Testing ALGOL 60 Compilers

B. A. WICHMANN
National Physical Laboratory, Teddington, Middlesex, England

B. JONES
Computing Laboratory, University of Newcastle,
Newcastle-upon-Tyne, England

Software — Practice and Experience, Vol 6, 261-270 (1976)

Abstract

This paper describes some programs used to test ALGOL 60 compilers. The results of the tests on six compilers are given together with some comments on their likely effectiveness in locating bugs in other compilers.[1]

KEY WORDS: ALGOL 60, Testing, Validation, Compiler tests.

1 Introduction

It has been observed by Dijkstra that 'testing shows the presence, not the absence of bugs'[1]. We should like to echo this remark with respect to some tests designed to test an ALGOL 60 compiler. The current state of the art does not permit one to prove the correctness of anything as complex as an ALGOL 60 compiler with an acceptable amount of effort. A LISP compiler for a PDP10 has been shown to be logically correct[2] but the code concerned was only two pages in length. Until the state-of-the-art and compiler construction techniques are suitably improved, those of us concerned with assessing the quality of software must be content with less formal methods. The technique presented here consists of subjecting the compiler to a large number of small test programs all of which contain some unusual feature designed to exercise rarely executed parts of the compiler code. The tests were written at NPL and tested initially on the two KDF9 compilers [3]. The method is analogous to accelerated wear tests used to assess consumer durables. It is interesting to note that consumer testing rarely involves an analysis of the design, which we know presents the only long-term solution for computer software.

With some test programs, it is not possible to state what the 'correct' action should be, because of defects in the specification of the language. In such cases,

[1]Received 29 May 1975, Revised 18 August 1975

any 'reasonable' action by the compiler is accepted, i.e. a correct diagnostic or a plausible action on execution. In practice difficulties in the interpretation of the ALGOL 60 Report[4] is not a problem in evaluating the results.

2 Construction of the tests

When faults are found in a compiler, it is usually because the program being compiled contains an unusual feature which the compiler-writer failed to allow for. A set of test programs must therefore attempt to collect together a sufficiently large set of unusual features to execute a reasonable percentage of the total code in the compiling system. In some cases, the method of constructing such tests is straightforward. For instance, in testing the exponentiate operator, the ALGOL Report[4] defines 18 different cases when all the type information is considered. Eighteen individual test programs can be written to illustrate these cases, although in practice several of those which should not fail during execution can be combined, When these tests were first run on the two KDF9 compilers, both compilers were shown to have a coding error even though the routines had apparently been working for nearly 10 years!

Unfortunately, the construction of the other tests is not so simple, nor is it possible to have the same degree of confidence in the compiler if no errors are detected. To be useful in correcting the compiler, the tests should be small and should not use an exotic feature of ALGOL except the one under test. Unfortunately, many ALGOL compilers restrict the language level significantly in different ways so one must write the tests without:

- Spaces in identifiers,
- Identifiers which could be reserved words,
- Spaces in constants,
- Recursion,
- **goto** out of a function,
- Strings,
- Parameter comments,
- More than one case of characters,
- More than six significant characters to an identifier,
- Use before declaration, etc.,

unless explicitly testing such features.

The tests themselves are available on cards or paper tape in an encoded form [3]. A program must be written to convert this encoded form into the local harware representation used. This program can also generate control cards, etc. to batch several programs together— highly desirable because there are currently 180 tests.

3 The various types of test

A small number of tests can apparently exhaust all the possibilities in limited areas, such as tests of the Boolean operators. Other tests can merely demonstrate the apparent correctness of a language feature such as integer divide, precedence of operators, integer to real and real to integer type conversion, Jensen's device, the three types of for-list element, etc.

Two overall tests are of particular interest because they can be regarded as being exhaustive in a restricted sense. One has been obtained from Mr. Patterson of Glasgow University and consists of a two-page program which should compile and execute. The test is exhaustive in the sense that it exercises every part of the Whetstone compiler [5], i.e. every interpretive instruction is both compiled and executed. This test is hardly likely to be exhaustive in the same sense for any other compiler because of the peculiar nature of the Whetstone interpretive system. As an illustration of this, only one $\uparrow$ operator appears even though several paths are likely to be taken through the code in a non-interpretive system. The second example of an exhaustive test consists of a program which contains all the valid adjacent pairs of ALGOL basic symbols. For the purposes of this test, the upper case and lower case letters are regarded as being synonymous as are the digits. Since several bottom-up compilers execute a different program unit for each delimiter or operator pair, the test should exercise the major parts of such compilers.

