# Program Testing by Demonstrating Consistency with Specifications

D. J. Richardson

COINS Technical Report 79-02
February 28, 1979

Department of Computer and Information Science
University of Massachusetts, Amherst
Amherst, Massachusetts 01003

This work was supported by the National Science Foundation under grant # NSFMCS 77-02101 and the U.S. Air Force under grant # AFOSR 77-3287.

## Abstract

An approach to program testing by demonstrating the consistency between procedures and specifications is described. Operational specifications, which describe the output of a function by transformations on the input values, are considered as test oracles. Three properties of consistency between a procedure and an operational specification - compatibility, isomorphism, and equivalence - are proposed. General approaches to demonstrate each of the three consistency properties are outlined, and the proposed method is illustrated with a decision table specification. A test data selection strategy based on both the procedure and the specification is presented by which consistency can be reassured through actual execution on the test set chosen. By demonstrating the consistency between a procedure and a specification, and ascertaining the consistency through actual execution, the reliability of the procedure is established.

# I. Introduction

In testing a program, the ultimate goal is establishing the correctness of the program. The usual testing process entails the execution of the program on a set of test data that is considered representative of the program's input domain. It is well-known, however, that testing is capable of detecting errors in a program, but not in general of establishing their absence (exhaustive testing is an exception to this disclaimer). Various alternative strategies for attempting to demonstrate program correctness have been proposed.

The program verification approach [14] of proving the correctness of a program is one such alternative. In general, this approach employs appropriate axioms and rules of inference to develop a rigorous mathematical proof stating that the program correctly transforms input values described by initial assertions into output values described by final assertions. Provided the assertions precisely characterize the intended purpose of the program, a complete correct proof (including proof of termination) verifies the correctness of the program. At the current time, however, program verification is extremely complex and expensive. In addition, proofs of correctness are limited to conclusions about program behavior in a postulated environment, whereas program testing has the advantage of obtaining well-founded information about a program in its natural environment.

With this is mind, several attempts have been made to aid in the selection of test data sets [2,3,9,11,21] that increase assurance in the program's reliability as a result of correct test runs. In the absence of the expedient ability to verify the correctness of a program, attesting to the reliability of the program is a practical

goal for program testing.

In order for testing to be effective in demonstrating the reliability of a program, there must be some mechanism, or test oracle, by which the correct processing of test data can be recognized. There are many and various guises of test oracles, ranging from input-output pairs residing in the head of the programmer to a correct program that computes the intended function. A specification that formally describes the task that the program is supposed to perform, can also serve as the necessary test oracle.

When a specification providing a program's intended output is available, the testing process may be done by comparing the program's computations with those of the specification. In this paper, program testing is considered as a process that attempts to demonstrate the consistency between a program and an associated specification. Three consistency properties are proposed - compatibility, isomorphism, and equivalence. By demonstrating the consistency, or lack thereof, between the procedures and their specifications, the reliability of a program is exhibited. As a practical application, the use of decision tables as an approach to program specification is emphasized. To illustrate the testing method proposed, the consistency between a procedure and a decision table is partially demonstrated. In addition, a test data selection strategy is proposed by which consistency can be reassured through the actual execution of the program on the test data set chosen. Demonstrating the consistency of a program with its specification, and insuring the run-time consistency, establishes the reliability of the program.

## II. Functional Specifications and Decision Tables

Throughout the design of a program, the specification becomes a more elaborate and refined description of the proposed program. The program specification usually follows a progression from a requirements specification, an informal description of the desired program behavior, to a group of procedure specifications, an identification of the program modules and a formal description of their intended behavior. A procedure specification usually dictates performance constraints that must be realized by an implementation of the procedure as well as a description of the task that the procedure is to perform.

This description of the intended task of a procedure is the functional specification. To enable meaningful analysis, the function of the proposed program must be described in a formal specification language, one whose syntax and semantics are precisely defined. There are basically two techniques for formal functional specification - input-output specification and operational specification [13]. By either technique, the specification must be complete and unambiguous (one and only one output value is specified for each input value) in order to correctly describe the intended function as a mapping from the inputs to the outputs. In addition to the intended input-output relationships, an operational specification provides more extensive properties of the function with which the procedure may be compared.

By the technique of input-output specification, the relationships between the input values and the output values are described by pairs of assertions. Whenever the input values satisfy an initial assertion, a correct implementation of the procedure will compute output values satisfying the corresponding final assertion. The

technique of input-output specification has been used extensively in the inductive assertion method of program verification [8].

An operational specification differs from input-output specification in that transformations on the input values are described explicitly by indicating actions that serve to provide the desired output values. Operational specifications may take on several forms - input domains with corresponding output computations, decision tables, program design, and correct procedures, to name only a few. In the approach to program testing presented here, the operational type of specification is considered because it provides a more informed test oracle. Many of the ideas, however, are applicable to input-output specifications as well. Furthermore, the applicability of decision tables is examined in more depth since they are better defined and more restrictive than the general operational specification.

Decision tables provide a practical method for specifying an extensive, though restricted, class of functions and procedures. The American National Standards Institute defines a decision table as "a table of all contingencies that are to be considered in the description of a problem, together with the actions to be taken" [15]. Hence, as an operational specification for a procedure, a decision table provides the intended input-output relationships by displaying precisely all conditions that must be satisfied before the indicated actions are to be performed.

The basic structure of a decision table [12, 15, 17] is illustrated in figure 1. The table header identifies the decision table with a name, and may identify the inputs and outputs, as well. The remainder of the table is divided into four quadrants. The condition stub in the upper left quadrant contains all conditions that have bearing on

the decisions in the function being specified. The lower left quadrant is the action stub, which prescribes explicitly the actions that may be applied in obtaining the output specified for the procedure. The condition entries in the upper right specify the relevant outcomes of the combination of conditions, while the action entries in the lower right dictate the actions to be applied under each specified combination.

