #

{In section 1.1, the grammar (although formally correct) should be brought into line with the corresponding grammar in the Formal Definition}.

1.1+1:1.1+17 # ??? =>

```
compilation input:
    prelude packet &
       imposed module interface option &
       joined module interface {for definition modules, if any,
          accessed by this one};
    definition module packet &
       imposed module interface option &
       joined module interface {for definition modules, if any,
          accessed by this one} &
       hole interface;
    particular program &
       joined module interface {for definition modules, if any,
          accessed by the particular program};
    stuffing packet &
       hole interface &
       joined module interface {for definition modules, if any,
          accessed by the stuffing}.

source packet:
    prelude packet {a module-declaration within the standard environment};
    definition module packet {a module-declaration within
          a specified hole};
    particular program {a stuffing within the standard environment};
    stuffing packet {a stuffing within a specified hole}.
#
```


AB47.3.2 Errata to "ALGOL 68 Transput, part II: An Implementation Model.

The following errata were authorized by the ALGOL 68 Sub-Committee of IFIP WG2.1 on May 12 1981. They are to be applied to:

"ALGOL 68 Transput, Part II: An Implementation Model" by
J.C.van Vliet (see AB44.1.1),

as published as Mathematical Centre Tracts 111
and also in J.C.van Vliet's doctoral thesis.

1. {Typing error}
   p47+7: # intercative => interactive #

2. {Improper handling of the logical file end in the primitives 'do newline' and 'do newpage' at page overflow}
   p47c)+11, p48d)+30:
   At the end of a call of 'do newline' or 'do newpage', the buffer is newly initialized if one is currently writing to the book. The logical end as recorded in the book then still is at the end of the previous line (because of the preceding call of 'write buffer'). In general, this is no problem, since the 'write back'-flag is raised by 'init buffer' if one is writing, so the buffer will eventually be written back, also resulting in the proper updating of the logical file end information. However, the 'write back'-flag is not (and should not be) raised if a page overflow is detected by 'init buffer'. In that case, an immediately following call of a routine like 'close', 'set' or 'reset' will not properly set the logical end (since no call of 'write buffer' results). Therefore, the following change should be made in both routines:

```
# (init buffer OF cover)(f) =>
  (init buffer OF cover)(f);
  IF status OF cover SUGGESTS page end AND
     status OF cover SUGGESTS lfe in current line
  THEN set logical pos(f)
  FI
#
```

3. {Improper handling of the logical end in 'do newpage' at file overflow}
   This error is similar to the one reported above, and so is the remedy:
   p48d)+27:

```
# THEN status =>
  THEN set logical pos(f);
    status
#
```

4. {Interchanging lines in 'open'}
   p69+12/13:
   Since a call of 'set write mood' may lead to a call of 'init buffer', the buffer primitives ought to be made available first. Therefore, these two lines should be interchanged.

5. {Improper handling of the physical file end in 'associate'}
   p70+25, p72+19:
   It must be recorded that for associated files the physical file is also ended if the logical file is (on reading). (Otherwise a bounds error may occur after a change to write mood.) Therefore:

```
  # logical file ended =>
    logical file ended AND physical file end #
```

6. {Improper updating line end information in 'set write mood'}
p74+15:
In various routines, like 'space' and 'get char', the setting of the 'logical file end'-flag has preference over that of the 'line end'-flag, though both may occur at the same position. Therefore, in case the 'logical file end' is undone by a change to write mood, the test for the line being ended must still be made.

```
  # IF =>
    IF c OF cpos OF cover > char bound OF cover
    THEN status ANDAB line end
    FI;
    IF
  #
```

7. {Wrong pragmatics for the operator 'EXPLENGTH'}
p114h)+2: This line should obviously read:

```
C The smallest E > 0 such that 'whole(exp, (sign ¦ E ¦ -E))' succeeds. This
```