Another class of tests is designed to ensure that any size limitations are not excessive. The ALGOL Report gives 12 examples of simple recursion used to define lists—digit length, string length, identifier length, switch list, left-part list, for list, array list, array segment list, declaration list, labels, array dimension and parameters. For each list, a program has been constructed containing a list which is thought to be larger than that required in almost all practical programs. Similar tests are included to check the depth of nesting permitted. The nested constructs that are checked are: procedures, blocks, for-loops, conditional statements, conditional expressions, designational expressions, arithmetic expressions, subscripts and function calls. An interesting additional test of a similar nature contains a 'complex' expression (Boolean, arithmetic or designational) in every position in the syntax where an expression is permitted. This test, like the ALGOL basic symbol pairs, has proved to be very difficult for compilers to handle correctly. All of the tests in this set were constructed using ML/I [6].

Since ALGOL 60 contains the mathematical functions sqrt, sin, cos, arctan, ln, and exp tests could be devised for these. It is hoped that such tests can be added, since this problem, unlike many of the others, is common to FORTRAN. A set of programs do, however, check the numerical accuracy of reading constants which appear both in the program and in data. A further test checks that the printed representation of floating point numbers does not lose accuracy. For FORTRAN tests of mathematical functions see Reference [7].

Tests are also included to examine the handling of the exotic features of ALGOL 60. (Some of the more bizarre features, rarely implemented, such as

dynamic **own** arrays, integer labels and non-specification of parameters are omitted.) The difficulty of handling these exotic features could be blamed upon the definition of the language. However, few other widely used languages can claim a better record than ALGOL 60 in this respect. For a discussion of the definition of the language see Reference[8]. Several problems arise with these tests. They are themselves difficult to test, and indeed if it had not been for the very high reliability of the Whetstone system, their construction would have been too time consuming to be worthwhile. The interpretation of the results from some tests is awkward since many compilers reject them because of language restrictions. It should be noted that about one third of the tests have been designed to fail to compile or execute.

Tests to evaluate the diagnostic features of a system by means of simple programs containing common errors are not included since they are the subject of a separate report [9].

4 How effective are the tests?

It is not easy to say how effective the tests are in practical terms. It seems unlikely that any significant error in the syntax analysis part of a compiler would escape detection. A partial check on the completeness of the tests has been made as follows. A syntax checker for ALGOL 60 has been written in ALGOL 60 [10]. Of the 35 distinct error conditions that the checker can detect, 15 were produced by the first 150 validation tests. Further tests were therefore constructed to give the remaining 20 error conditions. However, these additional 20 tests did not reveal any further defects in the two KDF9 compilers not already noted by the first 150 tests.

The effectiveness of detecting errors in the semantics of an ALGOL 60 implementation is not so satisfactory. This can be seen from a simple count of the number of distinct error messages that a compiler can produce. This is about 400 for each of the KDF9 compilers, more than double that which can be produced by the test programs. Moreover, programs to detect possible errors in the code generation part of a compiler are hard to construct since they depend upon the machine in question and the structure of the compiler. The largest area where errors could reside without detection is probably in register allocation and loop optimization. Errors are likely to be highly context-dependent—for instance, use of integer divide in a subscript or a loop with more than two non-linear expressions in the control variable. Such tests are probably best constructed by inserting automatically program fragments in different contexts.

5 Results of running the tests on four compilers

5.1 The KDF9 Whetstone compiler (Wh)

This compiler is essentially the one described in Reference [5], but with changes made at Oxford University to permit machine-code procedures. For a com-

pilation error, the line number of the error is given together with the current identifier and delimiter. The compilation and run-time error messages are purely numeric, but the description against the numbers in the manual is usually accurate. The version of the system used at NPL for the test does not give the source text position of run-time errors, but a retro-active trace of procedure calls and labels passed does usually give adequate information. Although many of the characteristics of this compiler were known prior to running the tests, the following points were noted:

1. Slow execution due to interpretation. The execution is about twenty times slower than the Kidsgrove compiler, but this disadvantage is partly offset by the higher compilation rate.

2. Many type checks are performed only during execution. This means that errors can remain dormant in code which is not executed. Programmers must therefore ensure that test cases do exercise the program adequately. Five tests translated but failed on execution whereas a compiler should reject the tests on translation.

