Some validation tests for an Algol 60 compiler

B A Wichmann,
National Physical Laboratory,
Teddington, Middlesex, TW11 0LW, UK

March 1, 1999

Abstract

This report contains details of over 100 Algol 60 test programs available from NPL which can be used to validate a compiler. Interpreting the results of the tests requires a good knowledge of Algol 60.

Contents

1 Introduction

ALGOL 60 is comparatively well defined. One advantage of this is that it is possible to devise test programs independently of the compiler. Such tests cannot, of course, be expected to find all the errors in a compiler. The only possible way to perform a complete check is to analyse the logic of the compiler which represents an order of magnitude of more effort than writing the compiler in the first place, and so cannot be reasonably attempted in the existing state of the art. Hopefully future work on proving the correctness of algorithms could be extended to include an ALGOL 60 compiler. For a summary of existing research into this area see London [16] and Lucas [17].

Accepting that tests cannot be complete does not mean that a collection of tests cannot be useful. An attempt has been made here to collect tests which cover the major difficulties in the translation of ALGOL 60. At best, these tests could reveal all the known errors in an implementation, and at worst, such tests demonstrate that large parts of the compiler apparently work.

All the tests have been checked by executing them with the two compilers that are available on KDF9. Comments are included here on how the compilers handled the tests. It is to be hoped that the tests will be run on several systems to enable detailed comparisons to be made.

Many of the tests are of a controversial nature. In some cases, an implementor could reasonably state that such programs are not valid ALGOL 60. Even so, the compiler should deal intelligently with such programs by generating good diagnostic information. In many other cases, the test programs can be expected to violate a stated restriction of a compiler, but again, an intelligent message is to be expected.

The test system incorporates a number of special programs written by R S Scowen, I D Hill [7] and M Woodger. Hopefully it will be possible to add further tests which are known to have failed in at least one system.

The amount of effort required to run these tests depends upon the experience of the programmer, the ease of access to the computer and the facilities provided. Of crucial importance is the ease with which ten or so tests can be run one after another. As over half the tests are liable to fail to translate, it is necessary to have a large number of individual programs. As a rough estimate one to two man-months should be sufficient to run all the tests given reasonable machine access and the ability to batch several tests together. Testing a new release of a compiler should be much simpler, involving only a few days work.

2 The test system

It is very important that no punching errors should be introduced in the preparation of the tests. For this reason, the tests must be formed automatically from cards or paper tape supplied in a standard format. The formation process can be used to generate control cards, library requests and other data necessary for execution of the programs. Since there are over 100 tests, it is obviously important that batches of tests should be handled automatically by the system. To assist the automatic handling, the programs use a very simple input-output system, and do not use any input data except where strictly necessary.

The programs themselves are not reproduced here, since some of them are very long, and in any case, it would encourage hand conversion. The conversion itself is merely a transcription of an encoding of the ALGOL basic symbols into the local hardware representation, To ensure that this is adequate, various programming conventions are adopted which are known to be necessary because of the nature of certain implementations.

3 Programming conventions

The following restrictions are observed, except in these tests whose object is to investigate the nature of the restriction. The system having the particular limitation is noted in brackets.

1. No spaces in identifiers (Univac 1108)
2. No identifiers that could be reserved words (Atlas (cards) Univac 1108, B5500)
3. No spaces in constants (Univac 1108)

4. No recursion (IFIP)
5. No type-changing on a call-by-name assignment (KDF9)
6. No label parameter to a function or **goto** out of a function (B5500)
7. Regarding a **for** loop as a block should not invalidate the program (KDF9, Whetstone)
8. No strings (B5500)
9. No parameter comments (B5500)
10. No upper case alphabetic characters (B5500, CDC6000, Univac 1103, etc)
11. Only six significant characters in an identifier (IFIP)
12. Not more than 32 characters in an identifier
13. No use before declaration (Elliott, B5500)
14. No integer labels (Atlas, 1900, KDF9, B5500, etc)
15. No side effects on function calls (ALCOR)

These conventions are the main reason why it has not been possible to use many existing test programs.

A small number of procedures are used for input-output and finding the amount of processor time used by the program. These procedures are:

```
real procedure cputime;
        comment procedure gives the processor time taken by the
                program in arbitrary (but known) units. Microsecouds
                are preferred;
                <body>;

procedure outreal (channel, x);
        value channel, x;
        integer channel;
        real x;
        comment print x to full accuracy, see IFIP
                input-output procedures, CACM Vol 7 Mo 10,
                pp 628-630. The ACM convention is adopted
                on channel numbers. In fact channel is
                always 1;
                <body>;

procedure inreal (channel, x);
        value channel;
        integer channel;
        real x;
        comment IFIP procedure as above. Assigns to x the
                next number on the input tape. Channel is
                always 2;
                <body>;
```

Every test program requiring some of these procedures or the ALGOL standard functions will have the following in place of their declaration

library <n>;

where n is a number, which indicates the required procedures by its binary representation

- $n = 2^0$ outreal
- $n = 2^1$ cputime
- $n = 2^2$ standard functions
- $n = 2^3$ inreal

so that **library** 5; means that the standard functions and outreal are required.

The user can adopt two methods of handling these procedures. The **library** symbol can be replaced by the text of the procedures required. Alternatively, a macrogenerator or a good editor (if available) can be used to alter the calls of outreal, inreal and cputime to those of routines already available within the system.

The programs use these procedures only when necessary. The output produced is very small, as the majority of the tests are self-checking. Only two programs use inreal, the purpose of which is to check that input of real numbers does not have a consistent bias.

4 The automatic preparation system

Each ALGOL test program, and the data where relevant, is considered to be a sequence of ALGOL basic symbols. In order to allow convenient output without the use of an automatic formatting program, the ALGOL basic symbols are extended to include: space, tab and newline.

The tabulate character is used to indent the **begin-end** structure of the program and corresponds to eight spaces. Fewer spaces could be used with advantage if the representation of the basic symbols in terms of characters is more than that of the corresponding words. If a reserved word system is being used, then the conversion must ensure that **real array**, for instance, is separated by a space.

In addition to the above extensions, more ALGOL basic symbols are introduced to ease the conversion problem. These are

library This is used for textual insertion of the required input-output routines, as explained in the last section

endtext Denotes the end of the suite of programs

endprogram Separates one program and its data from the next

data If a program requires data, this is included after the program text, and is separated from that text by the **data** symbol

The idea behind these symbols is that, on detecting them, the conversion program can call special procedures to generate control cards automatically.

The sets of programs can be provided in a numerically encoded form of the ALGOL basic symbols on cards or paper tape. Each basic symbol is assigned a code between 0 and 259. This code is then represented by <letter><digit>, where (code $\div$ 10)+i is the number of the letter in the alphabet and the numerical value of the digit is code-code$\div$10 $\times$ 10. The conventional representation of letters and digits is used for cards using columns 2-71 inclusive. This gives 35 ALGOL basic

symbols per card, and in fact columns 1 and 72 are always blank and columns 73-79 contain a serial number. On paper tape, the ISO UK code is used with an even parity bit added. Thirty-five ALGOL basic symbols are again punched on each line in columns 1 to 70. Each line is terminated by CR, 12. No serial numbering is provided. In practice, fewer cards are used titan for an ordinary representation (about half), but the paper tape form is slightly longer. The ALGOL basic symbol code is listed on section 11.

A sum check of the individual code values is produced on a separate line after the endtext symbol. The partly completed line is blank on cards (apart from the serial number) and is terminated by CR, LF on paper tape. The sum check itself is six decimal digits and is module 1000000.

A KDF9 ALGOL program to perform the conversion to any local dialect is given below. This program will require extensive modification to output control cards and to set up the tables to drive the ALGOL basic symbol conversion.

5 The conversion program

(omitted)

6 Comments on the test programs

It cannot be expected that a compiler will pass every test. Indeed, in some circumstances, it is hard to say what 'pass' means. A critical part of the tests is therefore an assessment of the action taken by the compiler and run-time system. Several tests may violate restrictions in the language imposed by the compiler. Such tests should nevertheless be run in order to check that the compiler and/or run-time system correctly detects and fails such programs.

In order to give a single numerical indication of the success of a compiler a scoring system could be used. In this case, because of the difficulties mentioned above, no such system is used. However, in order to indicate actions that might reasonably be taken, the results of running the tests on the Whetstone and Kidsgrove compilers are given. This is accompanied by a short explanation, the expected output and comments on what might reasonably be expected.

On a translation failure, the Whetstone compiler produces a line number and additional context information on the position of error. In certain cases, an additional line number is given, for instance on an inconsistent use of an identifier, the two inconsistent uses are indicated. Execution errors use a retroactive trace (Randall [20]), but positional information is not given with the version of the compiler used for these tests. In practice, the retroactive trace is usually adequate. The Kidsgrove compiler gives positional indication only on simple errors found during the first pass of the translator. Execution errors do not give positional information either, but an option exists to give a retroactive trace for procedures and labels. An ALCOR [2] style post mortem dump is produced but compiler tables are needed to interpret it. A combination of both systems provides a reasonable environment for testing and production running, the main defect being that programs containing errors can translate correctly in Whetstone, but fail giving any positional information with the Kidsgrove compiler. Cases where no positional information was given or an unusual action was taken are noted against each test.

Almost all the test programs have been formatted with SOAP [21]. In some cases, this has lengthened the text substantially compared with a conventional layout.

7 Possible extensions

Although the 130 tests included in this report exercise most of the important features of ALGOL 60, there are some serious omissions which ought to be rectified. The major areas not tested include:

1. Own variables or arrays in recursive routines
2. Tests for the numerical behaviour of the standard functions *sqrt*, *ln*, *arctan*, *exp*, *sin* and *cos*. This is currently being investigated at NPL.
3. Tests on the completeness of the syntax checking. One should ensure that every valid delimiter pair occurs in at least one test program
4. Integer labels and unspecified parameters. Since these are not permitted in the majority of implementations, tests are not so important. Using these facilities pathological programs can be constructed whose meaning may be open to several interpretations.