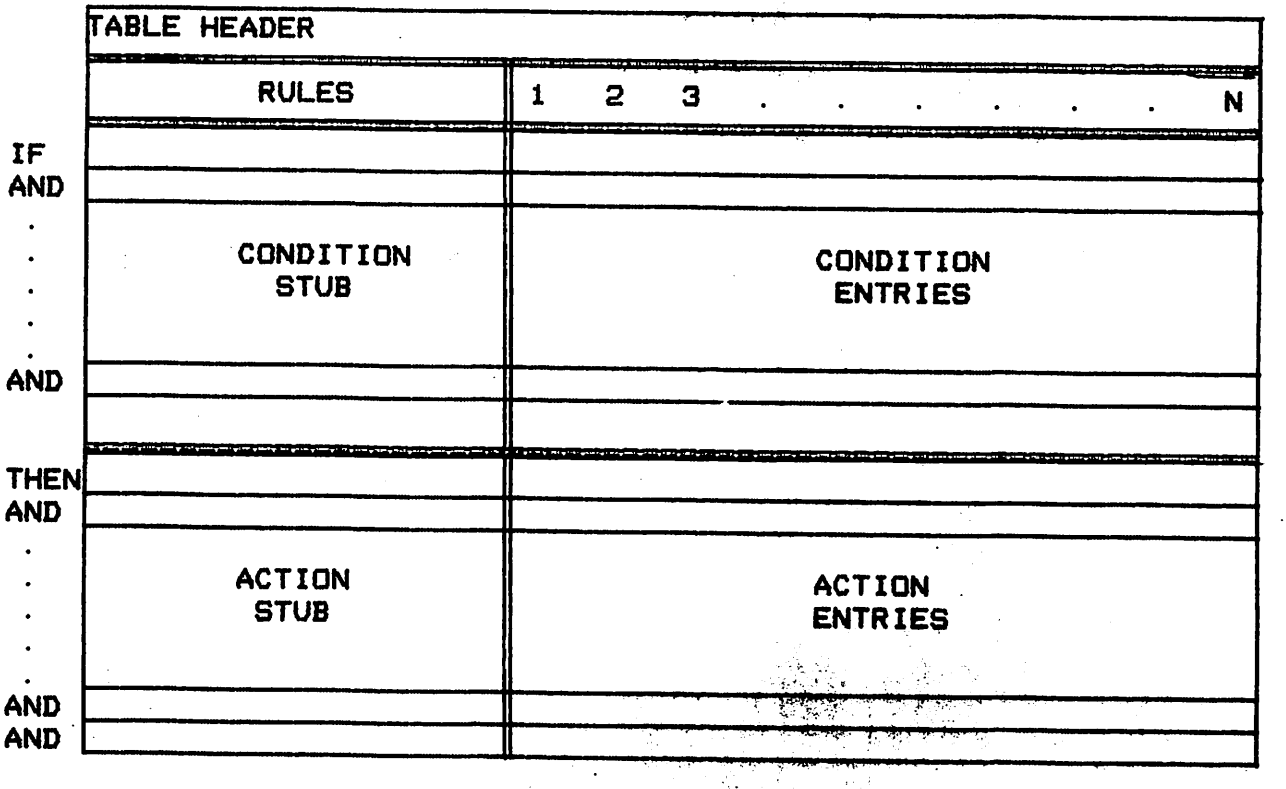


Figure 1. Decision Table Structure

There are several types of decision tables in general use, but each variation can be translated into the limited-entry form. This form restricts condition entries to three possible responses: "Y" indicates the condition must be true; "N" indicates the condition must be false; "-" (dash) indicates the condition is irrelevant. Similarly, an action entry may be one of two responses: "X" indicates the action is to be performed; " " (blank) indicates the action is not to be performed. A column in the condition entries portion of a

decision table corresponds to a boolean interpretation of the conditions in the condition stub; this interpretation specifies a particular combination of conditions. A column in the action entries portion specifies an ordered sequence of actions to be performed under certain conditions; this sequence is determined by the order in which the actions are listed in the action stub.

The association between condition entries and action entries in a decision table is called a rule and is represented by a column in the table. The decision table rules are numbered for identification purposes. A rule may be interpreted as an if-then construct, where the condition entries serve as the if portion and the action entries serve as the then portion of the rule. Each such decision table rule, therefore, describes a relationship between input values satisfying the specified interpretation of the conditions and output values produced by performing the indicated sequence of actions.

Any type of operational specification can be thought of in terms of rules describing the input-output relationships. An operational specification $R$, therefore, can be represented by a set of rules, $\{R_1, R_2, \ldots, R_N: \ 1 \leq N \leq \infty\}$. The general form for an operational specification must allow an infinite number of rules since some specification languages may allow a notation for indefinite iteration. For each rule $R_J$, the rule domain $D_J^R$ is the set of input values for which the rule is applicable and the rule computation $C_J^R$ is the transformation obtained by applying the actions indicated by the rule.

A procedure $P$ defines a set of paths, which are similar to the rules in an operational specification. A path through a procedure corresponds to some possible flow of control. In general, a procedure may define an infinite number of paths, since flows of control that differ in the number of iterations of loops are considered to be

different paths. Hence, $P$ defines the set $\{P_1, P_2, \ldots, P_M: \ 1 \leq M \leq \infty\}$. Associated with each path $P_I$ is the path domain $D_I^P$, which is the subset of the input domain that causes execution of the path, and the path computation $C_I^P$, which is the function that is computed by the sequence of executable statements along the path.

Symbolic execution [4, 5] assigns symbolic names to a program's input values and "executes" a program path. This method of evaluation provides representations for the path domain and the path computation. A similar procedure can be used to "apply" a rule to symbolic inputs, and thus construct representations of the rule domain and the rule computation. A path or rule domain is represented by a system of constraints on the input values $x$. The condition formed by these constraints defines the subset of the input domain for which the path or rule is applicable. This system of constraints can be translated into some canonical form, such as a simplified, conjunctive normal form. A path or rule computation is represented by a vector of symbolic formulas for the output values $z$. Each formula is a symbolic expression in terms of the symbolic names assigned to the input values $x$. These expressions may be converted to a canonical form, such as a simplified, ordered expression [19]. The canonical forms for the representations of the path and rule domains and computations exemplifies the similarity between the input-output relationships provided by paths and rules. Each provides a computation, which produces output values, that is applicable for a subset of the input values. In the next section, the comparison of a procedure and an operational specification by the examination of the paths and rules is explored.

## III. Consistency with Specifications

In comparing a program with a specification, the goal is to determine how consistent each procedure is in relation to the associated operational specification. Consistency should imply that the procedure is, to some extent, reliable. With this comparison in mind, three consistency properties are proposed - compatibility, isomorphism, and equivalence - which differ in the manner in which the procedure must conform to an operational specification. These properties are defined in terms of the general operational specification as presented in the previous section. An approach to demonstrating the consistency between a procedure and an operational specification is outlined in the next section.

The most basic form of consistency is the compatibility of a procedure and an operational specification. Assuming that all inputs and outputs are considered logically independent, compatibility must hold in order for the procedure to be a correct implementation of the specification.

> Definition: A procedure $P$ is compatible to an operational specification $R$ if $P$ and $R$ have the same input vector $x$, the same output vector $z$, and the same input domain $X$.

Compatibility states that the procedure and the specification have the same interface - that is, they have the same number and type of inputs and outputs - and the inputs are restricted to values from the same domain. The restriction on the domain must hold for the specification to be complete since the specification must describe an output for each input for which the procedure might be executed. In the trivial case, this input domain is the entire set of values for the type of the input, but some specification and programming languages allow

assumptions to further restrict the domain of input values. The compatibility property is a reasonable restriction on the class of associated procedures and operational specifications for further consideration. The other properties of consistency are defined under the assumption that compatibility holds.

The prevalence of compatibility does not imply that the procedure is correct. In order to realize the function described by an operational specification, the procedure must not only accept and produce values of the same type, it must also compute the output values specified for each vector in the input domain. A procedure $P$ is a correct implementation of an operational specification $R$ if for all $x \epsilon X$, $P(x) = R(x)$, where $P(x)$ is the output vector resulting from execution of $P$ on $x$, and $R(x)$ is the output vector specified by $R$ for $x$. For any input vector $x \epsilon X$, a particular path in the procedure, say $P_I$, is executed; thus $x \epsilon D_I^P$, and $P(x) = C_I^P(x)$. Similarly, a particular rule in the operational specification, say $R_J$, is applicable; thus $x \epsilon D_J^R$, and $R(x) = C_J^R(x)$. For this input vector, the procedure and the specification produce the same output values, $P(x) = R(x)$, if and only if the appropriate path and rule computations agree, $C_I^P(x) = C_J^R(x)$.