8. {Typing error}
p116+8: # o IF => o If #

9. {Re-initializing the buffer after a call of 'char error mended' in 'get char'}
p140d)+17:
The test 'status SAYS line ok' also fails when the buffer is not initialized. At various places inside 'get', however, it is only intended to test for a proper line end or logical file end (e.g., when reading numbers or strings). Therefore, these tests should be guarded against failure because of a non-initialized buffer. Remedy:

```
  # ; mended =>
    ;
    IF NOT (status OF cover OF f SAYS buffer initialized)
    THEN (init buffer OF cover OF f)(f)
    FI;
    mended
  #
```

10. {Small oversights after a late change of the notion 'FRAME' to 'DFRAME'}
p156g)+4, h)+1, P157i)+1, j)+1:

```
  # FRAME => DFRAME #
```

11. {Bug in 'update cp'}
p166+14:

```
  # [cp OF piece] => [cp OF piece - 1] #
```

12. {Typing error in 'do fpict'}
p173+2:

```
  # UPB i => UPB i1 #
```

13. {Oversight of ?}
p188+1:
Obviously, 'indit string' is a routine hidden to the user, so:


```
  # PROC indit string => PROC ? indit string #
```

14. {Re-initializing the buffer while reading according to a 'choice-pattern'}
p197+15:
This error is similar to the one in 'get char': the subsequent test 'status SAYS line ok' is only meant to test for a proper line end or logical file end. The remedy is the same as the one given under entry 9 above:

```
  # BOOL =>
    IF NOT (status OF cover OF f SAYS buffer initialized)
    THEN (init buffer OF COVER OF f)(f)
    FI;
    BOOL
  #
```

15. {Typing error in 'getf'}
p198+10:

```
  # (REF INT i) i:= k =>
    (REF INT i): i:= k #
```

16. {Syntax error in 'get bin'}
p204+24:
Both the assignment and the second parameter of the call of 'from bin' are syntactically incorrect. The intention is better phrased as follows:

```
  C The yield of 'from bin(f, itk, bin)' is assigned to the yield
    of 'it[k]', where 'itk' is the value referred to by the yield
    of 'it[k]' C
```

17. {Omissions of * in the Index}
p210/213:
A * is missing in the entries for 'associated format', 'do fpict' and 'get next picture'. (These routines all use special generators which cannot be properly expressed in ALGOL 68.)

AB 47.3.3 Survey of Viable ALGOL 68 Implementations

This survey, which we hope to republish from time to time as further information becomes available, has been restricted to implementations which you can actually obtain and use. Each of them is already in use on at least two sites, and has an identifiable person or organisation responsible for its maintenance.

Most of the column headings are self-explantory. "Deviations" means that it is possible to write some program, valid and with defined meaning both in the given implementation and according to the Revised Report, which will provide results different from those defined by the Revised Report. Under "Money", "nominal" means under $200, "yes" means a realistic commercial rate. "MC Test Set" means that it has been tested using the MC Test Set (see AB 44.1.2) and that the implementor claims that it ran correctly. In all cases, the people listed in the last column should be able to provide further information.

Name of System	Hardware	Operating System	Principal Sublanguage features	Principal Superlanguage features	Deviations?	Money?	MC Test Set?	Other features	Where to obtain it
FLACC	IBM 370 Amdahl Siemens	OS/VS/MVS /MFT/MVT CP/CMS MTS			No	Yes	Yes	load and go version available very complete checking slow running	Chion Corporation Box 4942, Edmonton Alberta CANADA T6E 5G8
ALGOL 68C Release 1	IBM 360 IBM 370	OS/MVT OS/VS2 OS/MVS OS/MFT OS/VS1	no sema no flex no format restricted transput no garbage collector scopes not checked	automatic op:= for any op upto, downto and until in loop-clauses displacement operator (:=:=) andf, orf and thef	Yes	Nominal to Universities	No	separate compilation fast running	ALGOL68C Distribution Service Computer Laboratory Corn Exchange Street CAMBRIDGE CB2 3QG United Kingdom
		CMS							Dr. S. Chidlow/Dr. R. Hill University Computing Service The University of Leeds LEEDS LS2 9JT United Kingdom
	DEC-10 DEC-20	TOPS-10 TOPS-20							Dr. R. G. Blake Computing Service University of Essex Wivenhoe Park COLCHESTER CO4 3SQ United Kingdom
	VAX								ALGOL68C Distribution Service (see above)
	Telefunken TR440 TR445		improved transput available						H. Wupper Rechenzentrum der Ruhr-Universitaet Postfach 102148 ,D-4630 Bochum FEDERAL GERMAN REPUBLIC
	Prime								Dr. E. F. Elsworth Dept. of Computer Science University of Aston in Birmingham BIRMINGHAM, United Kingdom