In addition to these specific areas, programs that are known to have failed in particular systems are of interest, especially when the form of the error is not illustrated by any of the other tests.

8 Acknowledgements

Many of the test programs have been constructed from or are copies of published algorithms. Several were also obtained from Mr M Woodger of the Computer Science Division in NPL. I should like to thank Mr Woodger, Mr R Scowen and Mr I D Hill for many suggestions for possible tests.

9 Notes on each test

Test 1

The program checks that integer or real constants which overflow the machine capability are correctly flagged by the compiler. The overflow of both integer and real constants should be detected in one compilation attempt.

The Whetstone compiled correctly, detecting the overflow on the integer, but missing the real number. The Kidsgrove compiler failed to give any error message owing to an attempt to execute an illegal instruction.

Test 2

Some difficulties arise in ALGOL 60 because the number of significant characters in an identifier is not defined. If two identifiers differ only after a large number of characters, then provided the declarations occur in the same block, the error will be detected. However, if the declarations occur in nested blocks, the program could be incorrectly translated without comment. Ideally, a compiler should check that nested declarations do not contain a 'similar' identifier of excessive length.

The program illustrates this difficulty with two identifiers of 33 characters long similar in all but the last character. The program should print 1.0, but almost all systems will print 2.0 because the two identifiers will not be distinguished. ALGOL 60 permits the implementer to use any algorithm for distinguishing identifiers, but every system known to the author takes the first few characters. Alternatively one could use the length in characters, the last few characters in addition. A

hash function of all the characters could be used, but this has the very substantial disadvantage that the algorithm could not be repeated by hand.

Both KDF9 compilers took the expected action. Whetstone compiled and executed the program giving the 'wrong' value 2 as only eight characters are significant in an identifier. The Kidsgrove compiler produced the correct answer 1, since up to 155 characters are significant.

Test 3

Knuth [13] has pointed out that every label must be in a block, but that a complete program can be a labelled statement and so is not formally in a block. This program, which is syntactically correct, could be rejected on semantic grounds by a compiler. The program is labelled and is not a block, which could cause difficulties with a compiler.

The program compiles and executes successfully in the Whetstone system. The Kidsgrove system fails undeclared identifier on the reference to the label. The original Kidsgrove compiler did not permit labelled programs, but initial labels have been used at NPL as a method of passing parameters to SOAP [21], the formatting program. To overcome this error in the Kidsgrove compiler, it was modified to skip to the first **begin**, thus ignoring any initial labels.

Test 4

The standard function identifiers can be redeclared for different uses. This program checks this using such identifiers for initial labels and within the program.

Compiles and executes successfully in both the Whetstone and Kidsgrove systems. In fact, the test was not formally handled correctly by the original Kidsgrove compiler as can be seen from the preceding test.

Test 5

An ALGOL program, although it can be labelled, must have a **begin** because if it is not a block, it must be a compound statement. The program in this test is therefore erroneous, although it is virtually identical to test 3.

Fails in both Whetstone and Kidsgrove compilers.

Test 6

A convenient facility that does not exist in ALGOL 60, is to abbreviate **if** x < y **and** y < z **then** to **if** x < y < z **then**. This program checks that such constructions are rejected by the compiler. A language extension may permit this, but the author does not know of such a compiler. CPL allows this construction with the obvious interpretation. Both APL and PL/I allow this syntactic form but the semantics are different — a real trap.

The Whetstone compiler fails this program with the very explicit message 'relation on each side of simple arithmetic expression'. The Kidsgrove compiler, although failing the program, gives no indication of the position and the message is less helpful namely 'failure in selection matrix, two operators are being compared which are invalid'.

Test 7

The implementation of call-by-name parameters has many far-reaching consequences. One difficulty is that the validity of an assignment to a name parameter depends

upon the actual parameter. It is therefore difficult for a compiler to check this, and so many systems make a dynamic check at run-time. It is not at all clear whether a program containing an assignment to a name parameter, with an actual parameter inconsistent with this, can be legitimately rejected at compile-time. It may well be that the assignment is not executed when the actual parameter is not a variable. The author's view is that the check should be dynamic, but if an actual parameter is inconsistent with this, the compiler should print a warning message (and compile the program).

Both KDF9 compilers translate the program and fail it on execution. The Whetstone compiler error message 'integer assigned to real actual parameter called by name' is not quite correct. The Kidsgrove compiler error message is less precise 'invalid assignment to name parameter'.

Test 8

This program contains a trivial syntax error, a declaration follows a statement. It should fail to compile.

Fails to compile in both systems with the messages 'declaration follows statement' (Whetstone) and 'declarator not in block head or specifier not in procedure head' (Kidsgrove).

Test 9

The removal of comments from an ALGOL program is not as straightforward as one might expect. The reason for this is that comments are of three types, following **comment**, following **end** and a parameter comment. All of these forms have different constraints, and the context in which they appear requires a nearly complete syntax check. The next six programs illustrate these problems. This test ought to compile although it contains some very odd comments. There is a grave danger that a statement could get lost in an **end** comment, so a compiler should give a warning if a delimiter appears in an end comment (as in this case). Note that strings and comments are unconstrained in their use of bracketing pairs (i.e. 0, [ ] , **begin end**, **for do**). The program contains one nested string which could be removed if they were not allowed in an implementation.

Compiles and executes successfully with the Whetstone system, but prints a warning that st, a and b are declared but not used, and that an **end** comment contains a delimiter. Compiles and executes without comment with the Kidsgrove compiler.

Test 10

This is also a valid program containing odd comments. If one 'comment' were removed, then it is possible that the program could fail by having too few parameters to a procedure call. Compiles and executes successfully with both KDF9 systems (producing no output), but the Whetstone compiler gives a warning of identifiers declared but not used.

Test 11

This program is invalid. A parameter-like comment appears where a comma is expected but not the comma between procedure parameters. Thus if a compiler replaced any parameter-like comment by a comma before doing the rest of the syntax analysis, this program could compile.

Fails to translate in Whetstone, with the message 'misused) other than in expression'. Fails to translate in Kidsgrove with the message 'various syntax errors in block or procedure head'. Both compilers pinpoint the error accurately.

Test 12

A parameter comment must have a colon after the final letter. This program is invalid because this colon is omitted both in a procedure declaration and a procedure call.

The Whetstone compiler detects both errors with the message 'illegal parameter comment', but does not adequately recover to find any further errors. The Kidsgrove compiler fails to compile the program, nor does it produce any diagnostic.

Test 13

The syntax of comments following the symbol **comment** prohibits labelled comments. This program tests for this point, and should therefore fail to compile.

Fail 'comment does not follow begin or;' in Whetstone. The Kidsgrove compiler executed an invalid instruction without producing any error message. It is known that the compiler accepts labelled comments.

Test 14

Parameter comments can only contain letters and digits. A parameter comment in this program contains **comment**, and is therefore invalid. It is not clear if this program should be rejected when a reserved word representation is used, since the basic symbol **comment** will only contain letters.

Fails to translate in Whetstone with three error messages 'illegal parameter comment' twice and then 'end of file inside program'. The Kidsgrove compiler executed an invalid instruction without printing an error message.

Test 15

Basically similar to test 10, this contains strings and paranieter comments in a form that could confuse a poor syntax analyser. This program is invalid, unlike test 10.

Fails to translate in Whetstone, with two error messages 'illegal parameter delimiter after string', and 'closing string quote misplaced'. Fails to translate in Kidsgrove with the message 'bracket mismatch'.

Test 16

A valid program, checking that nested strings are acceptable to the compiler.

Compiles and executes successfully with both the Whetstone and Kidsgrove systems (no output is produced).

Test 17

Checks that identifiers can be used that are spelt the same way as basic symbols. This test will fail in any reserved word representation system.

Compiles and executes successfully with both the Whetstone and Kidsgrove systems (no output is produced).

Test 18

This program checks that redeclaration of 'similar' identifiers within the same block fails to translate. The identifiers, while formally distinct, are the same in the first 47 characters. The position of the error detected by the compiler should indicate the number of significant characters in an identifier. Also checks that spaces are allowed in identifiers and constants.

Fails to translate 'redeclaration of an identifier' because only the first eight characters are significant in Whetstone. Compiles and executes successfully (no output) in Kidsgrove because 155 characters are significant in an identifier.

Test 19

This program contains a procedure call in which the dimension of the formal and actual array parameters are different. This is invalid ALGOL 60, but since array specifications do not give the dimension, it is difficult for the compiler to check.

The Whetstone system compiles the program, but it fails in execution with the message 'incorrect number of subscripts to formal array'. The Kidsgrove ALGOL compiler also compiles it, but fails on execution with the message 'overflow set on exit from procedure sum 2'. This has been produced because the core-store is initially set to an undefined floating point value whiich is likely to cause overflow. The invalid access to the array elements evidently fetches one of these words causing overflow. The error message is therefore very misleading, as the system makes no explicit check for this form of error.

Test 20

One difficulty with ALGOL 60 is that the dimension of a formal array parameter cannot be specified. Where necessary, the compiler must deduce this from the source text. However, if a formal array parameter is only handed on to a further procedure, the dimension may not be determinable without quite complex analysis. It is not clear from the Report if a program which uses an array parameter with different dimensions can be regarded as correct.

These problems are illustrated by this and the following test. Both programs attempt by different means to provide a procedure to add up the element of two arrays of one and two dimensions. This test is the least dubious, as each procedure uses the formal array consistently. A compiler could, with good reason, refuse to compile the program on the grounds that the actual arrays submitted to the procedure 'sum' vary in dimension. Certainly a compiler should print a warning message and insert additional run-time cheching code.

Compiles and executes on both KDF9 systems, printing the two numbers 14 and 60.

Test 21

Similar program to test 20, except that within one procedure a formal array parameter is used with a variable number of subscripts. The use of each array is correct, but the compiler cannot check this at compile time. Since a run-time check would have to be made at each array access, to accept (and validate) programs like this, the author's view is that this program should not compile.