During execution, a path computation is carried out over the elements in the corresponding path domain only. Likewise, a rule computation is applied only to those elements in the rule domain. Due to this similarity, the equivalence of a path and a rule can be considered.

Definition: A path $P_I$ is equivalent to a rule $R_J$ if $D_I^P = D_J^R$ and for all $x \epsilon D_I^P$ $(= D_J^R)$, $C_I^P(x) = C_J^R(x)$.

Thus, a path and a rule are equivalent if the path domain and the rule domain are equal and for each element of that domain, the path

computation and the rule computation produce equal output values. If a one-to-one correspondince between the paths in a procedure and the rules in an operational specification can be found based on this path-rule equivalence, an isomorphism can be defined between the procedure and the operational specification.

Definition: A procedure $P$ is isomorphic to an operational specification $R$ if there exists a bijective mapping $I : P \rightarrow R$ such that if $I(P_K) = R_K$, then $D_K^P = D_K^R$ and for all $x \in D_K^P (= D_K^R)$, $C_K^P(x) = C_K^R(x)$.

If the operational specification correctly describes the desired function, isomorphism is sufficient, but not necessary, for the procedure to be correct. In addition, isomorphism gives evidence that the internal structure of the procedure and the specification are similar.

The close conformity of an isomorphism might be required when the specification is a high-level design that is to be used as a guideline for implementation of the procedure. On the other hand, the requirement may be that the procedure realize the intended function without regard to the method of implementation described by a specification for the function. This relaxed conformity might be desired when the operational specification is a design written for simplicity, but the procedure ought to be coded for efficiency.

In order for the procedure to realize the specification, the output generated for each element of the input domain must equal that indicated by the operational specification, although the strict one-to-one correspondence of an isomorphism is not necessary. A path computation and a rule computation must produce equal output values for any inputs to which they jointly apply, but the restriction that the two computations be defined over the same domain may be set aside.

When two computations are not defined over the same domain, they may be considered equivalent over a particular domain - namely, the joint domain over which they are defined - if they compute equal values for each of those elements. The intersection of a path domain and a rule domain is the joint domain over which the corresponding path and rule computations are applicable. A path computation and a rule computation can be considered equivalent over their joint domain if they compute the same output for each element of that domain. Note that when the path domain and the rule domain are equal, this computation equivalence implies that the path and the rule are equivalent.

> Definition: A path computation $C_I^P$ and a rule computation $C_J^R$ are equivalent over their joint domain $D_I^P \cap D_J^R$ if for all $x \in D_I^P \cap D_J^R$, $C_I^P(x) = C_J^R(x)$.

This definition gives rise to a definition of the equivalence of a procedure and an operational specification, which is in terms of the equivalence of the computations over joint domains for each path-rule pair.

> Definition: A procedure $P$ is equivalent to an operational specification $R$ if for all path-rule pairs $(P_I, R_J)$, $1 \leq I \leq M$ and $1 \leq J \leq N$, for all $x \in D_I^P \cap D_J^R$, $C_I^P(x) = C_J^R(x)$.

Since both the procedure and the specification are complete, the input domain $X$ can be represented as the union of the intersections of all path domains and all rule domains, $X = \bigcup [D_I^P \cap D_J^R$, for $1 \leq I \leq M$ and $1 \leq J \leq N]$. Thus the prevalence of computation equivalence over the joint domains for each path-rule pair means the procedure and the operational specification produce the same output for each input in the domain.

Equivalence is certainly the most important of the proposed consistency properties since under the assumptions that compatibility holds and the specification is correct, the procedure is correct if and only if it is equivalent to the associated operational specification. Equivalence is applicable to all classes of related procedures and operational specifications, whereas isomorphism holds for a limited class. As a restricted concept of equivalence, however, it is easier to determine whether isomorphism prevails.

These three properties of consistency allow the attachment of differing requisites on the conformity of a procedure and an operational specification. Compatibility implies that the procedure conforms to the specified interface. The prevalence of either isomorphism or equivalence implies that the procedure is correct, provided the specification precisely describes the intended function. Moreover, isomorphism implies that the procedure realizes the function in much the same manner as the operational specification describes that function.

# IV. Demonstration of Consistency

Demonstrating consistency between a procedure and an associated operational specification substantiates the reliability of the procedure. Complete consistency - that is, isomorphism or equivalence - implies that the procedure realizes the function described by its specification. Even when complete consistency cannot be shown for the entire procedure, determination of partial consistency provides some confidence in its reliability. The more complete the consistency between the procedure and its operational specification, the more meaningful this confidence in the reliability of the procedure. A general approach to demonstrating the consistency between a procedure and an operational specification is outlined in this section. A more detailed explanation of the steps involved is given in [18]. This approach is then illustrated for a decision table specification and a corresponding FORTRAN procedure. Though in theory the consistency properties apply to all forms of operational specification, practical methods for demonstrating consistency for the general form remain to be explored more fully.

The compatibility of a procedure and an operational specification is fairly easy to determine provided the specification and the programming languages have constructs for declaring parameters and variable types. By comparing such declarations, it is possible to determine if the procedure and the operational specification have the same number and type of inputs and outputs. Data flow analysis methods [16] may be required in order to determine that each parameter has the same input/output class in the procedure as is described in the operational specification. In addition, the specification and programming languages might allow assumptions on the inputs, thus

constraining the values the inputs may assume. These assumptions may also be compared in order to determine the equivalence of the domain of input values. If the input vector and its domain and the output vector agree, then the procedure is compatible to the operational specification.

When compatibility holds, the demonstration of additional consistency can continue with a comparison of the path domains with the rule domains and the path computations with the rule computations. Without further restriction, complete consistency can only be demonstrated when there are a finite number of paths and rules. In the case where finiteness is not met, partial consistency of a subset of the paths and rules may be considered. Another approach involves the specification of loops by recurrence relations. In the symbolic execution of the procedure, similar recurrence relations could be derived [1], and these relations might then be compared. In the discussion that follows, though, the class of procedures and operational specifications under consideration is limited to those with a finite number of paths and rules. When this restriction holds, symbolic execution of all the paths in the procedure and all the rules in the operational specification can be performed in order to generate representations for the path and rule domains and computations. Since both the procedure and the specification are unambiguous, the path domains are mutually disjoint as are the rule domains. No restriction, however, has been made on the computations; neither the path computations nor the rule computations must be distinct. With this in mind, the comparison of a procedure and an operational specification should first be based on the relationships between the path domains and the rule domains; after which the corresponding path computation and rule computation must be compared. There are two

problems inherent in demonstrating either equivalence or isomorphism: showing domain emptiness and showing computation equivalence. These problems will be treated after the general processes for demonstrating isomorphism and equivalence are presented.