3. The reading of real constants was not as accurate as possible, and indeed had a bias.

4. The syntax checking was poor in a number of minor cases, none of which was likely to lead to serious errors in execution. The compiler failed to detect the following errors:

   - **if true then** label: **if true then**
     (was accepted as a valid statement)
   - **array** a[1 :1], **procedure** p;
     (was accepted as a declaration with p regarded as a real procedure; this error has now been corrected)
   - x : = 1.;
     (compatibility with FORTRAN ? also now corrected)
   - x/y compiled as (x)/y rather than (x/y)

   The reason for the first two is that the syntax check is made by state variables, using delimiter driven routines. It is difficult to validate this method because of the large number of paths through the code.

5. The other 'errors' detected by the tests relate to decisions taken about the language offered. The most significant one is that the body of a for statement is regarded as a block, even if it does not have the form of one. This permits a static check that no jump is made into a for-loop, but does mean that switch labels cannot straddle a for- loop. In common with many implementations, the Whetstone compiler regards a jump to an undefined switch designator as an error.

5.2 The KDF9 Kidsgrove compiler (Kd)

This eompiler was designed to be used in conjunction with the Whetstone eompiler. The aim was to produee very efficient compiled eode at the cost of a much increased eompilation time [11, 12]. The optimization of for-loops and array accessing has proved to be unreliable, and hence the compiler option for this has been suppressed for these tests. It is thought that almost all the tests would behave identically with the optimization option. Modifications have been made at NPL to do local optimization and given a post-mortem [13].

The reliability of the compiler is significantly worse than the Whetstone system. About 30 programs failed, but 10 failed in such a way as to give the high-level language pro- grammer no useful information (i.e. the equivalent of PSW=<something in hexadecimal>). However, many of these programs could have been found to be erroneous by using the Whetstone eompiler. Error messages produeed on eompilation and execution are less than satisfactory. Errors detected in the first two passes of the compiler do indicate the source text position adequately. Errors concerned with type checking, which are detected in later passes, give no indieation of position, nor any relevant identifier. At runtime, no subscript checking is performed; this can result in erroneous programs overwriting code, etc., from which no sensible diagnostic can be expected. Ordinarily on failure, the last few labels and procedures passed are indicated as well as a post-mortem print-out. No identifiers are given with the failure information, so a certain amount of decoding is necessary to understand the information printed. The shortcomings detected by the tests can be classified as follows:

1. Poor syntax checking. Nine tests were compiled in spite of elementary syntactic errors: but only one would have been passed by the Whetstone system (the '**then** label: **if**'). A typical case is '**array** a[1: 1.1: 1]' (point, not comma) which is accepted as an array declaration but fails on execution giving no meaningful diagnostic.

2. Complexity restrictions. The compiler will not accept more than 95 procedures or a switch list of more than 63 elements. The first restriction, which is fundamental to the working of the compiler, has proved very awkward in practice.

3. Formal procedure parameters. The compiler checks the correspondence between formal and actual parameters at compile-time. To allow this, restrictions are made on formal procedure parameters which are supposed to be checked by the compiler. Unfortunately, the compiler rejects some eorreet programs meeting this restriction and very oceasionally eompiles a program where the parameter eorrespondence is not correct (and hence the program will fail without a meaningful diagnostic on execution). A test program based upon an obscure error discovered by Lucas in the environment handling of the PL/I F compiler aetually caused the Kidsgrove compiler to loop. The handling of formal procedures is also incorrect in that no level pointer is maintained and hence the correct activation record of the procedure is not necessarily found.

4. Designational expressions. Complex designational expressions, especially as a para- meter or in a switch list, often result in a compilation error. In only one case was the execution of invalid code attempted. This was with a bizarre test of Knuth's[14] which calculates the Fibonacci numbers by a side-effect on a switch subscript. This test also revealed that the compiler does not take into account recursion via a switch.

5. A variety of minor semantic errors. Dummy programs fail to compile (need they compile?). No check is made that the dimension of an array parameter is correct. A jump is permitted into a for-loop via a switch which can result in the program going wild if the loop has more than one for-list element.

This catalogue of errors may give the impression that the compiler is useless. This is far from the case. Sensible use of the two compilers yields a reasonable system with the exception of the restriction to 95 procedures and the handling of formal procedure parameters.

5.3 Honeywell 6000 Series compiler (H6)

This compiler is also available on the 600 series machines (originally produced by General Electric). The compiler was first developed in France but has since been re- written in Sweden using the same logic. The tests were run at the University of Waterloo, Ontario, Canada.