Fails to compile with the Whetstone system giving the message 'wrong number of subscripts or parameters'. Compiles and executes 'successfully' with the Kidsgrove compiler printing the two numbers 14 and 60; however this is only achieved at the cost of performing no parameter checking.

Test 22

The implementation of label parameters in ALGOL 60 is quite complex because of dynamic storage allocation. The corresponding **goto** must not only jump to the correct machine address, but it must also restore the correct activation of the block enclosing the label.

These difficulties are illustrated with this example involving a recursive procedure (to get several activations of the label 'exit'). The program prints the numbers 1 to 10, and ran successfully with both the KDF9 compilers.

Test 23

A further difficulty with label parameters is caused by the fact that they can appear in the value list. The corresponding actual parameter to a label by name must not be evaluated until the **goto** is reached. Some of these points are illustrated in this example, which is constructed so as to loop or fail if the labels are not evaluated at the right point.

Whetstone compiler translated and executed the program correctly, printing the two numbers 1 and 2. The Kidsgrove compiler failed to translate the program with the message 'monadic (operand 1) not available, machine or real array identifier used as actual parameter where formal was real by value'. This message is completely misleading and, as no positional information is given, it is virtually useless. The error is apparently caused by having a switch element as the actual parameter to a label by value.

Test 24

Array parameters can also be called by value, in which case, a completely new copy of the array must be made. This mechanism causes obvious implementation difficulties and is sometimes excluded by a compiler. In this test, a further complication arises because the actual and formal arrays are of different types. Hence a type conversion is required. If value arrays are allowed, but the type conversion is not, then the integer from the actual array declaration can be omitted.

Compiles and executes correctly with both KDF9 compilers, printing the value 45. Value arrays are not important, especially with type conversion, so failing this test is not serious.

Test 25

The syntax of real numbers is surprisingly complex, and all fifteen different forms are used in this program. As equality of real numbers cannot be guaranteed, the numbers are merely printed. The program prints 43 numbers, the output of which is obvious from the program text.

The program also checks that numbers with excessive zeros immediately after a decimal point are not converted with loss of accuracy. Similarly π is expressed with excessive precision, but should be printed to the full accuracy of the machine.

Compiles and executes correctly in both the Whetstone and Kidsgrove systems. The round-off characteristics of the two implementations is different (see test 55), and so the actual output is marginally different, but not excessively so.

Test 26

Although some ALGOL experts would disagree, there appears to be three levels of nomenclature in a procedure. These are the procedure identifier (if a function) and the parameters, labels to the procedure body, and locals to the procedure body.

This program contains three clashing identifiers at each level. The purist may expect such a program to compile, but the author's view is that it should not, or at least a warning should be produced.

The program fails to compile in Whetstone with the message 'redeclaration of an identifier'. Compiles and executes (no output) without comment with the Kidsgrove compiler.

Test 27

This test was constructed by P Lucas to demonstrate an error in environment control in an implementation of PL/I. It was communicated to the author by Dr E Satterthwaite. Formal procedures, like labels, involve an activation level apart from a machine-code address. The purpose of this test is to show that the activation level is handled correctly.

The program compiles and executes correctly with the Whetstone system, printing the numbers 2, 1, 2, 2. The program fails to translate with the Kidsgrove compiler, with no intelligible error message. Investigation showed that this was caused by the compiler attempting (unsuccessfully) to discover the parameter specification of formal procedure parameters. The chain of formal procedure calls involved a loop, so the compiler itself looped attempting to find an actual procedure giving the specification details. Successful compilation and execution of programs that are as complex as this is not of great practical significance, but failure may indicate more serious defects in the compiler. The Kidsgrove compiler's performance cannot be regarded as satisfactory.

Test 28

This program is another test of the activation levels of formal procedures. The original version was from Dr Satterthwaite in ALGOL W. The coding In ALGOL 60 could use labels (test 30) or procedures as in this test.

The program compiled and executed successfully with the Whetstone system, printing the six numbers 4, 5, 4, 5, 6, 7. The Kidsgrove compiler compiled the program but executed it incorrectly printing the 12 numbers 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 6, 7. This was traced to the fact that the compiler does not maintain an activation level with formal procedures — only the machine-code address. This is a logical error in the compiler, but it is not likely to have a significantly detrimental effect in most applications programs.

Test 29

This program is a modified version of test 27. Some compilers, with good reason, demand that the parameter specification of each actual procedure given as a formal parameter should be identical. Test 27 violates this requirement because it has a formal parameter g in p which may require two parameters or none, depending on the actual procedure parameter. The program should therefore compile and execute as for test 27, as is the case in the Whetstone system. The Kidsgrove compiler failed in the same manner as test 27, that is, it looped.

Test 30

This program is a logically similar test to number 28, using label parameters instead of procedure parameters.

The program compiled and executed successfully with both KDF9 systems. Hence the Kidsgrove compiler handles the environment of labels correctly, but not that of formal procedures.

Test 31

This is an invalid program and should fail to compile. An **if** may not follow a **then**, even in those cases where the context, such as a conditional expression, does not necessarily make it unambiguous.

Whetstone fails '**if** misused' twice. Kidsgrove compiler fails 'conditional expression with no else part', with no indication of the source text position.

Test 32

There are substantial difficulties in implementing name parameters in ALGOL, especially when assignments are made to the parameter. This is because the validity of the assignment can only really be checked at run-time. A further difficulty arises if the formal and actual parameters are not of the same type but are both integer or real. In this case, a dynamic check must be made on assignment to see if a type conversion is necessary. Neither of the KDF9 compilers allows for this type changing on name assignment, but both compile the program. Some compilers reject this program at compile time because with the first call of *ass* the second actual parameter is -1 which cannot be assigned. However, although an assignment does appear in the body of the procedure, it is not executed. The program should output 8 and 16.0.

Results: Whetstone compiler fails 'real number assigned to an integer actual parameter call by name', on the last call of *ass*. The Kidsgrove compiler fails at the same point with the message 'invalid actual parameter on name assignment'.

In the author's view to fail at either compile time or run-time as above, or to fail to execute correctly are all reasonable. The important point is that if it does not execute correctly a reasonable diagnostic should be produced.

Tests 33-51

The 19 tests from test 33 to 51 have been generated automatically using the ML/I system [3]. Their purpose is to test that programs which are complicated in some sense do not overflow any compiler tables. They have been chosen by taking every syntactic list which appears in the ALGOL report (defined by simple recursion). The test consists of a simple program which contains this list which is about three times longer than might ordinarily be expected in programs of a few pages in length. Thus, except for the one long list, the programs are very simple.

Test 33

The nesting of procedures can cause implementation difficulties. This program contains six procedures nested within each other. There is no output.

Both Whetstone and Kidsgrove compilers apparently compile and execute this successfully.

Test 34

Contains 15 nested blocks. There is no output. It is compiled and executed correctly in both Whetstone and Kidsgrove.

Test 35

Program declares 300 real variables in one list. They are all initialised to 1.0, and their sum is printed out. Compiles and executes correctly in Whetstone and Kidsgrove. Prints the number 300.

Test 36

Program declares 60 1-dimensional arrays of ten elements each in one array segment. The first element of each array is set to 1.0 and their sum is printed.

Compiles and executes successfully in Whetstone. Fails in Kidsgrove at compile time with the error message 'too many dimensions in array or too many arrays in segment'. The manual states that not more than 31 arrays can be declared in a segment. This is not a severe restriction, as they can easily be broken up.

Test 37

The program contains 12 nested for loops, and should print the number 8190. Compiles in Whetstone ALGOL, but fails due to lack of time after 3 minutes execution. Compiles and executes successfully in Kidsgrove (26 seconds to compile, 3 seconds to execute).

Test 38

The program contains 24 nested conditional statements, and should print the number 24. Compiles and executes successfully withi both the Kidsgrove and Whetstone systems.

Test 39

This contains a conditional expression nested nine deep, and should print the number 2. Compiles and executes successfully in both the Whetstone and Kidsgrove systems.

Test 40

Tests that switches if length 300 are allowed. The program also executes a to each label of the switch, and should print the number 301.

The Whetstone compiler translates the program in 25 seconds, and takes 14 seconds to execute the program successfully.

The Kidsgrove compiler fails to translate the program because of a stated restriction that switches are limited to 64 elements. In the authors opinion this is an annoying but not a very severe restriction (longer switches can be broken into two or more).

Test 41

Contains a function with 60 integer value parameters. Prints the number 60. Compiles and executes successfully with Whetstone and Kidsgrove compilers.

Test 42

This test is to check the avaiability of arrays of large dimension. An array of 12 dimensions is used containing 2^{12} elements. The first element is initialised and printed (value is 1.0).

Compiles and executes successfully with the Whetstone and Kidsgrove compilers.

Test 43

Contains an expression nested 15 deep. The nesting is straightforward but of such a form that special action would be necessary to produce good code on a left to right parse. With the Kidsgrove compiler, this checks that possible overflow of the KDF9 nesting store is handled correctly.

Prints the number 15. Compiles and executes correctly with both the Whetstone and Kidegrove systems.

Test 44

Contains a nested exprcssion 15 deep which is rather more complex than the last one. Prints the real number $4 - 3/2^{15} = 3.99981689.....$

Translates and executes successfully in both Whetstone and Kidsgrove systems.

Test 45

Contains a nested call (9 deep) of the standard function *sqrt*. Prints the number 1.03811889....

Translates and executes successfully in both Whetstone and Kidsgrove systems.

Test 46

Contains a nested designational expression of depth 6. Produces no output. Translates and executes successfully in both Whetstone and Kidsgrove systems.

Test 47

Program consists of six labelled dummy statements only. Translates and executes successfully with the Whetstone system. Although the program fails to translate with the Kidsgrove compiler, no error message is produced. This is thought to be because the program contains no executable code (as for test 13).

Test 48

Program contains a series of compound statements nested 24 deep. Prints the number 24.