The property of isomorphism holds if there is a one-to-one correspondence between the paths in the procedure and equivalent rules in the operational specification. Isomorphism is not even considered unless there are the same number of paths and rules. A correspondence between a path $P_K$ and a rule $R_K$ is made if the path domain $D_K^P$ and the rule domain $D_K^R$ are equal. When the demonstration of this equality cannot be acheived by a term-by-term comparison of the constraints in the symbolic representations of the domains, this equality can be demonstrated by showing that $D_K^P \cap \sim D_K^R$ is empty (where $\sim D_K^R$ is the complement of $D_K^R$). After determining domain equality, the equivalence of the path computation $C_K^P$ and the rule computation $C_K^R$ over this domain must be determined. If a one-to-one correspondence can be constructed in this way, then the procedure is isomorphic to the operational specification.

A procedure is equivalent to an operational specification if for each path-rule pair, the path computation and the rule computation are equivalent over the subset of the input domain formed by the intersection of the path domain and the rule domain. For each path-rule pair $(P_I, R_J)$, the joint domain $D_I^P \cap D_J^R$ over which this path-rule pair applies can be constructed by conjoining the representations of the path domain and rule domain. If the joint domain is empty, then the path-rule pair need not be considered further, since they are not both mututally satisfiable - no data element exists that causes execution of the path and for which the rule is applicable. The computations are trivially equivalent in this

case. For a non-empty intersection, the corresponding path and rule computation must be compared for equivalence , $C_I^P = C_J^R$, over this joint domain. If the computations are equivalent over the joint domains for each path-rule pair, then the procedure is equivalent to the operational specification.

One of the problems that must be approached in demonstrating euivalence or isomorphism between a procedure and an operational specification is the determination of domain emptiness. This involves demonstrating the satisfiability or unsatisfiability of the condition defining that domain. One approach to this problem is the axiomatic approach, which uses first order predicate calculus to prove whether or not the conjunction defining the domain is satisfiable. This method is subject to the limitations of automatic theorem proving [7]. Another approach is the algebraic approach, which attempts to find a solution to the constraints defining the domain. If the set of constraints is unsatisfiable, the domain is empty. Several algebraic techniques, such as linear programming or a gradient hill climbing algorithm, can be used to solve the system of constraints. No method, however, can solve any arbitrary system of constraints [6], and determining the emptiness of a domain is undecidable.

Another problem that must be addressed is the determination of computation equivalence over a particular domain. In some cases, this decision is trivial, but in general, the equivalence of computations is undecidable. The first approximation to determining whether a path computation $C_I^P$ and a rule computation $C_J^R$ are equivalent is also done by comparing their canonical representations. When a term-by-term comparison of the symbolic representations reveals that the two computations are symbolically identical, they are equivalent over any domain. Otherwise, the two compuations are equivalent over the domain

in question if the symbolic difference between their canonical representations, $C_I^P - C_J^R$, is zero for all elements of that domain since this implies that the two computations are equal for all elements of that domain. The most straight-forward method for determining whether this holds is by finding the solutions of the equation $C_I^P - C_J^R = 0$ using a mathematical package for finding the zeroes of a function. If the condition defining the domain restricts the inputs to values in this solution set, then the symbolic difference is zero over the domain. Several other approaches to deciding on computation equivalence are proposed in [18].

These processes for demonstrating consistency by comparing a procedure to the associated operational specification can be readily applied to decision tables. A simple example of a function that is amenable to operational specification by a decision table is the calculation of Capital Gains and Losses for 1978. This is done by filing Schedule D of the Federal Income Tax Form 1040. Part III of that form, which is the only portion under consideration here, is shown in figure 2. By reading this part of the form, one can see that there are four entries, which are calculated in earlier parts of this form - namely lines 3 and 5 from part I and lines 11 and 13 from part II - that constitute the input to part III. The interrelationships between these inputs is logically complex enough to make this an interesting example for specification by a decision table. The decision table CGLDT, shown in figure 3, is an operational specification for the calculation performed in part III of schedule D. A FORTRAN function CGL that attempts to calculate Capital Gains or Losses appears in figure 4; the procedure is annotated with statement numbers that will be used later in describing the paths. This FORTRAN function and the decision table corresponding to it will be used as an

example of demonstrating consistency between a procedure and an operational specification.

**Part III** **Computation of Capital Gain Deduction**
*(Complete this part only if line 14 shows a gain)*

| | | | |
|---|---|---|---|
| 14 | Combine lines 5 and 13, column (f), and enter here. If result is zero or a loss, do not complete the rest of this part. Instead skip to Part IV, line 24 on page 2 | 14 | |
| 15 | Enter line 13, column (f) or line 14, whichever is smaller. If zero or a loss, enter zero and skip to line 23 | 15 | |
| 16 | If line 11, column (g) is a gain, combine lines 3 and 11, column (g), and enter here. If this line or line 11, column (g) shows a loss or zero, enter a zero and skip to line 20 | 16 | |
| 17 | Enter line 11, column (g) or line 16, whichever is smaller | 17 | |
| 18 | Enter line 15 or line 17, whichever is smaller | 18 | |
| 19 | Enter 60% of amount on line 18 | 19 | |
| 20 | Subtract line 18 from line 15 | 20 | |
| 21 | Enter 50% of amount on line 20 | 21 | |
| 22 | Add line 19 and line 21. This is your capital gain deduction | 22 | |
| 23 | Subtract line 22 from line 14. Enter this amount on Form 1040, line 14 | 23 | |

Figure 2. Capital Gains and Losses Form, Part III

CGLDT INPUT: REAL L3, L5, L11, L13
OUTPUT: REAL CGL

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1: | L5+L13>0.0 | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| C2: | L13<L5+L13 | – | N | N | N | N | N | Y | N | Y | Y | Y | Y | Y | Y | Y |
| C3: | L13≤0.0 | – | – | – | – | – | – | – | – | N | N | N | N | N | N | Y |
| C4: | L5+L13≤0.0 | – | N | N | N | N | N | N | Y | – | – | – | – | – | – | – |
| C5: | L11>0.0 | – | N | Y | Y | Y | Y | Y | – | N | Y | Y | Y | Y | Y | – |
| C6: | L3+L11≤0.0 | – | – | N | N | N | N | Y | – | – | N | N | N | N | Y | – |
| C7: | L11<L3+L11 | – | – | N | N | Y | Y | – | – | – | N | N | Y | Y | – | – |
| C8: | L13<L11 | – | – | – | – | – | – | – | – | – | – | – | N | Y | – | – |
| C9: | L13<L3+L11 | – | – | – | – | – | – | – | – | – | N | Y | – | – | – | – |
| C10: | L5+L13<L11 | – | – | – | – | N | Y | – | – | – | – | – | – | – | – | – |
| C11: | L5+L13<L3+L11 | – | – | N | Y | – | – | – | – | – | – | – | – | – | – | – |
| A1: | L14=L5+L13 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| A2: | L15=L13 | | | | | | | | | X | X | X | X | X | X | X |
| A3: | L15=L5+L13 | | X | X | X | X | X | X | X | | | | | | | |
| A4: | L15=0.0 | | | | | | | | X | | | | | | | X |
| A5: | L16=L3+L11 | | | X | X | X | X | X | | | X | X | X | X | X | |
| A6: | L16=0.0 | | X | | | | | X | | X | | | | | X | |
| A7: | L17=L11 | | | | | X | X | | | | | | X | X | | |
| A8: | L17=L16 | | | X | X | | | | | | X | X | | | | |
| A9: | L18=L15 | | | | X | | X | | | | | X | | X | | |
| A10: | L18=L17 | | | X | | X | | | | | X | | X | | | |
| A11: | L19=.6*L18 | | | X | X | X | X | | | | X | X | X | X | | |
| A12: | L20=L15–L18 | | X | X | X | X | X | X | | X | X | X | X | X | X | |
| A13: | L21=.5*L20 | | X | X | X | X | X | X | | X | X | X | X | X | X | |
| A14: | L22=L19+L21 | | X | X | X | X | X | X | | X | X | X | X | X | X | |
| A15: | CGL=L14–L22 | X | X | X | X | X | X | X | v | X | X | X | X | X | X | |

