P/00/009
Regulation 3.2

AUSTRALIA

Patents Act 1990

DRAFT

PROVISIONAL SPECIFICATION

Invention Title: **METHOD FOR TESTING AND DEBUGGING COMPUTER PROGRAMS**

The invention is described in the following statement:

THIS INVENTION relates to a method of testing and debugging an executable behaviourally unknown computer program by comparing the behaviourally unknown computer program against an executable behaviourally known computer program. The invention is particularly useful, but not necessarily limited to, testing and debugging in which both programs have been compiled from the same source code or when both programs originate from similar algorithms having identical variable values at specific break points.

Conventional software debugging tools allow a user to interrogate the state and control the execution of a computer program. Such debugging tools require the user to have knowledge of the correct operation of the program in order to analyse various states to thereby find errors. Unfortunately, this may be a problem when the program is poorly structured, sparsely documented or when non-descriptive variable names are used.

Once a computer program (behaviourally known program) has been debugged and operates correctly on one type of computer, having a particular compiler and operating system, there is no guarantee that the program will behave as required either compiled by a different compiler or when run in a different environment.

Consequently, comparison methods have been developed in which the behaviourally known program,

functioning in a desired manner, is compared with a program under test (behaviourally unknown program) having the same source origin as the behaviourally known program. Comparison methods may also be used to test and debug programs originating from similar algorithms having identical variable values at specific break points. The currently available comparison methods are usually effected in an ad-hoc fashion in which either respective program states are dumped to a file and subsequently compared manually or two debugging sessions are required to manually compare respective program states.

If the state is written to a file, then a file comparison program is used to detect the differences. If the output is compared manually, the user must set breakpoints in the two programs, dump the variables manually, and then compare the values.

Further, a hybrid method has also been used in which two debugging sessions dump the values of the variables to a file after which a file comparison program is run to detect differences.

Although more efficient than conventional debugging tools, the above comparison debugging methods are relatively time consuming. Further, they do not directly display differences to a user nor do they not allow the user to interactively view either differences in program states or differences in specified variables.

It is an object of the invention to overcome or alleviate at least some of the problems with prior art comparison debugging methods.

According to one form of the invention there is provided a computerised method for testing and debugging an executable behaviourally unknown computer program the method including the steps of:

(i) controlling the execution of one or more instructions of an executable behaviourally known computer program;

(ii) obtaining a value of at least one variable resulting from step (i);

(iii) controlling the execution of one or more corresponding instructions of said behaviourally unknown computer program;

(iv) obtaining a value of at least one variable resulting from step (iii);

(v) comparing said value of at least one variable resulting from step (ii) with said value of a corresponding at least one variable resulting from step (iv); and

(vi) outputting differences resulting from said comparing.

Alternatively, according to another form of the invention there is provided a computerised method for testing and debugging an executable behaviourally unknown computer program, the method including the steps of:

(a) allowing the selecting of a break point for said behaviourally unknown computer program and an executable behaviourally known computer program;

(b) controlling the execution of said behaviourally known computer program until the break point is reached;

(c) obtaining a value of at least one variable resulting from step (b);

(d) controlling the execution of said behaviourally unknown computer program until said break point is reached;

(e) obtaining a value of at least one variable resulting from step (d);

(f) comparing said value of at least one variable of said behaviourally known computer program with said value of a corresponding variable of said behaviourally unknown computer program; and

(g) outputting differences resulting from said comparing.

Suitably, both said programs may have been compiled from the same source code origin. Alternatively, both said programs may have originated from similar or identical algorithms in which both said algorithms, when functioning in a desired manner, have identical variable values at selected break points.

Preferably, the method may be further characterised by the step of allowing the user the

choice of selecting which variables are to be compared by comparing step (v) or comparing step (f).

Suitably, the method may be further characterised by said programs being executed on different computer means. The method may be suitably effected on a computer means which executes the behaviourally known computer program. Alternatively, said programs may be executed on the same computer means.

Preferably, the method is further characterised by the step of outputting being effected by a visual display unit.

The method may also suitably include the step of interacting, after the step of outputting, wherein the interacting allows the user to control the method such that steps (i) to (vi) or steps (a) to (g) are repeated.

Preferably, the comparing step of (v) may include the steps of:

(vii) subtracting said value of at least one variable resulting from step (ii) with said value of a corresponding at least one variable resulting from step (iv); and

(viii) determining if the modulus resulting from the step (vii) is greater than a threshold value.