Translates and executes successfully with both the Whetstone and Kidsgrove systems.

Test 49

Contains an assignment statement with 15 elements on the left hand side. Prints the number 15.

Translates and executes successfully with both the Whetstone and Kidsgrove systems.

Test 50

Contains a for loop with 60 for test elements. Prints the number 1530. Translates and executes successfully with both the Whetstone and Kidsgrove systems.

Test 51

Declares in one list 60 arrays of ten elements with separate bounds. Initializes the first element of each array and prints their sum (60). Translates and executes successfully with both Whetstone and Kidsgrove systems.

Test 52

An expression can appear in a number of places in the syntax of ALGOL. The expression can be arithmetic, boolean or designational. A complicated expression of all these types has been constructed and inserted in examples of all the valid places. For this purpose ML/I was used (indeed, it would be tedious to do it without such a tool). Prints the number 6.8, but could be subject to substantial rounding errors[1].

Compiles and executes successfully with Whetstone in two seconds. Fails to compile in Kidsgrove with the error message 'generator nest with more than one result (conditional expression)'. Since no positional indication is given, the compiler cannot be regarded as satisfactory. Closer inspection reveals that the error is probably due to having a complicated expression in the **step** part of a **for** list element.

Test 53

Tests the integer divide operator against the definition given in the ALGOL report using sign and entier. No output is produced if agreement is found with 20 tests. However, the ALGOL definition relies upon the fact that, for example, entier (2/1) is 2. Rounding errors could result in output being produced although the integer divide is calculated correctly. It would have been much better if integer divide had been defined without the use of real operations. The output allows a hand check to be made.

Compiles and executes successfully with both Whetstone and Kidsgrove systems.

Test 54

Tests sign, entier and abs, all of which should not involve any rounding errors. Real equality is used with abs, which could be criticised on the grounds that this is not well defined. However it is hard to believe in this case that equality should not be found. As with test 53, entier (100.0) may be correctly implemented and yet, as above not equal to 100.

Compiles and executes successfully with both Whetstone and Kidsgrove systems.

Test 55

This is quite a long program, whose object is to test the consistency of different methods for obtaining a rational constant. The three methods are direct calculation using one real division, using an explicit constant in the program, and reading the number off the data tape. The three real quantities obtained can be compared for equality or the form of inequality. In the author's view, the constant in the program and that on the data tape should produce identical results since the same routine should be used. The calculated value could be different, since it will depend upon the floating point division hardware. In all cases there should be no consistent bias, as this would indicate a lack of care in dealing with round-off conditions.

[1] As with the other examples, the program was then formatted with SOAP [21]. This revealed an error in the line justification part of SOAP.

The program works by using 399 rational numbers of the form d.dd (d.dd = 0.01 to 1.00), lOd.dd (100.01 to 101.00), lOOOd.dd (10000.01 to 10001.00) and 0.<20 digits> (value = $1/(i \times i - 1), i = 2$ *to* 100). The last case involves rational fractions specified to 20 decimal digits, which should be adequate for up to 64-bit real arithmetic (the fractions are correctly rounded as decimals). The test using the last 99 numbers is substantially more severe since it involves reading more digits than the precision of the machine allows.

The program counts each form of inequality, and prints this out after each block of tests, producing 36 numbers in all.

The results are best summarised by a table. Whetstone ALGOL completed the program in 6 seconds and executed in 28 seconds. The final matrix of inequalities was

operator	(read,calculated)	(calculated,constant)	(constant,read)
<	182	0	1
=	216	372	233
>	1	27	165

There is therefore a consistant bias except for one number, with constan $\leq$ calculated $\leq$ read. Strict ineqality occurs 41% and 7% of the time respectively. On this basis, the routine in the Whetstone compiler for reading constants must be regarded as suspect. The Kidsgrove system uses the same routine for reading data and so gives the same 7% deviation from the calculated values as above. The full matrix is

operator	(read,calculated)	(calculated,constant)	(constant,read)
<	30	0	0
=	369	372	396
>	0	27	3

Hence there is a one per cent deviation between the calculated and constant values. The program took 1 minute 6 seconds to compile and 8 seconds to execute.

In order to see if another input-output package would produce better answers, the program was converted and run in Babel [22]. The read routine is common to the compiler and run-time system and is known to work double length. With 360 numbers (fewer due to a compiler limitation) only on iequality was found.This shows that good results are attainable with the KDF9 harduare.

Further tests should be made to measure the magnitude of the deviations, but interpretation of the results is likely to be very machine dependent.

Test 56

The operator $\uparrow$ is very difficult to implement correctly because it dynamic type checking. Most systems redefine the operator so as to avoid this. The other related difficulty is that there are 18 different cases depending upon the type and sign of the assignments. Ten of these are valid and eight are invalid. This test program attempts all the ten valid cases and one invalid one.

The program rises real equality against the defining formula given in the Report. It can be argued that equalily is not to be expected because, unlike integer divide, the operator is not defined by an ALGOL algorithm (and so any algorithm with different round-off properties could be used). Nevertheless, failure of the equality test does indicate that further analysis shiould be done.

Tho test compiles and fails to execute in both systems on the final computation of (-2.0)↑ 3.0. Note that this final test could give the answer -8 if the real nature of the arguments is ignored. Whetstone ALGOL prints the value of the arguments on failure, and the Kidsgrove system produces a stores print which indirectly qives the value of the arguments.

Test 57

Attempts to calculate 0.0↑ 0. Variables are used so that the calculation of constant expressions by the compiler is not used. Ideally this should be checked as well.

In both systems the program compiles and fails to execute in the manner required by the Report.

Test 58

Attempts to calculate 0 ↑ 0, which is invalid. Fails on execution in both systems.

Test 59

Attempts to calculate 0.0 ↑(-1.0). Fails on execution in both systems, but the post mortem print in the Kidsgrove system also failed which has since been corrected.

Test 60

As for test 59 with 0 ↑(-1.0). Results as for test 59.

Test 61

As for test 59 with (-2) ↑ 3.2. Results as for test 59.

Test 62

Test to check that 0 ↑ (-2) fails on execution. This fails 'overflow in ↑' in the Whetstone system. Produces the answer 0 with no error indication in the Kidsgrove system. This was due to clearing the hardware overflow flag. This error has now been corrected.

Test 63

As for test 62 with 0.0 ↑ (-2).

Test 64

It is not clear from the ALGOL Report if a conditional expression can have a different type depending on the result of the condition. In this test, the two types are integer and real, but when a real type is produced it must fail because integer divide can only have integer arguments. In the author's opinion this test should fail at compile time.

Both the Whetstone and Kidsgrove systems compile the test and execute the condition successfully if an integer value is produced (printing 0.0). The Whetstone system performs a dynamic type check. The Kidsgrove compiler always produces a floating point answer from such conditions, but if the value is not integral, the integer divide operation will fail. This technique is not formally correct, since the expression (**if true then** 1.0 **else** 2) ÷ 6 will not fail. Nevertheless it is probably as near to the Report as can be managed.

Test 65

The type of the result <integer> ↑ <integer> depends upon the sign of the exponent. Dynamic type checks are therefore required if the Report is followed in detail. This test calculates 2 ↑ 3 ÷ 3, which should probably be rejected at compile time.

Both the Whetstone and Kidsgrove compilers accept this, and the methods used follow that noted in test 64. The value printed should be 2.

Test 66

Tests that the precedence of the integer operators is apparently correct. Should print the number 25. Also contains a test similar to number 65, 50 it could fail just because of this.

Compiles and executes successfully with both systems.

Test 67

This test is similar to number 65 and should fail to execute. Compiles but fails to execute in both Whetstone and Kidsgrove.

Test 68

There is a difficulty in ALGOL 60 concerning switches and for loops. Jumps into the middle of a for loop are not allowed although the for loop statement does not necessarily constitute a block. Apart from improving program clarity, there are good practical reasons for this, namely that the control variable may be undefined, and the action with multiple for list elements will be indeterminate.

Since a switch list may involve several labels, any of which could oe selected in principle by a statement of the form **goto** s[i], all these labels must be such that no jump into a for loop can be made.

This test consists of a switch declared outside a for loop, but all the labels and uses of the switch are inside the loop. Thus there is no possibility of violating the rule about jumping into a for loop, although a compiler may have difficulty in checking this. Since the switch could more reasonably have been declared in the for loop, a compiler could reject this program on the grounds that it cannot make the necessary checks at compile-time. In some systems, the check is made at run-time.

The Whetstone compiler treats the body of the for loop as a block, so that the program fails to translate because the switch labels are undeclared. The Kidsgrove compiler translates and executes the program successfully (there is no output).

Test 69

This program contains a switch which, unlike test 68, could not possibly be shown to be valid at compile-time. The switch labels straddle a for loop), but no invalid jump is made.

Fails to compile with Whetstone because the body of the for loop is regarded as a block. Compiles and executes successfully with the Kidsgrove compiler, printing the numbers 1, 3 and 3.

Test 70

A test very similar to that of number 69, but it contains an invalid jump into a for loop. Since the body of a for loop containing multiple for list elements is usually implemented as a subroutine, run-time failures could result.

Fails to compile with Whetstone because the body of the for loop is regarded as a block. Compiles but fails to execute in Kidsgrove system. The failure was caused by jumping to an undefined location at the end of the for loop. This illustrates that the Kidsgrove compiler only handled tests 68 and 69 correctly because it made no check on the validity of the use of switches.

Test 71

One difficulty in implementing an efficient mechanism for calling procedures is handling parameters. A formal procedure call cannot always have its parameters checked at compile time because the parameter specification of the actual procedure is not necessarily fixed. Various techniques have been adopted to overcome this. The Whetstone system, being interpretive, can afford to check all parameters at run-time, but this cannot be regarded as a very effective solution for a true compiler. The KDF9 Egdon ALGOL compiler insists on a comment giving the parameter specification to each formal procedure. The Atlas ALGOL and Kidsgrove compilers attempt to determine the specification from actual calls, failing if this is inadequate.