Name of System	Hardware	Operating System	Principal Sublanguage features	Principal Superlanguage features		Devi-ations?	Money?	MC Test Set?	Other features	Where to obtain it
	TELSA 200 (similar to IBM 360)		no flex (except string) no union no sema no heap no exit	bounds in formal-declarers		No	No	No	TRACE facility independent compilation of routines fast running	J. Nadrchal Institute of Physics Czechoslovak Academy of Sciences 180 40 Praha 8, Na Slovance 2 CZECHOSLOVAKIA
CONTROL DATA ALGOL68	CDC 6000 -7000 170 series	NOS 1 NOS/BE SCOPE2	one long flexibility is an attribute of a multiple value	no transient name restriction icf macros allow definition of operators in machine instructions		No	Yes	Yes	separate compilation	Control Data Services B.V. P.B. 111 Rijswijk (24) THE NETHERLANDS
A68S	CDC Cyber	NOS 1 NOS/BE SCOPE 2.1	official sublanguage (see SIGPLAN Notice 12 5 May 1977 or Informal Introduction Appendix 4) but heap is allowed			No	Nominal	No	very complete checking fast compilation slow running	C. H. Lindsey Department of Computer Science University of Manchester MANCHESTER M13 9PL United Kingdom
A68RS	ICL 2900	VME/B VME/K	indicators to be declared before use no sema scopes not checked	mode vector indexable structures forall elements of array no transient name restriction		Yes	Yes	Yes	modular compilation	ICL local sales office
	UNIVAC 1100 series	EXEC-VIII	no garbage collector scopes not checked	bin of any primitive mode complex mathematical functions min and max matrix and vector operators		Yes	Nominal?	Yes	French representations (inhibitable by pragmat) independent compilation of routines	Daniel Taupin Laboratoire de Physique des Solides Universite de Paris XI 91405 Orsay FRANCE

AB47.4.1

Self-Replicating Programs and n-Cycle Programs.

Chris Thomson
Chion Corporation
Box 4942
Edmonton, Alberta, Canada T6E 5G8

After meeting Martyn Thomas at a Working Group meeting, I posed his problem of a short self-replicating program to my partner, Colin Broughton (fellow author of FLACC). I could remember only that the program Martyn told me about used print and two times a string.

Independently, Colin came up with the same program as appeared in AB46.2.1. However, he used the plus operator and only single parentheses in the print call, so his solution is four characters shorter than Wendland's. He then developed the following version, which works in all three of the standard stropping regimes (POINT, RES and UPPER):

```
(.STRING a="(.STRING a="";print(2*a[:12]+2*a[12:]))";
  print(2*a[:12]+2*a[12:]))
```

Of course, it too must be input as a single line.

Never content with a special case, Colin then went on to pose the more general problem of writing a cycle of programs, each of which produces the next (as in A produces B which produces A). He came up with the following pair of 2-cycle programs, which also work with any stropping regime, and must be input as single lines (the only spaces are those following STRING):

```
A: (.STRING a="(.STRING a="";2*b[64:75]+b[75:]+b[:19]+2*b[19:63]))
     (print((.STRING b="";print(2*a[52:70]+a[70:]+a[:12]+2*a[12:51]))";
     print(2*a[52:70]+a[70:]+a[:12]+2*a[12:51]))

B: (print((.STRING b="(print((.STRING b="";print(2*a[52:70]+a[70:]+
     a[:12]+2*a[12:51]))(.STRING a="";2*b[64:75]+b[75:]+b[:19]+
     2*b[19:63]))";2*b[64:75]+b[75:]+b[:19]+2*b[19:63]))
```

Note that program A is of the form (STRING a="...";print(...)), while B is of the form (print((STRING b="...";...))).