Preferably, the comparing step of (f) may include the steps of:

(h) subtracting said value of at least one variable of said behaviourally known computer program with said

value of a corresponding variable of said behaviourally unknown computer program; and

(i) determining if the modulus resulting from the step (h) is greater than a threshold value.

Suitably, the threshold value may be pre-defined.

In order that the invention may be readily understood and put into practical effect, reference will now be made to preferred embodiments illustrated in the accompanying drawings in which:-

FIG 1 is a flow diagram of a first embodiment of the invention;

FIG 2 is a flow diagram of a second embodiment of the invention;

FIG 3 is a flow diagram of the comparing step of FIG 1 and FIG 2;

FIG 4 is a flow diagram of a calculate function used in FIG 3; and

FIG 5 is pseudo-code illustrating steps in the calculate function of FIG 4.

Referring to FIG 1 there is illustrated a comparison method for testing and debugging a computer program in which the user selects an executable behaviourally known computer program or reference program Rp to be executed on a computer Cr shown at step 1. The user also selects an executable behaviourally unknown computer program which is the program to be tested Tp, shown at step 2, in which Tp is to be executed on a computer Ct. Program Tp has

been compiled from the same source code as the reference program Rp. Computers Cr and Ct are linked to communicate with each other and the selection of Rp and Tp includes the identification of their respective paths.

At step 3 at least one program instruction of Rp is then executed on Cr and a corresponding section of program instructions of Tp are executed on Ct at step 4.

By default a single program instruction is executed at each of steps 3 and 4. However, the user can determine the number of instructions to be executed at steps 2 and 3 or alternatively the user can select break points.

Values of variables of Rp and Tp are obtained (read) and then compared at 5 by a comparison program on Cr. These variables may be selected by the user if so desired. If a difference results, the differences are communicated at step 6 to the user via a visual display unit.

Certain errors such as inexact equality in floating point numbers are insignificant and therefore the comparing of step 5 returns a difference only when the result of the comparison is greater than a threshold value. The threshold value is set by the user and has a default value of 0. If the difference displayed at step 6 are insignificant a user may decide to continue debugging, at step 7, by selecting

the Yes option. Alternatively, the user may terminate the debugging, at step 7, by selecting the No option.

Referring back to step 5, if the comparison of Rp and Tp results in the same variable values being identified, steps 3, 4 and 5 are repeated until step 8 results in a Yes which corresponds to there being no more executable program instructions. Accordingly, an error free Tp status is reported at step 9 to the user via the Visual Display Unit.

Referring to FIG 2 a second preferred embodiment is shown in which the user selects the executable behaviourally unknown computer program or reference program Rp on computer Cr at step 10. The executable behaviourally unknown computer program which is the program to be tested Tp on computer Ct is selected at step 11. The paths of both Rp and Tp are also identified at steps 10 and 11.

Break points and variables to be compared are selected at step 12 and Rp is run on Cr at 13 until step 14 detects a break point. Tp is then run on Ct at step 15 until step 16 detects a break point.

The values of the selected variables of Rp and Tp are obtained (read) and then compared at step 17 in which the comparison is executed on Cr. If the comparison results in a difference, the differences are communicated at step 18 via the Visual Display Unit. The user may decide at step 19 to continue selecting break points and variables by selecting the

Yes option. Alternatively, the testing and debugging session can be terminated by selecting the No option.

If at step 17 the comparison of the variables is equal and the end test of Rp and Tp at step 20 results in a NO the method returns to step 12. Alternatively, if at step 20 the result is YES then the end of both programs Rp and Tp have been reached and step 21 reports to the user, via the Visual Display Unit, that there are no errors in Tp and the method terminates.

Referring to FIG 3 there is illustrated the comparing step (5 or 17) applicable both FIGS 1 and 2.

At step 22 respective variables A and B are selected from Rp and Tp. Variable A is then tested at step 23 to determine if it is of type dynamic data structure such as a pointer. If variable A is a dynamic data structure then at step 24 a graph of the data structures of both variables A and B is constructed. At step 25 each node in both graphs is compared by the calculate function of FIG 4 in which each node is compared in a depth first fashion.

If at step 23 it is determined that variable A is not a dynamic data structure step 26 determines whether or not variable A is an array or string. If variable is an array or string then at step 27 each element thereof is compared by the calculate function of FIG 4.