The Whetstone compiler insists that in any block a procedure is called with the same number of parameters. This test complies with that restriction, but the actual procedures have a different parameter specification so that compile-time checking is impossible. The test works with the Whetstone system, printing the number 5. It fails to compile with the Kidsgrove system, giving the message 'formal level statement without an actual statement including formal', and giving no indication of the text position or identifier concerned.

Test 72

This test is identical to that of test 71 except that a slight change has been made so that there is no longer a fixed number of parameters to the formal procedure. It therefore fails to translate with the Whetstone compiler. The Kidsgrove compiler gives the same error message on translation as before.

It is probably an acceptable restriction that procedure parameters should always have the same specification. Unlike the Kidsgrove system, compilers should produce adequate information to diagnose the error, which in this case may involve both the formal procedure parameters and the actual parameters.

Test 73

This is a performance test for recursive procedure calls, which has been used to compare a number of different systems [24]. The recursive procedure calculates Ackermann's function using the definitive algorithm. In the original paper, Sundblad measures two figures — the rate at which procedures are called in the evaluation, and the maximum n for which Ackermann (3, n) can be calculated. The second figure is in most implementations limited by the core size required for the stack. As the sise required for the stack doubles for each successive value of n, this is not a good measure. It would be better to state the number of words required for each recursion of the function (although this may not necessarily be available).

The program calculates Ackermann (3, n) for n upwards, noting the total time and the time per call on each iteration. The value of the function is also checked.

The test compiled and executed successfully in both the Whetstone and Kidsgrove systems. Whetstone took 11 milliseconds per procedure call and Kidsgrove 532 microseconds. No attempt was made to run the tests for the highest possible value of n since the rate of call of procedures did not change. An error was found in

the post-mortem routine in the Kidsgrove system in that it attempted to print out information on all the nested calls - sometimes over 300. This produced excessive line-printer output, and so was modified to print only the last 20 recursive calls. Each depth of recursion requires ten words, so it was possible to calculate the exact time and space requirements of the subsequent values. In fact, with Sundblad's standard core image (26K words), it is possible to calculate Ackermann (3,8) on KDF9 in 1482 seconds, which is further than any other known system has taken it. Sundblad therefore defines the 'capability' of the KDF9 to be 8. The results known to the author are:

Language/computer	Time per call (microseconds)	Capability
Algol 60, 360/75, F	870	3
Algol W, 360/75	103	6
Algol W, 360/67	121	6
Algol 60, CDC 6600, v1.1	470	6
Algol 60, CDC 6600, v2.0	410	4
Simula-67, CDC 6600, v1.0	354	6
PL/I, 360/75 v4.3,	270-550	5
Algol 60, KDF9, Mk2	532	8
Machine-code, KDF9	103	10

Bearing in mind the speed of the machine, the KDF9 is significantly better than any of the other systems, although ALGOL W is also good especially if a subsequent improvement of 20 per cent in the times are taken into ascount.

The reason for the good figures from KIDF9 is that all parameter checking is performed at compile-time and the calculation of value parameters is handled in a pre-call sequence. Similar remarks apply to ATLAS [25].

The last entry in the above table is that given by hand coding the function using the same logic. Recursive linkage was not set-up unless the call was genuinely recursive, but otherwise the coding was not optimised to any great extent. Three words were required for each recurisve depth — one for each parameter and the third for the return address,

Test 74

The Report states that the evaluation of an expression in a bound-pair list cannot involve local variables. However, since the bound-pair lists are expressions within a block, their position is somewhat anomalous. In practice, it would be better if these expressions were regarded as being in the outer block to the array declaration, but this is not the case.

This test program, which is invalid ALGOL 60, illustrates the difficulty. The bound-pair list involves an integer n, declared both locally and in the outer block. The compiler should use the local n (which is declared after the array declaration) and hence fail. One-pass compilers could have difficulty with this unless, as in Whetstone, an explicit test is made far this error.

Fails to translate in both Whetstone and Kidsgrove compilers with a message that a local variable has been used in an array bound-pair list.

Test 75

This is an input-output test, similar in many respects to test 55. The specification of the IFIP input-output procedures states that 'outreal' and 'inreal' should be exact converses of each other. Therefore reading a real value previously printed

should give the same bit-pattern, This can only be achieved by allowing excess decimal digits to be specified in the format for output (excess, that is, over the representation in binary in the computer). This is a very tight constraint which is not likely to be achieved unless special care has been taken in designing the input-output routines.

The test program outputs 99 fractions, and reads the same values. The program must be run twice in order to construct the correct data. The numbers read are compared against the computed value of the fraction.

The results with Whetstone and Kidsgrove are the same, since the input-output package is the same. In fact

input > output 8 times
input = output 68 times
input > output 23 times

The tests were also run with the Babel system [22]. The output has a defect in that only 11 decimal places can be specified. Reading 12-digit ALGOL data gave the results (13, 68, 18) that is, much the same as ALGOL but with less bias. With 11 digit output as produced by Babel, the results were (40, 13, 46), which cannot be regarded as satisfactory.

Test 76

This program tests that variables can be used before being declared in the source text. The test is so constructed that a one-pass compiler may compile the program because outer block variables have the same identifier. Compilers not allowing use before declaration should not compile this program, and should give an intelligible error message. Otherwise it should compile and execute successfully, printing the number 0.

Compiles and executes successfully in both KDF9 ALGOL systems.

Test 77

The next two tests exercise the procedure parameter mechanism. Various obscure points connected with particular parameters are covered by special tests elsewhere.

An actual parameter can be one of five basic forms according to the Report. In these tests, an expression is one of three types, namely designational, boolean or arithmetic. A formal parameter can be specified to be of one of thirteen different types, seven of which can be listed as being by value. This gives a total of $20 \times 7 = 140$ different pairs of formal-actual parameters. In practice the variation is even greater than this because an array identifier can be of three types, and a procedure identifier of four types. Of the 140 cases considered here, 26 are valid and 114 invalid.

This test uses all the 26 valid cases in a short calculation to demonstrate the mechanism. The single number 0 should be printed. The test compiled and executed successfully with the Whetstone system. A translation failure occurred with the Kidsgrove compiler, the reason being given as 'generator nest with more than one result (conditional expression)'. The meaning of this is not known and as the position of the source text error is also unknown, the compiler did not perform this test satisfactorily.

Test 78

This test contains an instance of each of the 114 invalid formal-actual parameter pairs. Ideally, the test should fail to translate, listing a good number of these invalid

parameters.

The test compiles in the Whetstone system. This is a substantial deficiency in the system and stems from the fact that almost all type-checking is left to run-time. The program fails on execution as expected. The program fails to translate with the Kidsgrove compiler giving the expected error message (but only one error indication).

Test 79

This test was written by Naur [19] to demonstrate the type-checking capabilities of the GIER ALGOL compiler. This version contains 27 type-check errors. It also uses variables before declaration, and uses a local variable in a bound-pair list. An interesting observation is to note how many errors are detected in one compilation attempt.

The Whetstone system produced eleven error messages from the compilation. The first four errors were easily identified with actual source text errors, but the next four could not be pinpointed. Being a one pass compiler which does not restrict the language to no use before declaration, details of the actual declarations must sometimes be deduced during compilation. This is not always successful, so many checks are left to run-time.

The Kidsgrove compiler only found one error 'BPL variable local to current block', evidently complaining that the variable i, appearing in the bound-pair list, was local to the block.

Test 80

Compiler-writers have difficulty in implementing a strange feature of ALGOL that the control variable for a for loop can be a subscripted variable. It can be argued that this is not legitimate as the Report refers to *the* control variable. Certainly many implementations reject this, not without justification. The compilation arises because the address of the control variable must be re-evaluated at least once per loop.

This program is a straightforward one, testing that a simple use of a subscripted control variable to initialise an array apparently works. A single number 0 should be printed. Compiles and executes successfully with both the Whetstone and Kidsgrove systems.

Test 81

As pointed out under test 80, the exact interpretation to be placed on the use of a subscripted variable as a for loop control variable is in doubt [12]. This test program, designed by Knuth, gives a different result according to the number of evaluations of the control variable address, step and limit in a for loop. The program may output a value between 4 and 23, in fact both KDF9 systems produce 18.

From the point of view of compiler validation, any value between 4 and 23 should be acceptable, but ideally the value should be deducible from the compiler specification. It is more important that the program should compile, or if it fails to translate because of a language restriction, the error message produced should be intelligible.

Test 82

A dubious feature of ALGOL 60 is that functions can be called by means of a procedure statement. It can be argued from the Report that this is not allowed as

the purpose of a function is to deliver a value, but it is not specifically excluded. It is a sensible restriction, although there is little difficulty in the implementation.

Both systems compiled and executed this successfully, printinig the number 1.

Test 83

Another dubious feature of ALGOL 60 is that a **goto** to a switch element out of its bounds should be regarded as a dummy. The Report is very explicit an this point, although the ECMA and IFIP subsets of ALGOL 60 specifically exclude this (and regard it as an error). There are significant difficulties in implementation, and as Knuth [13] has pointed out, it is impossible in any practical system to remove any side-effects of the subscript evaluation. This test program by Knuth illustrated this point. The program prints the 20th Fibonacci number 6765 provided out of bound switches are implemented as dummies. The program also illustrates that recursion of a procedure can be accomplished via a switch (and hence is difficult to spot).

The program translates and fails on execution in the Whetstone system with the expected error that a switch index is out of range. The program translates but fails in an unexpected manner in the Kidsgrove system. This has been traced to not preserving a register for the evaluation of a switch index involving something complex — in this case a recursive function call. The Kidsgrove compiler classifies procedures and determines wiiether they are recursive. The compiler misclassified the two procedures as being non-recursive, presumably because the switch was omitted.