Later, I posed the general problem of an easily modified program which would cycle after an arbitrary n to Danny Boulet, a friend at the University of Alberta computer centre. He came up with the 3-cycle program which follows:

```
(.INT i=(0+1)%*3;.STRING a="(.INT i=(0+1)%*3;.STRING a="";
  print(2*(a[:9]+whole(i,0)+a[11:28])+2*a[28:]))";
  print(2*(a[:9]+whole(i,0)+a[11:28])+2*a[28:]))
```

As always, this must be a single line of input. It can be made into a 9-cycle program by changing the two %*3's to %*9's. Each program in the cycle has a different value of i (the 0+1 changes to 1+1, etc.).

Carrying the spirit of oneupmanship a step further, he wrote the following program:

```
(.INT m=  1;.STRING a="(.INT m=  1;.STRING a="";
  print(2*(a[:8]+whole(.ENTIER(1+m*997*random)%*1000,-3)+
  a[12:23])+2*a[23:]))";print(2*(a[:8]+whole(.ENTIER (1+m*997*
  random)%*1000,-3)+a[12:23])+2*a[23:]))
```

which is intended to reproduce itself after a random-length cycle of expected length 1000. Note that the two spaces following the m= are necessary.

Unfortunately, if (as RR requires) last random is always initialized to ROUND(maxint/2), then the cycle length will not be random. If the program were to use 3+m*997 rather than 1+m*997, then the cycle might always be 1000 (see Knuth Volume 2, page 15, Theorem A), depending on the starting value of random. Using FLACC, this program completes a cycle for only 96 initial values, and the longest cycle is 39.

I took it upon myself to resolve this problem, and developed the following program:

```
(.INT s:=    0,e:=123;.STRING a="(.INT s:=    0,e:=123;
  .STRING a="";s:=(s*9+7)%*100000;print(2*(a[:9]+whole(
  (s%100=e|0|s),-5)+a[15:33])+2*a[33:]))";s:=(s*9+7)%*100000;
  print(2*(a[:9]+whole((s%100=e|0|s),-5)+a[15:33])+2*a[33:]))
```

The basic idea is that the seed of a five-digit congruential pseudo-random-number generator is passed from one program to the next, and the high order digits are compared with an expected value. When the test succeeds, then the seed is reset, thus ending the cycle. The cycles are of varying length, with an average of 1000. For e=0,123,127,187, the cycles are 1,1564,20,5634 long.

AB47.4.2

Algol 68 as a Living Language

C.M. Thomson
Chion Corporation
Box 4942, Edmonton, Canada

Algol 68 has the potential to become a widely-used, popular language. To date, it has not reached its potential. This paper examines some reasons for this, and suggests some future actions which would result in wider usage. The two things most needed are quality implementations, and a willingness to allow evolutionary change in the language.

1. Some History

Algol 68 (or Algol X, as it was called at the time) got off to a good start. It was widely recognized that there should be a successor to Algol 60, which had proven to be very popular in Europe, and moderately so in North America. Consequently, IFIP Working Group 2.1 began development of a new language with the expectation that it would be well received.

However, as the definition proceeded, it became clear that the Working Group was divided on the issue of how radical a departure Algol X should be from Algol 60. Ultimately, a political decision was made, and many members of WG2.1 resigned over the decision. Those who remained eventually went on to produce Algol 68, while those who departed produced Algol-W, and later Pascal.

As might be expected, because Algol-W was the simpler language, it had working compilers very early. Algol 68, because it required some invention of new implementation techniques, took much longer to be implemented. In fact, the exact language defined in 1968 was never implemented: all compilers were for (often very) different languages.