Figure 3. Decision Table CGLDT

```
      FUNCTION CGL(L3,L5,L11,L13)
C
C     CGL returns the Capital Gain or Loss
C     by completing Part III of Form 1040, Schedule D
C     (Capital Gains and Losses)
C     Input Variables:
C            L3  - Sched D, Part I, Line 3
C                  Short-term Gain or Loss for 1978
C            L5  - Sched D, Part I, Line 5
C                  Net Short-term Gain or Loss
C                  including carryover (years after 1968)
C            L11 - Sched D, Part II, Line 11
C                  Long-term Gain or Loss for 1978
C            L13 - Sched D, Part II, Line 13
C                  Net Long-term Gain or Loss
C                  including carryover (years after 1968)
C     Output Variables:
C            CGL - Capital Gain or Loss
C                    Enter this amount on Form 1040, Line 14
C
      REAL L3, L5, L11, L13
      REAL L14, L15, L16, L17, L18, L19, L20, L21, L22
C
C     Initialize all local variables to zero
      L14 = 0.0
      L15 = 0.0
      L16 = 0.0
      L17 = 0.0
      L18 = 0.0
      L19 = 0.0
      L20 = 0.0
      L21 = 0.0
      L22 = 0.0
      CGL = 0.0
C
C     Line 14:  Combine lines 5 and 13 and enter here.
C               If result is zero or a loss,
C               do not complete the rest of this form.
C               Instead, skip to Part IV, Line 24.
 1    L14 = L5 + L13
      IF (L14.GT.0.0) THEN
C
C         Line 15:  Enter line 13 or line 14,
C                   whichever is smaller.
C                   If result is zero or a loss,
C                   enter zero and skip to line 23.
          IF (L13.LT.L14) THEN
 2            L15 = L13
          ELSE
 3            L15 = L14
          ENDIF
          IF (L15.LE.0.0) THEN
 4            L15 = 0.0
          ELSE
C
C             Line 16:  If line 11 is a gain,
C                       combine lines 3 and 11 and enter here.
C                       If this line or line 11 shows a loss or zero,
C                       enter a zero and skip to line 20.
```

```
                IF (L11.GT.0.0) THEN
 5                  L16 = L3 + L11
                    IF (L16.LE.0.0) THEN
 6                      L15 = 0.0
                    ELSE
C
C
C                       Line 17:  Enter line 11 or line 16,
                                  whichever is smaller.
                        IF (L11.LE.L16) THEN
 7                          L17 = L11
                        ELSE
 8                          L17 = L16
                        ENDIF
C
C
C                       Line 18:  Enter line 15 or line 17,
                                  whichever is smaller.
                        IF (L15.LE.L17) THEN
 9                          L18 = L15
                        ELSE
10                          L18 = L17
                        ENDIF
C
C
11                      Line 19:  Enter 60% of line 18.
                        L19 = .6 * L18
                    ENDIF
                ENDIF
C
C               Line 20:  Subtract line 18 from line 15.
12              L20 = L15 - L18
C
C               Line 21:  Enter 50% of line 20.
13              L21 = .5 * L20
C
C               Line 22:  Add line 19 and line 21.
C                         This is your Capital Gain Deduction.
14              L22 = L19 + L21
            ENDIF
C
C           Line 23:  Subtract line 21 from line 14.
C                     Enter this amount of Form 1040, Line 14.
15          CGL = L14 - L22
        ENDIF
        RETURN
        END
```

Figure 4. FORTRAN Function CGL

In order to determine the compatibility of the procedure CGL with the decision table CGLDT, the function header and the declarations of the procedure must be compared with the decision table header. The function header does not indicate precisely which parameters are input and which are output, but the use of data flow analysis would

determine that the procedure CGL has four inputs - L3, L5, L11, and L13 - and one output - CGL. The table header for CGLDT describes the same inputs and outputs. These variables are declared real, and neither the procedure nor the decision table have any assumptions on the input values, which would further restrict the input domain. The implicit assumption, then, for both CGL and CGLDT, is that each input may be assigned values from the entire real domain. There are no inconsistencies between the input vector, the output vector, or the input domain, so the procedure CGL is compatible to the decision table CGLDT.

Once compatibility has been demonstrated, concentration on additional consistency - that is, the demonstration of isomorphism or equivalence - can begin by examination of the paths of the procedure and the rules of the decision table. Symbolic execution of the procedure provides the representations of the path domains and computations (and numbering of the paths) of CGL. Representations for the rules of CGLDT (already numbered) may be obtained by a similar method. These representations are given in the appendix. The two processes outlined previously for deciding whether isomorphism or equivalence hold for a general operational specification are combined into one process here (this combination could be done for the general operational specification, as well). This is more efficient, because once a path-rule pair has been determined to be equivalent, neither the path nor the rule need be considered further.

The process of demonstrating consistency between the paths in the procedure CGL and the rules in the decision table CGLDT begins by searching for equivalent path-rule pairs, and discarding those paths and rules from further consideration. Then for the remaining paths and rules, computation equivalence over the joint domain of each

Compatible inputs and outputs:
Inputs:
(L3,L3), (L5,L5), (L11,L11), (L13,L13)
Outputs:
(CGL,CGL)

Incompatible inputs and outputs:
none

Equivalent path-rule pairs:
$(P_1,R_1)$, $(P_2,R_2)$, $(P_8,R_8)$, $(P_9,R_9)$, $(P_{15},R_{15})$