Test 84

This is basically a performance test, derived from the GAMM measure [6]. This measure is the weighted average of the time taken to execute five simple numerical calculation loops; adding two vectors of thirty elements, multiplying two vectors, a polynomial loop of ten terms, finding the maximum of ten elements and evaluating a square root by Newton's method for five iterations. The loops can be programmed in any language, and in this case ALGOL 60 is used. In fact the loops are repeated with the arrays involved as name parameters to a procedure, to see if this makes any difference to the times taken. The same program was used to evaluate five ALGOL compilers [25].

The program prints 96 numbers, the 13th and last being the value of the square root of 2, the 8th and 21st being about 1.45423 while the rest give the loop times. The GAMM figure is the time (usually quotes in mircroseconds) calculated from the average of the five loop times t1, t2, t3, t4 and t5 by the formula

$$GAMM = (t1/30 + t2/60 + t3/20 + t4/20 + t5/15)/5$$

The GAMM figures as calculated by this program are:

Language	Computer	GAMM (μs)
Algol W	360/67	12.9 (in program)
		17.5 (in procedure)
Algol 60	KDF9 (Kidsgrove)	134
	KDF9 (optimiser)	53.4
Algol 60	KDF9 (Edgon)	97.6
Algol 60	6600 (Mk2)	12.1
Algol 60	1108 (NU)	12.2
Algol 60	1907 (XALT)	48.9
Algol 60	Atlas	56.4
Algol 60	B5500	63.7

The time taken to execute the program is about $10^6\times$ GAMM, which can be reduced by altering n appropriately.

It is important to remember that with the GAMM figure compiler optimisation can be very effective because the loops are so simple. The program can be used to show the limits of compiler optimisation, as it is unlikely that optimisation could be more effective on ordinary programs. It cannot be regarded as a good benchmark because the use of the various features of ALGOL is very one-sided — only real one-dimensional arrays and simple for loops.

Test 85

This test is one of a number provided by Mr M Woodger from various sources, and used in a comparison of five compilers [25]. The test is not a difficult one, but the program contains a number of features not usually found in ordinary programs. These are, procedure with a dummy body, use of a boolean before declaration, redundant brackets in a boolean expression and an expression involving a parameterless function, type conversion and variables at different block levels.

The program compiled and executed successfully with both the KDF9 systems, printing the single number -5.5. The program also ran successfully on the CDC 6600, 1900, 1108 and Atlas compilers. A slight modification was required on the B5500 owing to the use before declaration.

Test 86

This is another test from Woodger's collection. The program contains a few unusual features viz: labels in peculiar positions, a complex designational expression, use before declaration, a nested procedure and a formal integer procedure.

The program compiled and executed successfully with the Whetstone system, warning that the identifiers d,e, f and g were declared but not used. The output is the two numbers 1 and 3. The program failed to compile with the Kidsgrove compiler although it worked in the test reported in the Computer Journal [25]. The reason for this is that labels to programs are ignored by the compiler, so that it failed 'prog' not declared. Inserting and extra **begin end** round the text allows correct compilation and execution.

The program ran successfully on the CIDC 6600, 1900, 1108 and Atlas compilers. Several modifications were necessary for execution on the B5500, because of use before declaration, an identifier clashing with a reserved word and the inability to address variables declared in the outer procedure of two nested procedures.

Test 87

Knuth [13] points out that the meaning of the step-until element with a real control variable is not clear. As an example, he gives this test program where the step value is the for loop control variable itself. In practice, a compiler is likely to have little difficulty in implementing this, although it must be agreed that the loop is very odd.

The program compiles and executes successfully with both the Whetstone and Kidsgrove compilers, printing the number 25.

Test 88

The question of side-effects in ALGOL 60 was left unresolved in the Report. Such effect occur whien a function produces some other effect apart from delivering a

value. Some implementations, notably ALCOR [2], prohibit side-effects. If side-effects are allowed (and it is difficult to detect them at compile-time) then various ALGOL 60 constructions become sensitive to the order of evaluation of consistuent expressions. Knuth [13] illustrates this with a test program which is copied here.

Compiles and executes successfully with both the KDF9 systems, printing the value 1/3. In view of the substantial difference in the implementation of these two compilers, it is surprising that they give the same value on ibis test.

Test 89

This is an example of the use of the General Problem Solver of Knuth and tierner as given by Randall and Russell [20]. The test shows how complex the call by name mechanism can become in ALGOL 80. The result of the procedure is checked by real equality which could cause an error on correct results owing to rounding errors. If the check is successful, only the number is printed.

Compiles and executes correctly with both KDF9 systems.

Test 90

This in the third program from Woodger's collection. It contains sone arrays with complex bounds, a string including compound basic symbols and some unusual for loops and expressions.

The program compiled and executed successfully in both KDF9 systems. The program prints three numbers, 9, 4 and 0. Owing to real to integer rounding, the first two numbers could be 8 and 3 without being formally incorrect. The program ran correctly on the CDC 6600, 1900 and 1108 compilers. On Atlas minor modification was necessaring owing to the end within a string as this caused the preprocessor to get out of step. On the B5500, modifications were necessary to overcome use before declaration, and to the remove the string, which is not permitted in this version of ALGOL.

Test 91

This is the fourth program from Woodger's collection. It contains a complicated switch, parameter comments on a call and declaration, and several different representations of 0. Strictly, it is not valid ALGOL 60, since the value of the control variable 'c' is used after exhaustion of the for loop. It should print the number 0.

The program compiles successfully with the Whetstone system, producing warning of an end comment with a delimiter and variables a, b, c, d, f, h and j declared but not used. The execution was also successful. The program failed with the Kidsgrove compiler 'switch element not in a parameter list or following a goto'. The program does contain a switch element in a switch list, but a similar example was accepted in test 83.

The program ran correctly on the CDC 6600, 1108, 1900, Atlas, and with minor modifications on the B5500. A parameter comment had to be removed on the 1108.

Test 92

This is the fifth of the Woodger collection. It contains two integer procedures which are called formally. It should print the number 42.

The program compiled and executed correctly with both KDF9 systems. It also ran successfully on the CDC 6600, 1108, 1900, B5500 and Atlas computers.

Test 94

This program is a performance test devised by Knuth [11]. It was called 'Man or boy?' because only man-sized compilers were supposed to be able to compile and execute it successfully. This version calculates the tenth number in the series defined by the main real function and prints three numbers, the middle one being -67. The time that this takes is taken as the performance measure, which is the difference of the first and last numbers printed. The program has been slightly modified to avoid calling a function by a procedure statement. The execution involves recursive function calls, nested procedures and very heavy use of name parameters.

The program compiled and executed successfully with both KDF9 systems, requiring about 12,500 words of storage for the stack and program.

The time taken to execute the program on a number of systems is as follows:

KDF9 Kidsgrove	2.0 seconds
KDF9 Whetstone	11.6 seconds
CDC 6600	.175 seconds
NU 1108	.295 seconds
1.1μs 1907	7.0 seconds
Atlas	.431 seconds

The program cannot be run on the B5500 because of both nested procedures addressing problem.

Test 94

This program is another from Woodger's collection, and tests for loops. It involves some severe side-effects on the evaluation of expressions. For this reason, the number printed may be different from 232, the number obtained from strict left to right expression evaluation, without being provably incorrect. The most likely answer otherwise is 133, and any other is likely to indicate an error.

Compiled and executed successfully on both KDF9 systemmis. Also ran correctly on CDC 6600, Atlas and B5500. The alternative answer, indicating a different order of expression evaluation, occurred on the 1108 and 1900.

Test 95

This test was written by Woodger to include various combinations of formal procedure call. The actual procedure parameters always have the same specification, indicated by a commnent (which is that required by the Egdon ALGOL compiler for KDF9).

The Whetstone compiler successfully executes the program printing the number 7. The Kidsgrove compiler fails on translation giving the message 'formal level statement without an actual statement including formal'. Apparently, this actually means that the compiler was unable to deduce the specification of the formal procedure parameters, required by the compiler for successful code generation.

The program can correctly on the CDC 6600, 1108, 1900, and Atlas. Modification was required on the B5500 to allow access to a variable outer to two nested procedures, but then executed correctly.

Test 96

This is a further test from the Woodger collection. It tests labels at different block levels and label parameters, including one for a formal procedure.

The program compiled and printing the single number 5. The program ran correctly on the CDC6600, 1900, 1108 and Atlas computers. It failed to translate on the B5500 because of a restriction prohibiting label parameters to functions. This restriction could conversion of some programs, but it is not a common requirement.

Test 97

This program was written by Dijkstra to test Jensen's device on the Xl compiler, which ran successfully in August 1960. It compiled and executed successfully cim both KDF9 systems producing the number 16 for the double summation.

Executed correctly on the CDC 6600, 1108, 1900, Atlas and B5500 systems.

Test 98

This is a further test from the Woodger collection. It contains a test of name parameters, access to which is recursive because of a nested call of a function. The program should print the single number 23. Compiled and executed successfully on both KDF9 systems. It also ran correctly on the CDC 6600, 1108, 1900, B5500 and Atlas computers.

Test 99

The final test from the Woodger collection. It containis an array by value with type conversion, an array declared with large values to the subscript bounds and an assignment statement with a large left-part list. It should print the single number 14. It compiles and executes successfully with both the KDF9 systems. It also ran correctly on the CDC 6600. Atlas and 1103 both required the removal of the type conversion on the value array. On the B5500, slight modifications were necessary to give the dimensions of formal arrays and to overcome use before declaration. The test failed on the 1900, producing the value 13 instead of 14. The reason for this is not known.

Test 100

A simple typing error lead to the discovery of an omission in the Whetstone compiler of a check on the syntax. A declaration **integer** i, **array** a [1,1] is accepted with the array declared as an integer array. The error has since been corrected by the introduction of a new state variable which is set if a comma appears in a type declaration.

The program failed to compile with the Kidsgrove systemn.

Test 101

The program contains an elementary syntax error, typical of a punching mistake. It should fail to translate, and the clarity of the error message should be assessed. Whetstone fails program 'letter, digit, .; or $_{10}$ misused', whereas the Kidsgrove compiler gives no positional information and the message 'failure in selection matrix'.