If at step 26 it is determined that variable A is not an array or string then it is assumed that A is an

integer, real, boolean or character. Accordingly, the calculate function of FIG 4 is executed once to compare A and B.

If at step 29 it is determined that there are more variables to be tested the comparing method returns to step 22, otherwise the method terminates.

Referring to FIG 4, step 30 determines if variable A is an integer or real number. If variable A is an integer or real number then at step 31 B is subtracted from A and the modulus of the result is compared against a threshold value. This value can be pre-defined if so desired. If the result is less than the threshold value then A and B are considered equal and an equal condition is returned at step 32. If the result is greater than the threshold value a difference condition is returned at step 33.

At step 30, if variable A is neither an integer or real then step 34 determines if A is a boolean or character. If A is neither a boolean nor a character an error condition is detected at 35 which terminates the method. If A is either a boolean or character then step 36 determines if A is the same as B and an equal condition is returned at step 32. If A is different from B then a difference condition is returned at step 33. After steps 32 or 33 the method terminates.

The threshold value provides a means of reducing or eliminating errors in inexact equality. This

threshold value can be either pre-determined by either a user or it may have an appropriate default value.

In addition to FIGS 3 and 4 reference is now made to the pseudo-code of FIG 5. The function simple_types_equal illustrates how FIG 4 is implemented. Note the subtraction operator and less than operator have been overloaded so that it can test integers, booleans, reals and characters.

The function one_dim_arrays_equal tests one dimensional arrays or strings. Because it is possible that arrays may be of different sizes (due to compilation differences or otherwise), the maximum of the lower boundary and the minimum of the higher boundary are considered.

The function multi_dim_arrays_equal is similar to that of the function one_dim_arrays_equal. As implemented, and as shown, a different function is required for each array dimension. However, a recursive function or a function in which the dimension of the array is passed to the function may be used.

The above embodiments describe two methods for testing and debugging by comparing two programs running on different computers. The method as illustrated in FIGS 1 and 2 are suitably for testing and debugging executable programs compiled from the same source code. The method as illustrated in FIG 2 is also suitably for testing and debugging programs

originating from identical of similar algorithms in which specific variables will be identical at selected break points. The methods as described are also suitable for testing and debugging programs running on the same computer, in such a case Cr and Ct are the same computer.

Although, the flow diagrams show the computers Cr and Ct running sequentially, they can also run concurrently. Further, both the embodiments may have a counter or timer for handling error conditions such as when one of the programs do not return from a break point or if an infinite loop is entered.

The invention has been described with reference to preferred embodiments, however, it is to be understood that the invention is not limited to the specific embodiments hereindescribed.