The compiler performed very well in the tests—only marginally worse than the Whetstone system even though it is a true compiler. A compilation error gives a number and explanatory text together with the line number of the offending line in the source text. Syntax errors tend to get the compiler out of step with the source text, whereas type errors do not. With the options used for these tests, a source text line number and explanatory text is also produced for execution errors. In a few cases, execution is continued after an error. Out of the total of 130 tests which were run, only 19 failed. They were:

1. Elementary syntax errors; five failed by accepting 1. etc. as a number, and a further three failed by not accepting more than 18 digits in a number. The compiler accepted '**if** $x < y < z$ **then**' and also (for [which is a representation problem. Permitting no digit after a point is presumably a concession to FORTRAN programmers but is clearly contrary to the ALGOL report. The corresponding error in Whetstone is definitely a coding error.

2. Minor semantic errors; the interpretation used for $\uparrow$ is not formally correct since (-3.0)$\uparrow$2.0 did not fail. No check is made that the expressions in array bounds involve only non-locals (a tedious special case in a compiler). The compiler could not handle three programs which were a labelled or unlabelled compound statement.

3. Five programs failed on execution without giving a sensible diagnostic. One involved a nested call of the *sqrt* function, another involved a jump into a for-loop via a switch, one contained all 26 valid formal-actual procedure parameters, another failed on an invalid actual parameter and the last on a labelled program.

Only four tests failed (but with adequate diagnostics) because of a language restriction even though the program was formally valid. The acceptance of such a wide class of programs is not necessarily ideal since many erroneous programs fail on execution rather than compilation. This is particularly true with parameter handling, which is checked solely on execution. The test program which contains all the 114 invalid formal-actual parameter pairs did compile (as it did in the Whetstone system). The system compiled and executed successfully Knuth's switch test mentioned above.

An additional property of the compiler which the tests illustrated was that of producing warnings on the use of odd but formally correct features of ALGOL 60. Warnings were given for a semicolon in a string and for arrays called by value.

5.4 ALGOL F level compiler for the 360 (AF)

This compiler has been available for a number of years on the 360 but has never been widely used. It is known to produce inefficient code and is not actively supported by IBM. It did not come as a surprise that this compiler did not perform these tests well. Although no errors were detected in the syntactic analysis, numerous flaws were found in the handling of semantics in the compiler and in the run-time system. The major difficulty is that such a

large number of the programs fail in the compiler or the run-time system with a program interrupt. The only information given is the hexadecimal address of the failure plus, in the case of a run-time error, the number of the last statement executed. Examples of this were the $x < y < z$ test, a labelled program and several containing invalid formal parameters. The prize must go to the program which contains all 114 invalid formal-actual parameter pairs which was executed without giving any error indication.

The poor showing of the compiler is not offset by the language level offered. Having only six significant characters in an identifier is thought to be unduly restrictive. Four tests failed because fixed compiler tables were exceeded.

5.5 Delft compiler for the 360 (Df)

The poor performance of the F compiler has led to the construction of a new ALGOL compiler at Delft IJniversity. Unfortunately, part of the specification of the compiler is designed to maintain program compatibility with F, so only six characters are significant in an identifier (unless a special option is used). The compiler also shares with the F compiler the restriction that own variables are not permitted. The Delft compiler is a substantial improvement upon the F

compiler. Minor errors include accepting **not not** as valid, and not accepting a labelled compound statement as a program. In almost every respect the Delft compiler is superior to F, but one exception is that run-time errors do not report the number of the last statement executed (at least with the options used for running some of the tests).

Several problems arose in running these tests which have not been entirely resolved. The difficulties appear to stem from local operating system modifications which means that special interfacing code is used. Some tests which originally failed at Cambridge have been re-run successfully at University College, London. Since not all the tests were re-run, the results may be slightly optimistic. An unsatisfactory feature of the compiler is that it can give'warnings' about a program for very different underlying reasons. For instance, a warning is given for a labelled comment (reasonable enough, perhaps, although the program is incorrect), a formal-actual parameter correspondence which *could* result in a run-time error or an invalid formal-actual parameter pair. Not only are such warnings a potential trap for programmers but deciding whether the compiler has 'passed' any test is made unnecessarily difficult.