Path-rule pairs with equivalent computations:
Those over empty joint domains:
$(P_3,R_7)$, $(P_3,R_{10})$, $(P_3,R_{11})$, $(P_3,R_{12})$, $(P_3,R_{13})$, $(P_3,R_{14})$,
$(P_4,R_7)$, $(P_4,R_{10})$, $(P_4,R_{11})$, $(P_4,R_{12})$, $(P_4,R_{13})$, $(P_4,R_{14})$,
$(P_5,R_7)$, $(P_5,R_{10})$, $(P_5,R_{11})$, $(P_5,R_{12})$, $(P_5,R_{13})$, $(P_5,R_{14})$,
$(P_6,R_7)$, $(P_6,R_{10})$, $(P_6,R_{11})$, $(P_6,R_{12})$, $(P_6,R_{13})$, $(P_6,R_{14})$,
$(P_7,R_3)$, $(P_7,R_4)$, $(P_7,R_5)$, $(P_7,R_6)$, $(P_7,R_{10})$,
$(P_7,R_{11})$, $(P_7,R_{12})$, $(P_7,R_{13})$, $(P_7,R_{14})$,
$(P_{10},R_3)$, $(P_{10},R_4)$, $(P_{10},R_5)$, $(P_{10},R_6)$, $(P_{10},R_7)$, $(P_{10},R_{14})$,
$(P_{11},R_3)$, $(P_{11},R_4)$, $(P_{11},R_5)$, $(P_{11},R_6)$, $(P_{11},R_7)$, $(P_{11},R_{14})$,
$(P_{12},R_3)$, $(P_{12},R_4)$, $(P_{12},R_5)$, $(P_{12},R_6)$, $(P_{12},R_7)$, $(P_{12},R_{14})$,
$(P_{13},R_3)$, $(P_{13},R_4)$, $(P_{13},R_5)$, $(P_{13},R_6)$, $(P_{13},R_7)$, $(P_{13},R_{14})$,
$(P_{14},R_3)$, $(P_{14},R_4)$, $(P_{14},R_5)$, $(P_{14},R_6)$, $(P_{14},R_7)$,
$(P_{14},R_{10})$, $(P_{14},R_{11})$, $(P_{14},R_{12})$, $(P_{14},R_{13})$
Those over non-empty joint domains:
$(P_3,R_3)$, $(P_3,R_4)$, $(P_3,R_5)$, $(P_3,R_6)$,
$(P_4,R_3)$, $(P_4,R_4)$, $(P_4,R_5)$, $(P_4,R_6)$,
$(P_5,R_3)$, $(P_5,R_4)$, $(P_5,R_5)$, $(P_5,R_6)$,
$(P_6,R_3)$, $(P_6,R_4)$, $(P_6,R_5)$, $(P_6,R_6)$,
$(P_{10},R_{10})$, $(P_{10},R_{11})$, $(P_{10},R_{12})$, $(P_{10},R_{13})$,
$(P_{11},R_{10})$, $(P_{11},R_{11})$, $(P_{11},R_{12})$, $(P_{11},R_{13})$,
$(P_{12},R_{10})$, $(P_{12},R_{11})$, $(P_{12},R_{12})$, $(P_{12},R_{13})$,
$(P_{13},R_{10})$, $(P_{13},R_{11})$, $(P_{13},R_{12})$, $(P_{13},R_{13})$

Inconsistent path-rule pairs:
$(P_7,R_7)$, $(P_{14},R_{14})$

Figure 5. Results from Consistency Demonstration Between the Procedure CGL and the Decision Table CGLDT

path-rule pair must be checked. There may, in fact, be inconsistent path-rule pairs - that is, pairs for which the intersection between the path and rule domains is not empty and the path and rule computations are not equivalent over this domain. When this occurs there is an error in the procedure. The overall results of the examination of the procedure CGL and the decision table CGLDT are shown in figure 5. As an illustration of the comparison which takes place in determining the consistency, or the lack of it, three

particular path-rule pairs are considered below.

In comparing the path $P_9$ (1, 2, 12, 13, 14, 15) with the rule $R_9$ (1, 2, 12, 13, 14, 15) it is found that they are equivalent. The path domain $D_9^P$ and the rule domain $D_9^R$ are represented by the condition (L5+L13>0.0)^(0.0<L5)^~(L13≤0.0)^~(L11>0.0), while the path computation $C_9^P$ and the rule computation $C_9^R$ are represented by the formula CGL = L5+.5*L13. Term-by-term comparisons of both the computations and the domains reveal that they are symbolically identical. The path-rule pair ($P_9$, $R_9$) is, therefore, equivalent.

The procedure CGL is not isomorphic to the decision table CGLDT, since there are paths and rules remaining for comparison after those which belong to equivalent path-rule pairs are eliminated. All remaining paths and rules must be pair-wise considered; one such pair is the path $P_5$ (1, 3, 5, 7, 10, 11, 12, 13, 14, 15) and the rule $R_6$ (1, 3, 5, 7, 9, 11, 12, 13, 14, 15). The joint domain for which both $P_5$ and $R_6$ apply is (in simplified form) $D_5^P \cap D_6^R$ = (L5+L13>0.0)^~(0.0<L5) (L11>0.0)^~(L3+L11≤0.0)^(0.0≤L3)^(L5+L13=L11). The symbolic difference between the path computation and the rule computation is (also in simplified form) $C_5^P - C_6^R$ = .1*L5+.1*L13-.1*L11. Since the joint domain over which the two computations will be performed restricts the input values to those for which L5+L13=L11, it is obvious that this symbolic difference has zero value over the joint domain. Thus, $C_5^P = C_6^R$ over $D_5^P \cap D_6^R$, and the path-rule pair ($P_5$, $R_6$) satisfies computation equivalence over the joint domain.

Another pair to be considered is the path $P_7$ (1, 3, 5, 6, 12, 13, 14, 15) and the rule $R_7$ (1, 3, 5, 6, 12, 13, 14, 15). In this case, the joint domain over which the path and rule computation will be carried out is $D_7^P \cap D_7^R$ = (L5+L13>0.0)^~(0.0<L5)^(L11>0.0)^ (L3+L11≤0.0). The symbolic difference between the two computations is

$C_7^P - C_7^R$ = .5*L5+.5*L13. The only way in which this difference has zero value is when L5+L13=0.0, but this violates one of the constraints defining the joint domain, namely, (L5+L13>0.0). The two computations are, therefore, not equivalent over the joint domain over which they apply, and the path-rule pair ($P_7$,$R_7$) is inconsistent. There is an error, therefore, in the procedure CGL, which is immediately obvious when the path $P_7$ is compared in full to the rule $R_7$ (statement 6 should be L16 = 0.0). The discovery of an inconsistency such as this would direct the examination of the path in order to help correct the procedure.

The procedure CGL is compatible, but not equivalent (nor isomorphic), to the operational specification of the intended task, which is provided by the decision table CGLDT. After the correction mentioned is made, the procedure CGL is equivalent (and partially isomorphic) to the decision table CGLDT and may be considered reliable in calculating Capital Gains and Losses for 1978.

# V. A Test Data Selection Strategy

Demonstrating consistency between a procedure and the associated operational specification provides confidence that the procedure performs the intended task. This method of attesting to program reliability, however, divorces itself from the run-time surroundings by showing consistency in a postulated environment, just as proving the correctness of a program is done in an artificial environment. To remedy this, the demonstration of consistency must be complemented by the actual execution of the procedure. The method of testing a procedure that has already been presented can be extended to include a test data selection strategy.