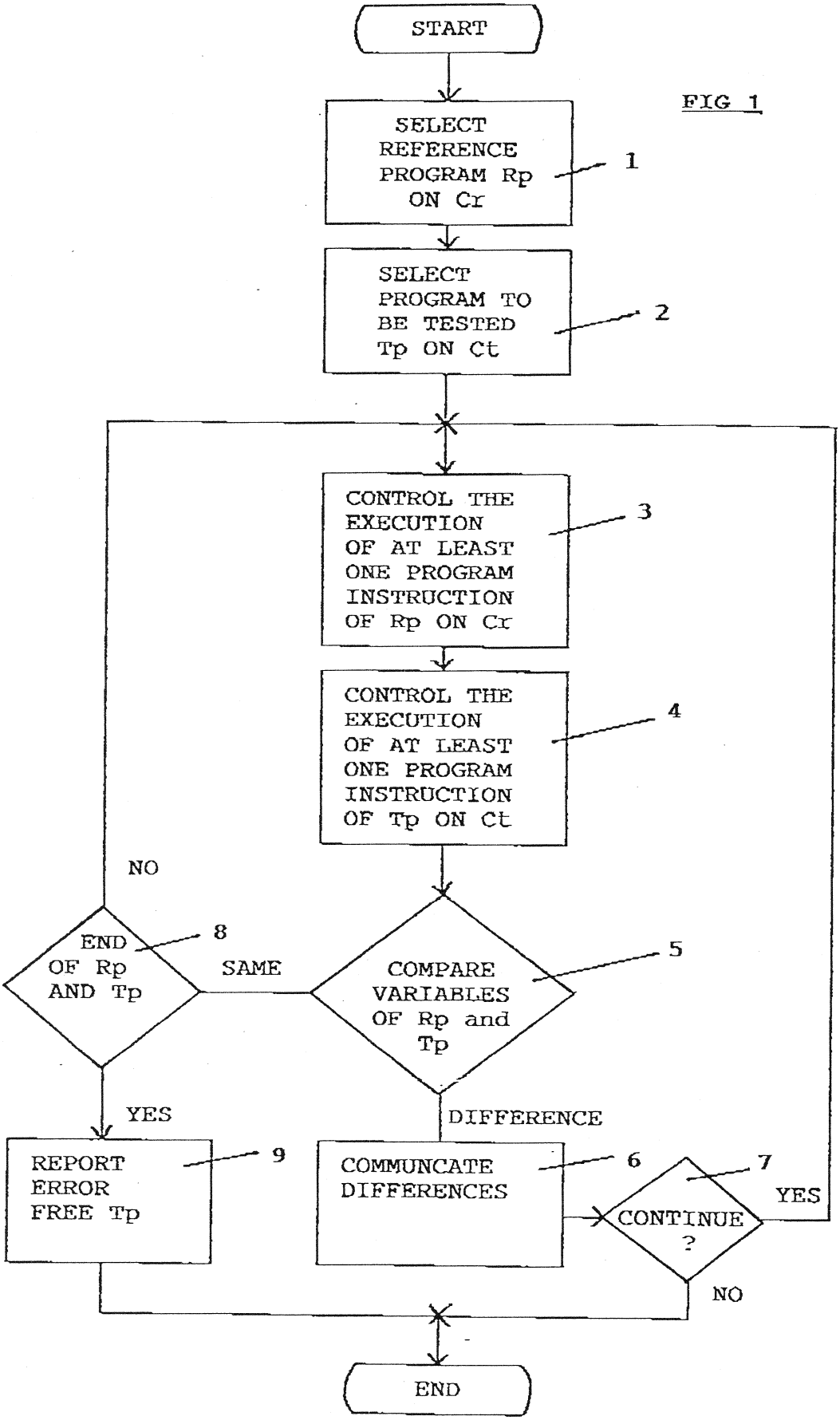
DATED this Fourteenth day of April 1994

GRIFFITH UNIVERSITY

By their Patent Attorneys

FISHER & KELLY

FIG 1


START
SELECT REFERENCE PROGRAM Rp ON Cr
1
SELECT PROGRAM TO BE TESTED Tp ON Ct
2
CONTROL THE EXECUTION OF AT LEAST ONE PROGRAM INSTRUCTION OF Rp ON Cr
3
CONTROL THE EXECUTION OF AT LEAST ONE PROGRAM INSTRUCTION OF Tp ON Ct
4
NO
END OF Rp AND Tp
8
SAME
COMPARE VARIABLES OF Rp and Tp
5
YES
DIFFERENCE
REPORT ERROR FREE Tp
9
COMMUNCATE DIFFERENCES
6
CONTINUE ?
7
YES
NO
END