The experience of the early implementations (particularly Algol68-R) indicated that a revision of the language was necessary. The transput subsystem was in especially bad shape, but other areas (such as call semantics) had not been accepted by implementors. To aid the revision process, WG2.1 set up a series of conferences called the Informal Implementors' Interchange. It was at these conferences that much of the work was done which led to the Revised Report in 1974.

The revised language proved to be rather larger than the original one, but was much better defined. It was around

this time that both of the available full-language implementations (CDC's Algol 68, and Chion's FLACC) were being actively developed. Having a stable target aided these efforts greatly.

During this period of revision, however, much of the original advantage of Algol 68 was lost. Algol-W had been in the field for some years, and Pascal was beginning to attract interest. Also, in its commendable obsession with exact specification of the language, WG2.1 devoted insufficient effort to promotion of the language to the end user. The result (in North America at least) was that most programmers who had heard of Algol 68 at all regarded it as an academic toy. This attitude was reinforced by the large number of never-completed implementations, and by the inscrutibility of the available documentation.

2. The Present Situation

In many respects, the current situation is much better. There are two full-language, commercially-supported implementations, and several large-subset implementations. Algol68-R is very popular in the U.K. There is a growing selection of textbooks on the language. In spite of the six years since its definition, Algol 68 is technically superior to its competitors.

It is the only general-purpose language which is fully specified. Those who can understand the Revised Report can always decide what a particular program is supposed to do. (Well, almost always: transput provides some confusion.) Because the definition leaves so little room for interpretation, compiler validation is much easier than it otherwise would be, and there is a resulting pressure on implementors to conform exactly wherever possible.

Although the years have been kind to most of the language, there are some aspects which have become dated. Newer languages have appeared, with many new ideas. Ada in particular has many facilities which would make welcome additions to Algol 68.

3. Language Evolution and Growth

It is instructive to look at how the most successful languages evolve. Both Fortran and Cobol go through periodic revisions on a five to ten year cycle. This similarity is no accident: any language which remains inflexible over a long period will eventually lose adherents. Periodic updating of its standard gives a language needed vitality. New programmers with modern ideas do not become disenchanted if they can see that a language

is assimilating these ideas.

As computer systems grow more complex, new demands are placed on languages to support facilities which may not have existed at the time the languages were designed. Examples from the past include interactive I/O, database access, and network communication. (It should be noted that Algol 68 addresses none of these.)

The evolution of other languages has been driven by a balance of these two forces: "academic" change caused by new people entering the user community with recent training in modern techniques, and "pragmatic" change caused by incapability of the language to perform functions being demanded of it. Interestingly, many pragmatic changes are implemented long before they are standardized, while academic changes are often proposed and standardized before they are implemented.

The support infrastructure of a language is at least as important as the language itself. To compete effectively with other languages (notably Fortran), Algol 68 must have optimizing and interactive debugging compilers. Currently there are no optimizing compilers, although there is a batch debugging one.

There is a chicken and egg problem here: quality compilers are expensive, and so there must be an assurance of heavy usage to justify their development, yet heavy usage can be assured only if there are good compilers. This problem can be overcome quickly by a massive infusion of money (as with Ada), or else by the gradual appearance of better compilers with its concommitant growth in language use. Algol 68 appears to be following the second route.

4. Some Proposals

In order for Algol 68 to grow and prosper, we believe that several things must happen.

Perhaps most important, Algol 68 must be taught at universities and trade schools. Historically, the languages taught to students have been the languages they have had the most inclination to use in the workforce.

There is a need for more quality implementations, particularly optimizing compilers. A frequent objection to Algol 68 is that its compilers cost more to run than Fortran compilers do. While this is probably not a fair condemnation, it is a real one which must be met before there will be wide acceptance of the language.

There are several facilities which Algol 68 lacks. These include exception handling, separate compilation, data abstraction, database interfacing, and operating system interfacing. There are proposals available for some of these, but none have been implemented.