5.6 Edinburgh compiler (EM)

During the latter half of 1974, P. D. Stephens of the Edinburgh Regional Computing Centre modified the IMP compiler [15] available on the 4/75 to compile ALGOL 60. To assist in this process, the 180 validation tests were made available, since there was no ready source of test programs. The results from these tests are quite different from those for the other compilers, since they had been 'seen' whereas the other systems were run without the compiler-writer's knowledge. Not surprisingly, no syntactic errors were found. The only significant run-time error was that the handling of numerical constants in the data and program was markedly less accurate than would be expected. As the hardware floating point on System 4 is not capable of rounding double length numbers correctly, this defect can only be overcome by the use of decimal arithmetic. The effort involved in hand-coding this is not thought to be worthwhile.

The Edinburgh compiler detected defects in two of the test programs. These programs were supposed to execute but failed owing to the use of unassigned variables. In one case, the unassigned variable was a control variable after the exhaustion of a for-loop. These errors show the difficulty in constructing the tests with the aid of compilers like Whetstone which do not do as much checking as one would like.

The compiler produces very helpful messages on compilation errors involving type-checking and also on run-time errors (assisted by a very good post-mortem). However, errors in the syntax merely give the exact character position of the error without any further explanation. This is awkward for programmers not experienced with ALGOL 60.

Since the tests were 'seen', several programs which might have produced incorrect code have been trapped by the compiler. In total 19 programs have failed to compile owing to an 'implementation restriction'. The language level

offered is as high as can be expected with all parameter checking performed by the compiler. (The parameters to formal procedures are specified by means of a comment as in System 4 ALGOL.) As far as possible the language offered is that specified in Reference [8].

When the compiler becomes more widely used, it will be interesting to note errors which have not been detected by the tests. Two have already arisen—the **goto** to the label of a labelled procedure body and the goto into a compound statement. (Programming without **goto**'s has its advantages.)

6 Numerical summary of results

A seven-way classification of the 'errors' detected by the tests has been made as follows:

1. Elementary error. An error in syntax which does not reflect upon the running program. For instance, accepting a labelled comment or not requiring a digit after a point in a decimal constant.

2. Major syntax errors in translator. None were detected, but an example might be permitting **if** directly after **then**.

3. Minor semantic errors in translator. These are errors which are unlikely to lead to erroneous programs. Examples include permitting **not not**, not permitting a labelled program or an inadequate diagnostic message on translation of an erronous program.

4. Major semantic error in translator. Included under this heading are errors in the translator which could lead to erroneous programs. Examples are the failure to detect invalid formal-actual parameter correspondence when it is clear from the text and no run-time check is performed.

5. Minor semantic error in run-time system. This includes the programs giving inadequate diagnostics as happened with many run-time error conditions using the IBM F compiler.

6. Major semantic errors in run-time system. This includes errors which could easily lead to erroneous programs. The failure to detect any of the 114 invalid formal-actual parameters is counted here.

7. Implementation restriction. When the compiler detects that the valid program violates an implementation restriction and clearly states this, the program is included in this category.

Clearly the compiler writer has several optionschecking at run-time rather than compile-time is easier but less helpful to the programmer. Compiler design clearly entails a careful specification of the language subset being offered. A small subset should be cheaper to implement, such as the omission of **own** variables by the 360 compilers. Poor design may mean that bugs can only be

	Unseen					*Seen*
Compiler	*Wh*	*Kd*	*H6*	*AF*	*Df*	*EM*
Elementary syntax errors in translator	7	9	10	0	1	0
Major syntax errors in translator	0	0	0	0	0	0
Minor semantic errors in translator	0	14	6	6	3	0
Major semantic errors in translator	0	1	1	2	0	0
Minor semantic errors in run-time system	1	8	4	26	1	1
Major semantic errors in run-time system	0	0	0	0	0	0
Implementation restriction	12	7	10	23	17	19

180 Tests run, 4 omitted with IBM F compiler. All entries are program counts.

Table 1: Summary table

'cured' by restricting the subset still further. The categories overlap but an attempt has been made to apply the decisions uniformly. One compiler error could result in the failure of several tests, but no allowance can be made for this. Hence it would be unwise to place much weight on the actual values obtained below. The ranking of the compilers must be roughly in the order Whetstone, Delft, Honeywell, Kidsgrove and then ALGOL F. The Edinburgh compiler cannot be fairly compared as the tests were 'seen'. Experience shows that the level of performance of the Kidsgrove compiler is only acceptable because it is not the sole compiler available for the KDF9.

7 Validation summary

See Table 1.