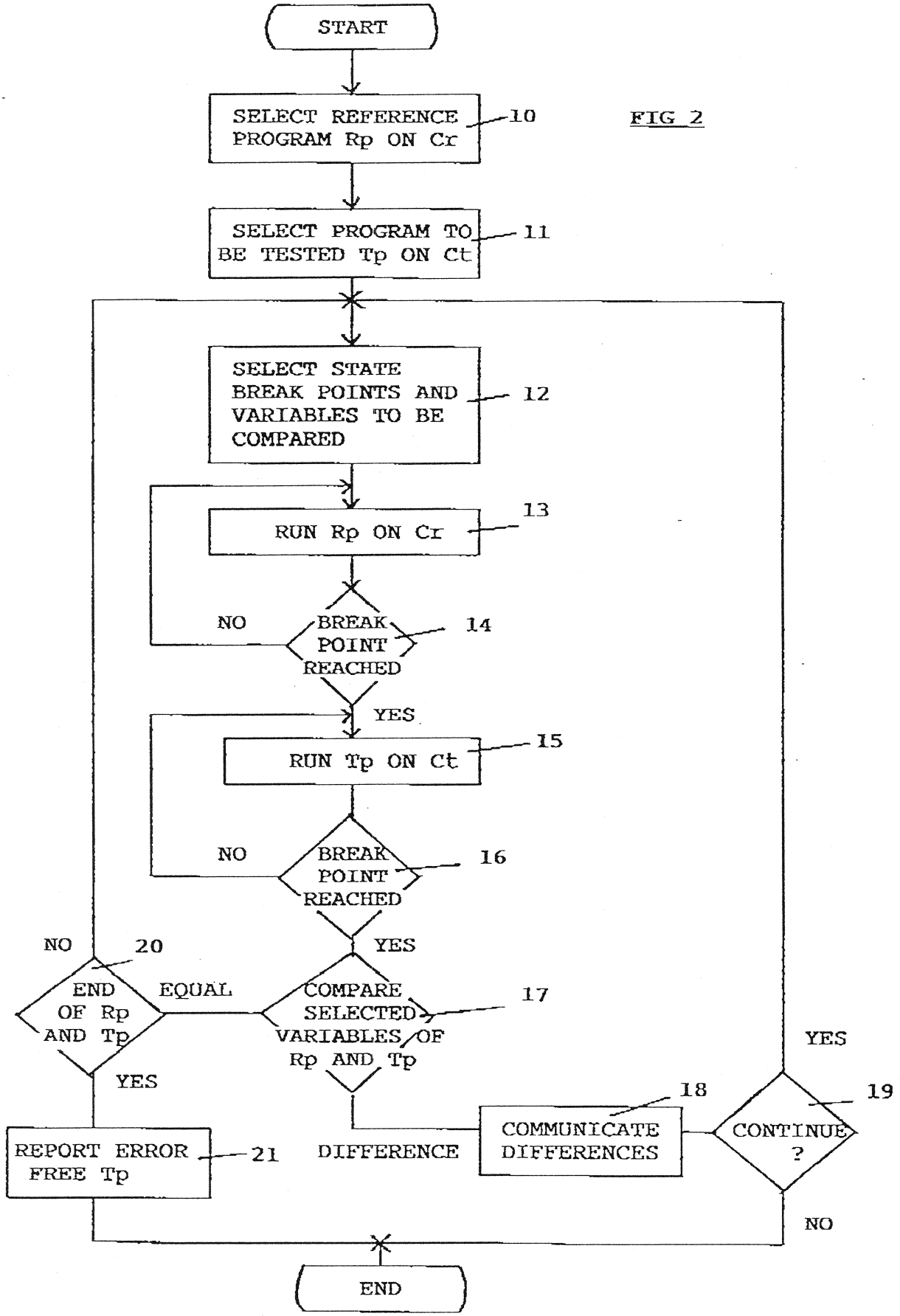

START
SELECT REFERENCE PROGRAM Rp ON Cr
10
SELECT PROGRAM TO BE TESTED Tp ON Ct
11
SELECT STATE BREAK POINTS AND VARIABLES TO BE COMPARED
12
RUN Rp ON Cr
13
NO
BREAK POINT REACHED
14
YES
RUN Tp ON Ct
15
NO
BREAK POINT REACHED
16
YES
NO
20
END OF Rp AND Tp
EQUAL
COMPARE SELECTED VARIABLES OF Rp AND Tp
17
YES
REPORT ERROR FREE Tp
21
DIFFERENCE
COMMUNICATE DIFFERENCES
18
CONTINUE ?
19
YES
NO
END