There is always a fine balance between stability and change in a language. Prior to 1974, Algol 68 was highly changeable. Since then, it has been highly stable. As more time passes, the pressures will increase to allow change again.

Our proposal is that Algol 68 be standardized after the model of Fortran and Cobol. That is, a widely recognized body such as ANSI, ISO, or, indeed, WG2.1 publish periodic standards on the language, and allow for changes between standards. The original report, and the revised report represent the first two documents in such a series. Perhaps the 1983 timeframe would be appropriate for a third standard on Algol 68.

Efforts are already underway to have Algol 68 standardized by ISO. We strongly support this venture.

AB47.4.3 An ALGOL 68 Implementation Companion.

by M.R.Levinson
(CEMI AS USSR)

1. In 1975-1977 the author elaborated an ALGOL 68 implementation draft project as a system consisting of:
- a computer independent translator,
- a concrete generator and
- a concrete operating environment.

The computer independent macroprogram serves as the translator output and the only linkage between the computer independent and concrete parts.

The translator itself is originally recorded in ALGOL 68 and then with the help of another ALGOL 68 compiler is "put through itself" and thus transferred to the macroprogram level.

2. In 1978 the Central Economic and Mathematical Institute of the USSR Academy of Sciences started work on the translator. In 1981 the work should be completed.

At the same time a formalized description of the concrete part of the project [1] as a monograph approximated as much as possible to the text of the official "Revised Report on the Algorithmic Language ALGOL 68" [2] was prepared and deposited with the All-Union Institute of Scientific and Technological Information (VINITI). It follows the text of the official Revised Report, replacing the description of the hypothetical computer and syntax and the semantics of the source language, correspondingly, by the description of the operating computer and syntax and semantics of the macroprogram. The syntax part is written in English, the semantics in Russian.

The description of the operating computer determines in a general way the composition and functioning of the operating environment, whereas the description of the macroprogram determines the tasks of the generator.

3. The operating computer is described twice. First its objects and actions are described; then it is described how these objects and actions "represent" the objects and actions of the original hypothetical computer.

The objects are given in terms of fragments in ALGOL 68 whereas the actions are described verbally.

4. Any operating program is the result of a consecutive translation of the source program into the macroprogram and of the macroprogram into the operating program.

The macroprogram is endowed with the property that, when being still in one to one correspondence with the source program, it completely determines the order of the (linear) generation of the operating program.

It (the macroprogram) represents a sequence of macros described by means of syntax and semantics. The syntax determines which sequences of macros constitute the macro image of the source program, whereas the semantics determines which actions of the computer are set by each macro.

5. The syntax of the macroprogram is a certain extension of the syntax of the source language, consisting of a number of hyper-rules and metaproduction rules from which the production rules are derived.

Each hyper-rule of the macroprogram is derived from one of the hyper-rules of the Revised Report by means of

- its possible deployment, i.e. the application of one or more steps of the consistent substitution;
- the inclusion of metanotions;
- the inclusion of additional predicates;
- the inclusion of macros and
- the omission of symbols.

(Without the omission of symbols the syntax of the macroprogram simultaneously describes the source and macro programs.) To the "old" metaproduction rules new ones are added.

6. The terminal productions are no longer sequences of symbols, but sequences of macros.

Each such macro consists of the number of the macro, preceded by a semicolon and possibly accompanied by one or more parameters separated by commas.

7. The semantics attributes to each macro a certain meaning determined through the "elaboration" of this macro by the generator. This elaboration is accounted for in terms of the executing of the operating program instructions obtained from this macro.

It is not determined in the semantics either which instructions are obtained from each macro, or the order of their processing by the generator (however the pragmatic remarks contain some indications).

[1] M.R.Levinson, ALGOL 68 implementation (draft project). Deposited with the VINITI 14.01.80 N 190-80 Dep., 200 pp., (obtainable through the British Library (lending division) or through other similar National Libraries).

[2] Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica, V 5, f 1-3, 1975.