In demonstrating equivalence, the input domain is partitioned into subsets of input values that are treated the same by the procedure as well as by the operational specification. Each subset contains inputs that cause execution of a particular path, say $P_I$, and for which a particular rule, say $R_J$, is applicable. This subset is the intersection of the path domain and the rule domain, $D_I^P \cap D_J^R$. A test data set can then be constructed by selecting one or more input values from each such subset of the input domain. The appropriate selection of input values from each subset can increase the probability of detecting errors and in some cases even verify the correctness of the path domain [21]. The procedure can then be executed on this test data set, while the operational specification provides the test oracle. This approach makes it possible to check the run-time consistency of each path-rule pair.

This test data selection strategy is similar to that used in path analysis approaches [2, 3, 10, 21], in which a test set is constructed by choosing elements from each path domain. Path analysis testing

strategies, however, are based solely on the structure of the procedure. The test data selection approach presented here is based on both the procedure and the specification, and thus takes into consideration what the procedure is supposed to do in addition to what is actually does. A testing methodology integrating information from the procedure and the specification should be a more effective strategy than one based on a single source of information.

# VI. Conclusion

The use of specifications in the program testing process can greatly enhance the reliability of software. When an operational specification is available, a program can be tested by demonstrating its consistency with the specification. By examining both the specification and the program, a more comprehensive set of test data can be generated than those obtained by analyzing either the program or the specification alone. Program testing is an expensive and laborious process that is usually left incomplete, thus the development of an effective means of assistance in these processes could dramatically reduce the cost of testing and increase assurance in the program.

If program specifications are to contribute effectively to the analysis of programs, more extensive formal specification languages must be developed. Formal techniques for specifying the intended function of a program can provide a concise and well-understood description, which should reduce the difficulty of comparing the specification and the implementation of a program. Decision tables are a widely-accepted technique for specifying the function of a program, and the proposed methods have been shown to be applicable for this type of operational specification. As progress in the development of formal specification languages is made, the practically of a mechanical means of using new types of specifications in program testing must also be evaluated.

The demonstration of consistency between a procedure and a functional specification increases the assurance in the procedure's reliability. This approach, which is a type of unit testing, relies on the independence of the procedures in a program. Multi-module

programs have not yet been examined, and the difficulty of checking all the interactions among numerous procedures in a single program is formidable. To the extent that a program can be considered the sum of its procedures, the proposed method is useful for testing a multi-module program.

The approach presented in this paper is concerned with the analysis of a program in relation to a specification for the intended function. The specifications considered are high-level and might correspond to a program description developed late in the design of a program. If analysis is not performed throughout the development process, there is no assurance that the specification indeed captures the desired behavior of the function. This problem is addressed by current work [20] in the development of tools that support the design and analysis of program descriptions during the earlier stages of development. In order to acheive the goal of producing more reliable software, a complimentary set of software tools for program specification, program design, program verification, and program testing must be integrated.

## References

1. T. E. Cheatham and D. A. Washington, "Program Loop Analysis by Solving First Order Recurrence Relations," Center for Research in Computing Technology Technical Report #13-78, Aiken Computation Laporatory, Harvard University, 1974.

2. L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," IEEE Transactions on Software Engineering, v. 2, n. 3, pp. 215-222, September 1976.

3. L. A. Clarke, "Automatic Test Data Selection Techniques," Infotech State of the Art Report on Software Testing, September 1978.

4. L. A. Clarke and D. J. Richardson, "Symbolic Evaluation Methods for Program Analysis," Department of Computer and Information Science Technical Report #79-01, University of Massachusetts, February 1979.

5. J. A. Darringer and J. C. King, "Applications of Symbolic Execution to Program Testing," Computer, v. 11, n. 4, pp. 51-68, April 1978.

6. M. Davis, "Hilbert's Tenth Problem is Unsolvable," American Mathematical Monthly, v. 80, pp. 233-269, March 1973.

7. B. Elspas, K. N. Levitt, R. J. Waldinger, and A. Waksman, "An Assesment of Techniques for Proving Program Correctness," ACM Computing Surveys, v. 4, n. 2, pp. 97-146, June 1972.

8. R. W. Floyd, "Assigning Meaning to Programs," Proceedings of the American Mathematical Society Symposium on Applied Mathematics, v. 19, pp. 19-32, 1967.

9. J. B. Goodenough and S. L. Gerhart, "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, v. 21, n. 2, pp. 156-173, September 1976.

10. W. E. Howden, "Reliability of the Path Analysis Testing Strategy," IEEE Transactions on Software Engineering, v. 2, n. 3, pp. 208-215, September 1976.

11. W. E. Howden, "Symbolic Testing and the DISSECT Symbolic Evaluation System," IEEE Transactions on Software Engineering, v. 3, n. 4, pp. 266-278, July 1977.

12. P. J. H. King, "Decision Tables," The Computer Journal, v. 10, n. 2, pp. 135-142, August 1967.

13. B. H. Liskov and V. Berzins, "An Appraisal of Program Specifications," Computation Structures Group Memo 141-1, Laboratory for Computer Science, Massachusetts Institute of Technology, 1974.

14. R. L. London, "A View of Program Verification," Proceedings of the International Conference on Reliable Software, pp. 534-545, April 1975.

15. H. McDanial, An Introduction to Decision Logic Tables, A

Petrocelli Book, New York, 1978.

16. L. J. Osterweil and L. D. Fosdick, "DAVE - a Validation Error Detection and Documentation System for FORTRAN Programs," Software - Practice and Experience, v. 6, pp. 473-486, 1976.

17. U. W. Pooch, "Translation of Decision Talbes," ACM Computing Surveys, v. 6, n. 2, pp. 125-151, June 1974.

18. D. J. Richardson, "Theoretical Considerations in Testing Programs by Demonstrating Consistency with Specifications," Digest for the Workshop on Software Testing and Test Documentation, pp. 19-56, December 1978.

19. D. J. Richardson, L. A. Clarke and D. L. Bennett, "SYMPLR, SYmbolic Multivariate Polynomial Linearization and Reduction," Department of Computer and Information Science Technical Report #78-16, University of Massachusetts, July 1978.

20. W. E. Riddle, J. H. Sayler, A. R. Segal, A. M. Stavely and J. C. Wileden, "DREAM - A Software Design Aid System," Proceedings of the Third Jerusalem Conference on Information Technology, August 1978.

21. L. J. White and E. I. Cohen, "A Domain Strategy for Computer Program Testing," Digest for the Workshop on Software Testing and Test Documentation, pp. 335-354, December 1978.