FIG 2

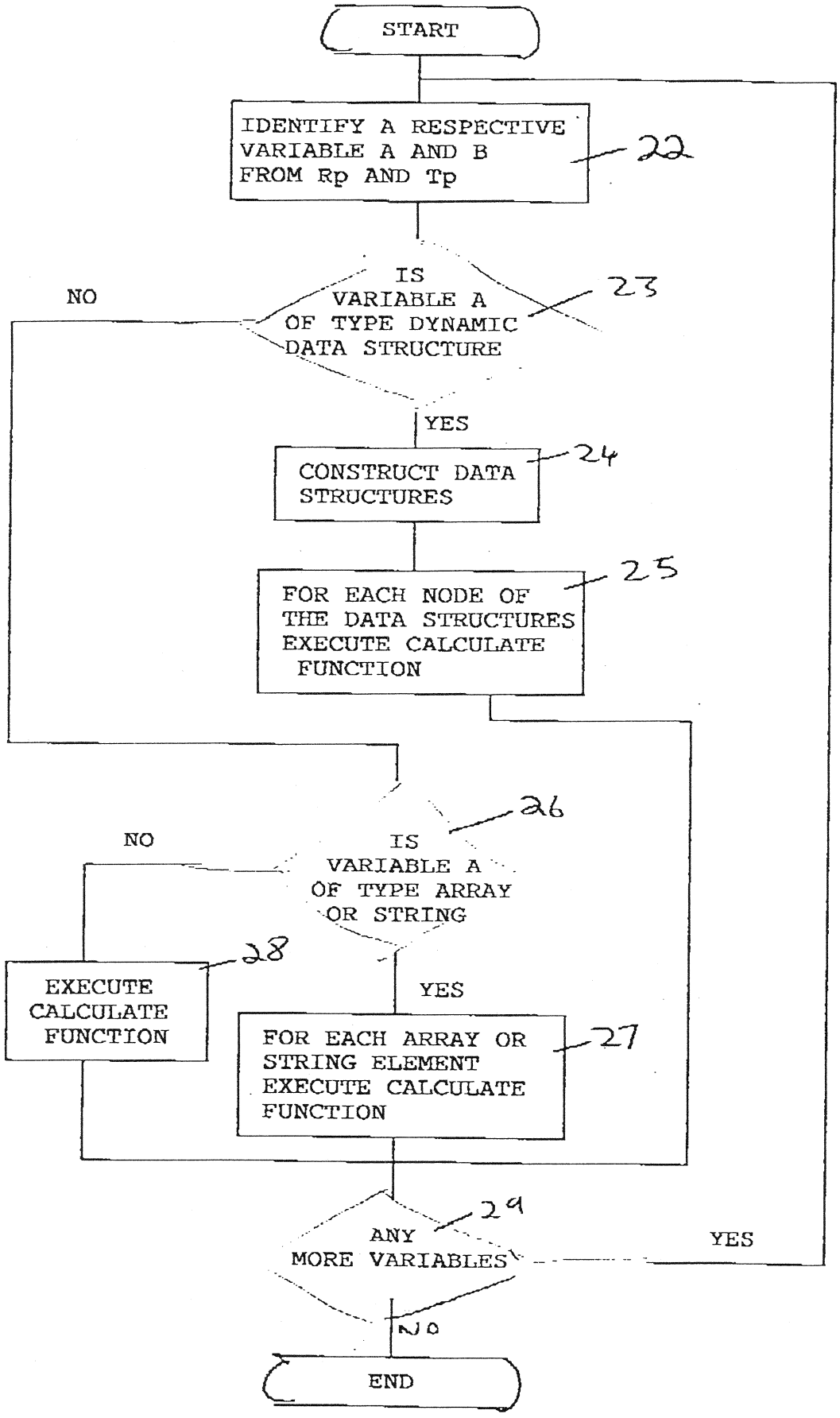

START
IDENTIFY A RESPECTIVE VARIABLE A AND B FROM Rp AND Tp
22
IS VARIABLE A OF TYPE DYNAMIC DATA STRUCTURE
23
NO
YES
CONSTRUCT DATA STRUCTURES
24
FOR EACH NODE OF THE DATA STRUCTURES EXECUTE CALCULATE FUNCTION
25
IS VARIABLE A OF TYPE ARRAY OR STRING
26
NO
YES
EXECUTE CALCULATE FUNCTION
28
FOR EACH ARRAY OR STRING ELEMENT EXECUTE CALCULATE FUNCTION
27
ANY MORE VARIABLES
29
YES
NO
END