Test 102

Elementary syntax error, similar to test 101. The Whetstone compiler gives the message 'adjacent delimiters inadmissible', whereas the Kidsgrove compiler produces no positional information and the message 'Nesting store not empty at end of statement'.

Test 103

Elemenitary syntax error similar to test 101. The Whetstone compiler gives the message ':= preceeded by a constant or used inadmissibly' whereas the Kidsgrove compiler gives no positional information and the message 'failure in selection matrix'.

Test 104

Programs contains an elementary syntax error by placing a comma between the thousands and hundreds in a constant. The Whetstone compiler fails 'misused comma or colon in ann expression' whereas the Kidsgrove compiler gives no positional information and the message 'failure in selection matrix'.

Test 105

Program contains an elementary syntax error by placing the exponent part of a real number in brackets. The Whetstone compiler gives the error message 'illegal number', whereas the Kidsgrove compiler with two messages (one for each bracket) 'invalid number' and 'bracket mismatch'.

Test 106

Program contains an elementary syntax error by placing a variabLe instead of an exponent part to a real number. The Whetstone compiler gives the error message 'letter, digit, . or $_{10}$ misused' whereas the Kidsgrove compiler gives the message 'invalid number'.

Test 107

Elementary syntax error caused by a missing multiply sign. The Whetstone compiler gives the message 'letter, digit, . or $_{10}$ misused' whereas the Kidsgrove compiler gives no positional information and the message 'Nesting store not empty at end of statement'.

Test 108

Elementary syntax error caused by a missing semicolon. The Whetstone compiler gives the message 'goto must not follow an identifier or a constant' whereas the Kidsgrove compiler produces the message 'statement flag wrong - Algol basic symbol used in incorrect statement/expression context'.

Test 109

Elementary syntax error whereby a constant is used as a statement. The Whetstone compiler produces the message 'constant used as a statement' whereas the Kidsgrove compiler gives no positional information and the message 'Nesting store not empty at end of statement'.

Test 110

Elementary syntax error caused by using a decimal point rather than a comma to separate a type list. The Whetstone compiler produces the message 'letter, dinit, $_{10}$ or misused', whereas the Kidsgrove compiler produces the messages 'various syntax errors in block or procedure head'.

Test 111

The program contains an elementary syntax error caused by omitting a closing round bracket in an expression. The Whetstone compiler fails with the message 'statement ends incorrectly' whereas the Kidsgrove compiler gives the message 'bracket mismatch'.

Test 112

The program contains two statements consisting of an identifier which is not a procedure identifier (but that of a simple variable). The Whetstone compiler produces two error messages 'current use of identifier inconsistent with previous uses' whereas the Kidsgrove compiler gives the erroneous message 'undeclared identifier' four times (once for each declaration and use).

Test 113

The program contains a call of 'outreal' with only one parameter. The Whetstone compiler fails 'wrong number of subscripts or parameters' whereas the Kidsgrove compiler gives no positional indication and the message 'number of formal and actual parameters not equal'.

Test 114

Program contains an elementary syntax error caused by omitting a do in a for statement. This program illustrates a defect in the error recovery in the Whetstone compiler as it got into a loop without reading any source text. The repeated error message was 'omission or error preceeds begin or begin follows :='. The Kidsgrove compiler produced two error messages 'statement flag wrong - Algol basic symbol used in wrong expression/statement context' and 'bracket mismatch'.

Test 115

Program contains an elementary syntax error caused by using a (instead of [in an array declaration. The program will translate in systems having only round brackets. The Whetstone compiler gives the message 'misused (in array or switch declaration'. The Kidsgrove compiler produces the messages 'various syntax errors in block or procedure head' anid 'no endmessage after end of program'.

Test 116

Program contains an elementary syntax error caused by omitting an identifier in a declaration. The Whetstone compiler produces the message 'declaration without identifier', whereas the Kidsgrove compiler gives 'various syntax errors in block or procedure head' and 'bracket mismatch'.

Test 117

This program and syntax error, two incorrect uses of identifiers, and an undeclared variable. The Whetstone compiler correctly locates three of the errors with the messages 'function designator used as designational expression' and two messages 'current use of an identifier inconsistent with previous uses'. The Kidsgrove compiler produces seven messages 'undeclared identifier' listing all the identifiers incorrectly used or not declared.

Test 118

The program contains an elementary syntax error caused by using a comma instead of a colon in an array declaration. The Whetstone compiler produces the message 'commas or colons wrong in array bounds' whereas the Kidsgrove program compiled and executed.

Test 119

The program contains an elementary syntax error caused by omitting a **then**. The Whetstone compiler produces the message 'goto must not follow an identifier or a constant', whereas the Kidsgrove compiler gives the message 'statement flag wrong - Algol basic symbol used in wrong statement/expression context'.

Test 120

The program contains an elementary syntax error caused by omitting a semicolon in a declaration. The Whetstone compiler produces the message 'real, integer or boolean misplaced' whereas the Kidsgrove compiler gives 'various syntax errors in block or procedure head'.

Test 121

A short, simple test of own variables. execute printing the numbers 3,6,12 and 24. The program compiled and executed successfully in both KDF9 systems.

Test 122

A short, simple test of own variables, having two own variables with the same identifier. The program should compile and execute printinig the numbers 3,0,0 and 0. Ran correctly on both KDF9 systems.

Test 123

A similar program to the last two, but is invalid owing to access of an own variable which has not been initialised. Failed on execution will the Whetstone system, giving the message 'access is made to a variable which has not been assigned a value'. Compiled and executed with the Kidsgrove system, printing -4 four times. Unless special hardware is available, it cannot be expected that this form of error can be trapped without substantial loss in processing speed.

Test 124

The next four tests contain an invalid form of number obtained by omitting necessary digits. The program prints the number '1'. The program compiles and executes in Whetstone ALGOL, printing 1.0, although the flow diagram given in Randall and Russell is correct. The compiler has been corrected. The program fails to translate with the Kidsgrove compiler with the message 'invalid number'. Programs transcribed from FORTRAN could contain this error.

Test 125

Contains the invalid number $1._{10}$ and the program fails to translate in both KDF9 systems giving the message 'invalid number'.

Test 126

Contains the invalid number $_{10}+$ which is trapped by the Kidsgrove system but not the Whetstone compiler (which gives 1.0).

Test 127

Contains the invalid number $_{10}$, which is trapped by both KDF9 systems.

Test 128

Contaiiis the invalid number ., which is trapped by the Kidsgrove system but not the Whetstone compiler (which gives 0).

Test 129

The test checks that underflow is handled correctly with constants. In fact, the Whetstone compiler fails the program with the message 'number too large'. On KDF9, underflow is not trapped by the hardware, so for consistancy with this the program should translate printing the value 0, which is the action taken by the Kidsgrove compiler.

Test 130

A test similar to no 22 which checks labels in recursive procedures. In this case, the labels are not parametric. The program should compile and execute printing the numbers 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9, 8, 1, 6, 5, 4, 3, 2. Correctly handled by both KDF9 systems.

10 Test summary

The 130 test programs are listed here, giving the results with the Whetstone and Kidsgrove compilers together with a short description of the test itself.

Apart from the test number and description, the other columns are as follows:

Expected Action. A subjective judgement by the author

- T translates
- E executes
- F fails (on translation or execution)

For instance:

- TE program should translate and execute without error
- TF program failure on execution, but not on translation
- FE should fail to translate, but if it does, execution is expected (for example, with a trivial syntax error)

Whetstone, Kidsgrove Action. Action is classified four way, which in increasing gravity are:

- C correct
- R action contrary to the Report
- M action contrary to the Manual

- D no diagnostic

Examples

- CC correct action on translation and execution
- CM translates, but action on execution contrary to Manual

No	Expected Action	Whetstone Action	Kidsgrove Action	Description
1	FF	C-	D-	Overflow on large integer and real number
2	TE	CR	CC	Long identifier causing incorrect answer
3	TE	CC	M-	Labelled compound statement as program
4	TE	CC	CC	Redefinition of standard function name
5	FE	C-	C-	Labelled statement that is not a program
6	FE	C-	C-	Double-sided relation
7	TF	CC	CC	Invalid assignment to name parameter
8	FE	C-	C-	Declaration after statement
9	TE	CC	CC	Valid but peculiar comments
10	TE	CC	CC	Comments and strings interspersed
11	FE	C-	C-	Invalid parameter-like comment
12	FE	C-	D-	Invalid parameter-like comment
13	FE	C-	D-	Labelled comment
14	FE	C-	D-	Invalid parameter comment
15	FE	C-	C-	Comments, strings and parameter comments interspersed
16	TE	CC	CC	Nested strings
17	TE	CC	CC	Use of basic-symbol-like identifiers
18	TE	R-	CC	Long identifiers
19	TF	CC	CM	Array parameter with incorrect dimension
20	TE	CC	CC	Array parameter with indeterminate dimension
21	TE	R-	CC	Array parameter varies statically, but is correct
22	TE	CC	CC	Recursive label level test
23	TE	CC	M-	Labels by value and by name
24	TE	CC	CC	Arrays by value
25	TE	CC	CC	Real number test
26	TE	R-	CC	Maximum redeclaration in a procedure
27	TE	CC	D-	PL/I environment control test
28	TE	CC	CM	Recursive level test using procedure parameters
29	TE	CC	D-	PL/I environment control test with consistent specification
30	TE	C-	C-	Recursive level test using label parameters
31	FE	C-	C-	if then if syntax error
32	TE	CR	CR	integer-real type conversion on name assignment
33	TE	CC	CC	Procedures niested 6 deep
34	TE	CC	CC	Blocks nested 15 deep
35	TE	CC	CC	300 variables in block
36	TE	CC	R-	60 identifiers in array segment list
37	TE	CC	CC	for loops nested 12 deep
38	TE	CC	CC	Conditional statements nested 24 deep
39	TE	CC	CC	Conditional expressions nested 9 deep
40	TE	CC	R-	Switch of 300 simple labels
41	TE	CC	CC	Procedure with 60 parameters
42	TE	CC	CC	Array of 12 dimensions
43	TE	CC	CC	Simple nested expression
44	TE	CC	CC	Complex nested expression
45	TE	CC	CC	Nested function call