8 Conclusions

The construction of this set of tests took about 6 man-months, whereas running the test on a new compiler takes about a fortnight. Re-running the tests need only take a day or two depending upon the facilities available for running a large number of short jobs. Hence the construction of such a set of tests is very worthwhile for a standard language when many implementations are available. Although the tests can never show that a compiler is bug free, when the tests are first run, good results imply that faults in the compiler are very unlikely to cause significant problems. Results from the six compilers above show that

existing implementations are very unlikely to pass the tests to the standard one would expect of a key software component.

It is important to note that good results from these tests do not necessarily mean satisfied users. No assessment has been made of the error indications producedmerely that they exist and apparently pinpoint the error. Similarly, post-mortem routines and tracing aids have not been deliberately subject to test. On the other hand, with poor results from these tests, it is hard to believe that there could be any satisfied users. It is interesting to note that some of the best results were obtained by those compilers which generated the most efficient object code.

An important question is the extent to which the tests can be disclosed to compiler-writers and yet provide a useful function. Experience with the COBOL validation tests [16] suggests that revealing the tests is unlikely to result in compilers which are nearly bug-free. A close examination of these tests reveals a similar situation. To disclose any tests which are only a small sample of all the possibilities would merely provide a useful tool to compiler-writers. Examples in this area are invalid formal-actual parameter correspondence and context dependence checking. On the other hand, revealing tests for syntactic errors would probably be more successful in reducing the incidence of such errors in tested compilers. Since syntax errors are not a major factor in the reliability of compilers, the conclusion must be that the tests should not be disclosed.

A major weakness with the COBOL and FORTRAN test systems [7, 16] that they contain no erroneous programs. This means that compilers passing these tests could contain major bugs owing to the lack of adequate checking of the source text. Over half the programs which detected errors in the ALGOL compilers above were not valid ALGOL 60.

9 Acknowledgements

The authors would like to acknowledge the assistance of Professor M. Gentleman, Miss A. Rogers and Mr. P. D. Stephens in running the tests. Many helpful suggestions were received from Messrs. R. Scowen and D. Schofield as well as the referee.

References

[1] E. W. Dijkstra, *Software Engineering Techniques*, NATO, Conference, Rome, 1969, p. 21.

[2] R. L. London, 'The correctness of two compilers for a LISP subset', *Computer Science Department Report CS240*, Stanford University, 1971.

[3] B. A. Wichmann, 'Some validation tests for an ALGOL 60 compiler', *National Physical Laboratory Report NAC 33*, 1973.

[4] Naur, P. (Ed.), 'Revised report on the algorithmic language ALGOL 60', *Computer J.*, 5, No. 4 349 (1963).

[5] B. Randell and L. J. Russell, *ALGOL 60 Implementation*, Academic Press, London, 1964.

[6] P. J. Brown, 'The ML/I macro processor', *Comm. ACM*, 10, 618-623 (1967).

[7] F. E. Holberton and E. G. Parker, 'NBS FORTRAN Test Programs', *Publication No. 399, NBS*, 1974.

[8] R. M. DeMorgan, I. D. Hill and B. A. Wichmann, 'A commentary on the ALGOL 60 Revised Report', *ALGOL Bulletin No. 38*, 1975, pp. 5-37.

[9] R. S. Scowen, 'The diagnostic facilities in ALGOL compilers', *National Physical Laboratory Report NAC52*, 1974.

[10] B. A. Wichmann, 'A syntax checker for ALGOL 60', *National Physical Laboratory Report NAC 53*, 1974.

[11] D. H. R. Huxtable and E. N. Hawkins, 'A multi-pass translation scheme for ALGOL 60', in *Annual Review in Automatic Programming*, Pergamon Press, Oxford, 1963, Vol. 3, pp. 163-205.

[12] D. H. R. Huxtable, 'On writing an optimising translator for ALGOL 60', in *Introduction to System Programming*, Academic Press, New York, 1964.

[13] B. A. Wichmann, *ALGOL 60 Compilation and Assessment*, Academic Press, London, 1973.

[14] D. E. Knuth, 'The remaining trouble-spots in ALGOL 60', *Comm. ACM*, 10, 611-618 (1967).

[15] P. D. Stephens, 'The IMP language and compiler', *Computer J.*, 17, No. 3, 216-223 (1974).

[16] AFSC, *User's Manual, COBOL Compiler Validation System*, Hanscom Field, Mass., 1970.

A Document details

1. Converted in October 2001.