FIG 3

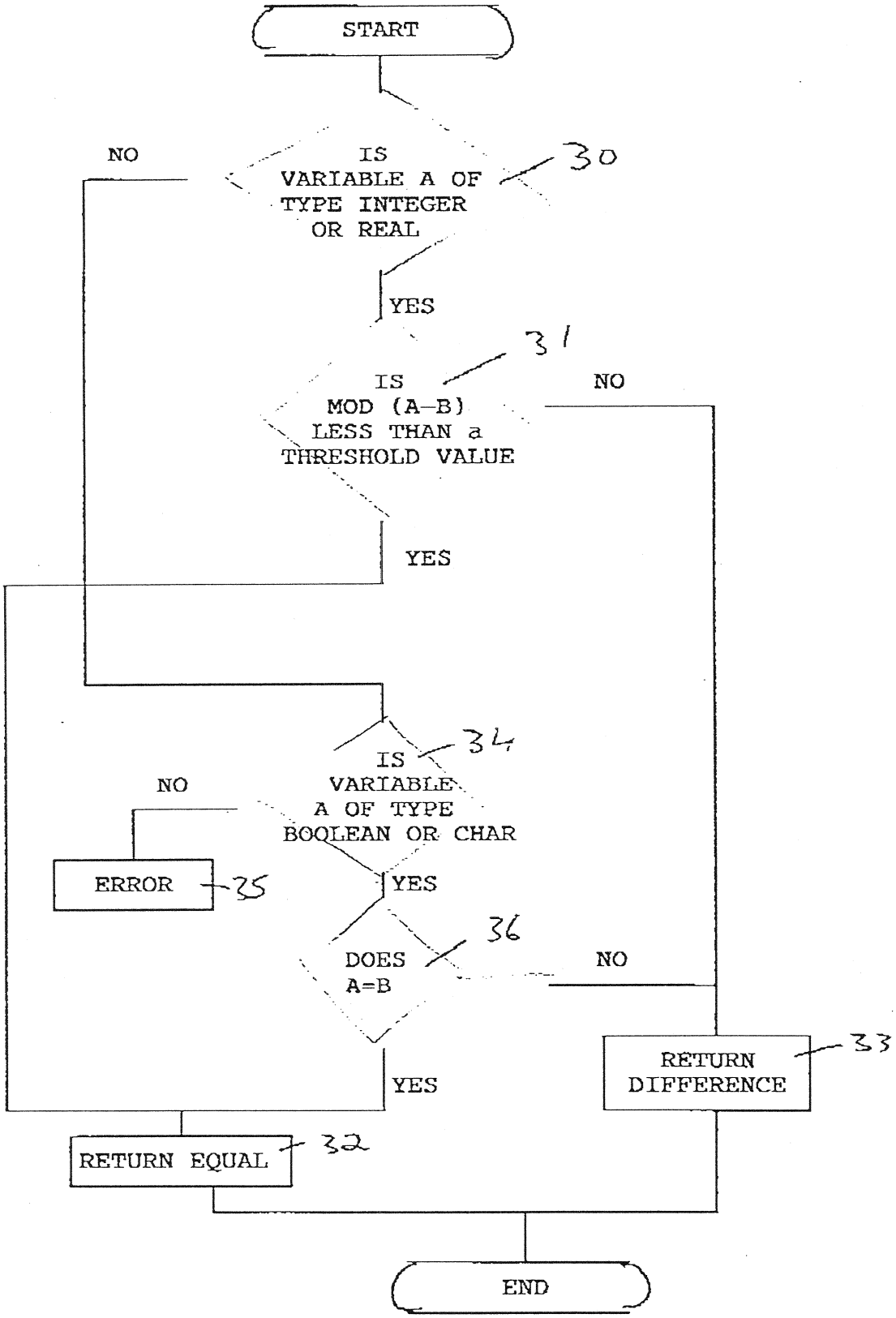

START
IS VARIABLE A OF TYPE INTEGER OR REAL
30
NO
YES
IS MOD (A-B) LESS THAN a THRESHOLD VALUE
31
NO
YES
IS VARIABLE A OF TYPE BOOLEAN OR CHAR
34
NO
ERROR
35
YES
DOES A=B
36
NO
YES
RETURN DIFFERENCE
33
RETURN EQUAL
32
END

FIG 4

```
test:= equal,difference;

function simple_types_equal(a,b,threshold_value) : test

begin

        if mod(a-b) < threshold-value then simple_types_equal := equal else

                simple_types_equal := difference;

end;

function one_dim_arrays_equal(a,b,threshold_value): boolean;

begin

        same := true;

        for i :=        max(low_bound(a),low_bound(b)) to
                        min(high_bound(a),high_bound(b)) do begin

                same := same and simple_types_equal(a[i],b[i],threshold_value)

        end;

        one_dim_arrays_equal := same;

end;

function multi_dim_arrays_equal(a,b,threshold_value): boolean;

begin

        same := true;

        for i :=        max(low_bound(a,1),low_bound(b,1)) to
                        min(high_bound(a,1),high_bound(b,1)) do

                for j :=        max(low_bound(a,2),low_bound(b,2)) to
                                min(high_bound(a,2),high_bound(b,2)) do

                        for k :=        max(low_bound(a,3),low_bound(b,3)) to
                                        min(high_bound(a,3),high_bound(b,3)) do

                                .   .   .   .

                                    .   .   .   .

                        same := same and simple_types_equal(a[i,j,k,...],

                        b[i, j,k,...],threshold_value)

        multi_dim_arrays_equal := same;

end;
```

FIG 5