No	Expected Action	Whetstone Action	Kidsgrove Action	Description
46	TE	CC	CC	Designational expression nested 6 deep
47	TE	CC	D-	6 labelled dummy statements
48	TE	CC	CC	Compound statements nested 24 deep
49	TE	CC	CC	15 variables on left hand-sid of assignment
50	TE	CC	CC	For list of 60 elements
51	TE	CC	CC	60 arrays declared in the same declaration
52	TE	CC	M-	Complex expression in all parts of the syntax
53	TE	CC	CC	Integer divide
54	TE	CC	CC	sign, entier and abs
55	TE	CR	CC	input of real niumbers
56	TF	CC	CC	↑, 10 valid cases, one invalid
57	TF	CC	CC	0.0 ↑ 0
58	TF	CC	CC	0 ↑ 0
59	TF	CC	CC	0.0 ↑ (-1.0)
60	TF	CC	CC	0 ↑ (-1.0)
61	TF	CC	CC	(-2) ↑ 3.2
62	TF	CC	CM	0 ↑ (-2)
63	TF	CC	CC	0.0 ↑ (-2)
64	FF	RC	RC	dynamic type conversion with if then
65	TE	CC	CC	dynamic type conversion with ↑
66	TE	CC	CC	Test of integer operators and priorities
67	TF	CC	CC	dynamic type conversion with ↑
68	TE	R-	CC	Valid switch inside for loop
69	TE	R-	CC	switch straddling for loop, each jump being valid
70	TE	R-	CD	Invalid jump into for loop via switch
71	TE	CC	R-	Formal procedure calls with different parameter specifications
72	TE	R-	R-	Formal procedure calls with different number of parameters
73	TE	CC	CC	Ackermann's function, recursive procedure calls
74	FF	C-	C-	Local variable in bound-pair list
75	TE	CC	CC	Checking input against output
76	TE	CC	CC	Use before declaration
77	TE	CC	M-	All valid formal-actual parameter pairs
78	FF	RC	C-	All invalid formal-actual parameter pairs
79	FF	C-	C-	Naur's test for CIER compiler
80	TE	CC	CC	Array variable as control variable
81	TE	CC	CC	Knuth for test
82	TE	CC	CC	Function call by procedure statement
83	TE	CR	CD	Knuth dummy switch
84	TE	CC	CC	GAMM test
85	TE	CC	CC	Woodger test t1
86	TE	CC	M-	Woodger test prog
87	TE	CC	CC	For logarithmic
88	TE	CC	CC	Knuth side-effects
89	TE	CC	CC	General Problem Solver
90	TE	CC	CC	Woodger test a:z:

No	Expected Action	Whetstone Action	Kidsgrove Action	Description
91	TE	CC	M-	Woodger test switch
92	TE	CC	CC	Woodger test p:
93	TE	CC	CC	Knuth's Man or boy?
94	TE	CC	CC	Woodger test tp16:
95	TE	CC	M-	Woodger test w2:
96	TE	CC	CC	Woodger test r9:
97	TE	CC	CC	Dijkstra's sigma test
98	TE	CC	CC	Woodger test r6:
99	TE	CC	CC	Woodger test tp25:
100	FE	MC	CC	Comma in declaration
101	F-	C-	C-	Elementary syntax error: + misplaced
102	F-	C-	C-	Elementary syntax error: variable missing
103	F-	C-	C-	Elementary syntax error: + misplaced
104	F-	C-	C-	Comma in middle of number
105	F-	C-	C-	Bracket round exponent part of numbr
106	F-	C-	C-	Variable used in exponent part of number
107	F-	C-	C-	Digit misplaced
108	F-	C-	C-	Semicolon missing
109	F-	C-	C-	Constant appears as statement
110	F-	C-	C-	Full-stop instead of comma
111	F-	C-	C-	Missing closing bracket in expression
112	FE	C-	C-	Variable used as procedure identifier
113	FF	C-	C-	Wrong number of parameters to outreal
114	F-	C-	C-	do missing
115	FE	C-	C-	Round brackets used for square brackets
116	F-	C-	C-	Identifier missing in declaration
117	F-	C-	C-	One syntax error anid two type-check errors
118	FE	C-	D-	comma instead of colon
119	F-	C-	C-	then ommitted
120	F-	C-	C-	semicolon ommitted at thie end of a declaration
121	TE	CC	CC	own variables
122	TE	CC	CC	own variables
123	TE	CC	CM	access to uninitialised own variables
124	FE	DC	C-	number ending in a decimal point
125	FE	CC	CC	Number with missing digits
126	FE	DC	C-	Exponent part of a number without a digit
127	FE	C-	C-	Number without a digit
128	FE	DC	C-	Number without a digit
129	TE	D-	CC	Constant which underflows
130	TE	CC	CC	Labels in a recursive procedure

The results can be summarised as follows. The Whetstone system passed all the tests except five in the sense that only five of the 130 test programs produced a result contrary to the manual (see test 100, 124, 126, 128, 129). However 13 tests were contrary to the Report, the only deviation likely to be of practical significance is that caused by introducing a block for every for statement (see tests 68, 69 and 70). Apart from the slow execution due to interpretation, the major defect is that a large amount of type checking is left to execution time, so that errors can remain dormant in code which is not executed (see tests 64 and 78).

The results from the Kidsgrove compiler cannot be regarded as satisfactory as 20 programs failed in a manner inconsistent with the manual, and nine of those did not produce any intelligible diagnostic. In practical terms, the most severe drawback is that no option is available for subscript checking and that compilation errors rarely give the source text position. Fortunately, the generation of invalid code is comparatively rare (but see tests 70 and 83). So the major difficulty with this

compiler is that some constructions, mainly formal procedure calls and switches, may fail even if the program is valid (although this is somewhat unusual). The actual error messages given in the manual could be considerably improved to avoid the use of jargon. Since the majority of program testing of KDF9 can be performed with the Whetstone system, the defects of the Kidsgrove compiler are not serious.

11 The ALGOL Basic symbol table

(omitted)

References

[1] AFSC (1970) User's Manual, COBOL compiler validation system. Directorate of systems design and development, HQ Electronic Systems Division (AFSC) L.G. Hanscom Field, Bedford. Massachusetts.

[2] Bayer, R. (1967), Gries D. Paul M and Wiehle, H.R. The ALCOR Illinois 7090/7094 post mortem dump. Comm ACM, Vol 10, No 12, pp804-808.

[3] Brown, P.J. (1967) The ML/1 macro processor, Comm ACM, Vol 10, No 10, pp618-623.

[4] Floyd, R.W. (1971) Towards interactive design of correct programs. IFIP Congress 1971 Vol 1, pp.7-1O North Holland, 1972

[5] Grau, A.A. (1967), Hill U, and Langmaack, H. Translation of ALGOL 60. Springer.

[6] Heinhold, J (1962) and Bauer, F.L. (Editors), Fachbegriffe der Programmierungstechnik. Ansgearbeitet vom Fachausschatz Programmieren der Gesellshaft für Angewandte Mathematik und Mechanik (GAMM) Oldenbourg, Munchen.

[7] Hill, I. D. (1968) ALGOL 60 Test programs. Private communication.

[8] Haxtable, D. H. R. (1963) and Hawkins, E. N. A multipass translation scheme for ALGOL 60. Anmual Review in Automatic Programming, Vol 3, pp163-205.

[9] IFIP (1964) Report on Input-Output procedures for ALGOL 60. Comm ACM, Vol 7, No 10 pp628-630.

[10] Knuth, D.E. (1961) and Merner, J. N. ALGOL 60 Confidential, Comm ACM Vol 4 No 6 pp 268-272.

[11] Knuth, D.E. (1964) Man or boy? ALGOL Bulletin, No 17, Page 7, Mathematishe Centrumn, Amsterdam.

[12] Knuth, D. E. (1965) A list of the remaining trouble-spots in ALGOL 60. ALGOL Bulletin No 19. Mathematishe Centrum, Amsterdam.

[13] Knuth, D. E. (1967) The remaining trouble-spots in ALGOL 60. Comm ACM, Vol 10, No 10, pp611-618.

[14] Lauer, P (1968) Formal definition of ALGOL 60. TR.25.088 1PM Laboratory, Vienna.

[15] London, B. L. (1970) Proving programs correct: Some techniques and examples. BIT Vol 10 No 2 pp 168-182.

[16] London, B. L. (1971) Correctness of two compilers for a LISP subset. Stanford Computer Science Department Report G5240.

[17] Lucas, P. (1971) Formal definition of programming languages and systems. IFIP Congress 1971. Vol 1, pp 291-297, North Holland 1972.

[18] Naur, P. (1963) Editor, Revised report on the algorithmic language ALGOL 60. Comm ACM, Vol 6, No 1, pp 1-17.

[19] Naur, P. (1965) Checking of operand types in ALGOL compilers. BIT Vol 5 pp 151-163.

[20] Randell, B. (1964) and Russell, L. J. ALGOL 60 Implementation, APIC studies in Data Processing No 5, Academic Press.

[21] Scowen, R. S. (1971), Hill, I. D., Hillman, A. L. and Schimell, M. SOAP — A program which documents and edits ALGOL 60 programs. Computer Journal, Vol 14, No 2, pp 133-135.

[22] Scowen, R. S. (1969) Babel, a new programming language. National Physical Laboratory Report CCU 7.

[23] Scowen, R. S. (1972) Debugging computer programs, a survey with special emphasis on ALGOL. National Physical Laboratory Report NAC 21.

[24] Sundblad, Y. (1971) The Ackermaun function, a theoretical, computational and formula manipulative study. BIT Vol 11, pp107-119.

[25] Wichmann, B. A. (1972) Five ALGOL Compilers. Computer Journal, Vol 15, No 1, pp8-12.