# Appendix

**Paths in the procedure CQL:**
(statements executed on each path,
along with the representations of
path domains and path computations)

$P_1$: $P^{(1)}$

$D_1^P$: ~(L5+L13>0.0)

$C_1^P$: CQL = 0.0

$P_2$: $P^{(1,3,12,13,14,15)}$

$D_2^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧~(L11>0.0)

$C_2^P$: CQL = .5*L5 + .5*L13

$P_3$: $P^{(1,3,5,8,10,11,12,13,14,15)}$

$D_3^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13)≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11≤L3+L11)∧~(L5+L13≤L3+L11)

$C_3^P$: CQL = .5*L5 + .5*L13 - .1*L3 - .1*L11

$P_4$: $P^{(1,3,5,8,9,11,12,13,14,15)}$

$D_4^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11≤L3+L11)∧(L5+L13≤L3+L11),

$C_4^P$: CQL = .4*L5 + .4*L13

$P_5$: $P^{(1,3,5,7,10,11,12,13,14,15)}$

$D_5^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11≤L3+L11)∧~(L5+L13≤L11)

$C_5^P$: CQL = .5*L5 + .5*L13 - .1*L11

$P_6$: $P^{(1,3,5,7,9,11,12,13,14,15)}$

$D_6^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11≤L3+L11)∧(L5+L13≤L11)

$C_6^P$: CQL = .4*L5 + .4*L13

$P_7$: $P^{(1,3,5,6,12,13,14,15)}$

$D_7^P$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
(L3+L11≤0.0)

$C_7^P$: CQL = L5 + L13

$P_8$: $P^{(1,3,4,15)}$ (infeasible path)

$D_8^P$: (L5+L13>0.0)∧~(L3<L5+L13)∧(L5+L13≤0.0)

$C_8^P$: CQL = L5 + L13

$P_9$: $P^{(1,2,12,13,14,15)}$

$D_9^P$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L13≤0.0)∧~(L11>0.0)

$C_9^P$: CQL = L5 + .5*L13

$P_{10}$: $P^{(1,2,5,8,10,11,12,13,14,15)}$

$D_{10}^P$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11≤L3+L11)∧~(L13≤L3+L11)

$C_{10}^P$: CQL = .1*L3 + .1*L11 + .5*L13

$P_{11}$: (1,2,5,8,9,11,12,13,14,15)

$D^P_{11}$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11≤L3+L11)∧(L13≤L3+L11)
$C^P_{11}$: CGL = L5 + .4*L13

$P_{12}$: $_{D^P_{12}}$(1, 2, 5, 7, 10, 11, 12, 13, 14, 15)
$D^P_{12}$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11≤L3+L11)∧~(L13≤L3+L11)
$C^P_{12}$: CGL = L5 − .1*L11 + .5*L13

$P_{13}$: $_{D^P_{13}}$(1, 2, 5, 7, 9, 11, 12, 13, 14, 15)
$D^P_{13}$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11≤L3+L11)∧(L13≤L11)
$C^P_{13}$: CGL = L5 + .4*L13

$P_{14}$: $_{D^P_{14}}$(1, 2, 4, 5, 12, 13, 14, 15)
$D^P_{14}$: (L5+L13>0.0)∧(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
(L3+L11≤0.0)
$C^P_{14}$: CGL = L5 + L13

$P_{15}$: $_{D^P_{15}}$(1, 2, 4, 15)
$D^P_{15}$: (L5+L13>0.0)∧(L13<L5+L13)∧(L13≤0.0)

$C^P_{15}$: CGL = L5 + L13

Rules in the decision table CGLDT:
(actions indicated for each rule,
along with the representations of
rule domains and rule computations)

$R_1$: $_{D^R_1}$(1)
$D^R_1$: ~(L5+L13>0.0)

$C^R_1$: CGL = 0.0

$R_2$: $_{D^R_2}$(1, 3, 12, 13, 14, 15)
$D^R_2$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧~(L11>0.0)

$C^R_2$: CGL = .5*L5 + .5*L13

$R_3$: $_{D^R_3}$(1, 3, 5, 8, 10, 11, 12, 13, 14, 15)
$D^R_3$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13)≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11<L3+L11)∧~(L5+L13<L3+L11)
$C^R_3$: CGL = .5*L5 + .5*L13 − .1*L3 − .1*L11

$R_4$: $_{D^R_4}$(1, 3, 5, 8, 9, 11, 12, 13, 14, 15)
$D^R_4$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧~(L11<L3+L11)∧(L5+L13<L3+L11)
$C^R_4$: CGL = .4*L5 + .4*L13

$R_5$: $_{D^R_5}$(1, 3, 5, 7, 10, 11, 12, 13, 14, 15)
$D^R_5$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11<L3+L11)∧~(L5+L13<L11)
$C^R_5$: CGL = .5*L5 + .5*L13 − .1*L11

$R_6$: $_{D^R_6}$(1, 3, 5, 7, 9, 11, 12, 13, 14, 15)
$D^R_6$: (L5+L13>0.0)∧~(L13<L5+L13)∧~(L5+L13≤0.0)∧(L11>0.0)∧
~(L3+L11≤0.0)∧(L11<L3+L11)∧(L5+L13<L11)
$C^R_6$: CGL = .4*L5 + .4*L13

$R_7$: (1, 3, 5, 6, 12, 13, 14, 15)

$D_7^R$: (L5+L13>0. 0)∧~(L13<L5+L13)∧~(L5+L13≤0. 0)∧(L11>0. 0)∧ (L3+L11≤0. 0)

$C_7^R$: CGL = . 5*L5 + . 5*L13

$R_8$: (1, 3, 4, 15) (infeasible rule)

$D_8^R$: (L5+L13>0. 0)∧~(L3<L5+L13)∧(L5+L13≤0. 0)

$C_8^R$: CGL = L5 + L13

$R_9$: (1, 2, 12, 13, 14, 15)

$D_9^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L13≤0. 0)∧~(L11>0. 0)

$C_9^R$: CGL = L5 + . 5*L13

$R_{10}$: (1, 2, 5, 8, 10, 11, 12, 13, 14, 15)

$D_{10}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L13≤0. 0)∧(L11>0. 0)∧ ~(L3+L11≤0. 0)∧~(L11<L3+L11)∧~(L13<L3+L11)

$C_{10}^R$: CGL = . 1*L3 + . 1*L11 + . 5*L13

$R_{11}$: (1, 2, 5, 8, 9, 11, 12, 13, 14, 15)

$D_{11}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L13≤0. 0)∧(L11>0. 0)∧ ~(L3+L11≤0. 0)∧~(L11<L3+L11)∧(L13<L3+L11)

$C_{11}^R$: CGL = L5 + . 4*L13

$R_{12}$: (1, 2, 5, 7, 10, 11, 12, 13, 14, 15)

$D_{12}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L13≤0. 0)∧(L11>0. 0)∧ ~(L3+L11≤0. 0)∧(L11<L3+L11)∧~(L13<L3+L11)

$C_{12}^R$: CGL = L5 − . 1*L11 + . 5*L13

$R_{13}$: (1, 2, 5, 7, 9, 11, 12, 13, 14, 15)

$D_{13}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L13≤0. 0)∧(L11>0. 0)∧ ~(L3+L11≤0. 0)∧(L11<L3+L11)∧(L13<L11)

$C_{13}^R$: CGL = L5 + . 4*L13

$R_{14}$: (1, 2, 4, 5, 12, 13, 14, 15)

$D_{14}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧~(L5+L13≤0. 0)∧(L11>0. 0)∧ (L3+L11≤0. 0)

$C_{14}^R$: CGL = L5 + . 5*L13

$R_{15}$: (1, 2, 4, 15)

$D_{15}^R$: (L5+L13>0. 0)∧(L13<L5+L13)∧(L13≤0. 0)

$C_{15}^R$: CGL = L5 + L13