Intern. Conf. of the SCCC, 12 - Santiago
16/10/92

Bridging the Gap in the Presence of Infeasible Paths: Potential Uses Testing Criteria *

José C. Maldonado
University of São Paulo (USP)
C. P. 668, 13560 São Carlos, SP, Brazil
e-mail:jcmaldon@icmsc.usp.ansp.br

Marcos L. Chaim
Mario Jino
State University of Campinas (UNICAMP)
C.P. 6101, 13081 Campinas, SP, Brazil

Abstract

Data Flow based Structural Testing Criteria have been introduced aiming at *"bridging the gap"* between branch testing and path testing, and at making stronger the structural testing criteria, but none in the literature *"bridges the gap"* in the presence of infeasible paths. Potential Uses Criteria Family (PU), based on *the potential use concept*, is introduced. Potential Uses Criteria are analysed in the presence of infeasible (unexecutable) paths; these criteria establish a hierarchy including all-edges and all-paths criteria in addition to satisfying the minimum coverage requirements from the data flow point of view. Potential Uses criteria were inspired the Data Flow Criteria Family (DF); each PU criterion includes its correspondent DF criterion and no other data flow based criterion includes PU criteria. Complexity analysis of other data flow criteria is revisited and disagreement with published results is highlighted; all data-flow based criteria have complexity greater than or equal to 2^t.

Keywords: structural testing criteria, complexity analysis, hierarchy of testing criteria, infeasible paths, data-flow based criteria.

1 Introduction

The major objective of software testing is to reveal the presence of program faults or defects; in other words, to refute the claim that a program is correct. Ideally, a program should be exercised for all possible input values; however, it is known that in practice exhaustive testing is impossible due to time and cost restrictions. Criteria have been

*This work was partly supported by CNPq, CAPES and SID Informática S.A..

devised to provide a systematic way to select an input domain subset — a test case set T — that is relatively small and, even so, effective in refutting the claim that the program is correct, within minimal time and costs.

One class of these criteria, named structural testing criteria, uses implementation information to characterize the required elements. The three most known and well-established criteria are statement testing, branch testing and path testing [Taylor[1992]]; basically, these criteria use control flow information of programs being tested to establish required elements and are referred to as control-flow based criteria.

Motivated by the fact that, even for small programs, control-flow based criteria are more appropriate for revealing the presence of domain-errors than computation-errors (i.e., do not contribute to reveal the presence of simple computation errors/defects) and that, in general, path coverage is impractical (there may be an infinite number of paths), criteria using data flow information have been introduced [Herman[1976], Laski[1983], Ntafos[1984], Rapps[1985], Ural[1988]]. This class of data-flow based criteria is more appropriate for revealing computation errors and requires basically that program paths, from points where variables are defined to points where those definitions are used, to be exercised by the test case set.

In selecting or establishing a structural testing criterion three basic conditions must be fulfilled:

- i) branch coverage must be ensured;
- ii) from the data-flow information point of view, one use of every computation result must be required (and ensured) by the criterion [Clarke[1985]]; and
- iii) for any program P, the test case set required by the criterion must be finite.

Condition iii) takes into account costs and practical considerations. The first condition is the minimal requirement of most testing strategies and ensures that every statement is exercised. The second condition is implied by the weakest data flow based testing criterion, "all-defs" [Rapps[1985]].

Although data-flow based criteria, such as Data Flow Criteria Family [Rapps[1985]] and Ntafos' criteria [Ntafos[1984]], were introduced aiming at *"bridging the gap"* between branch testing and path testing, and at making stronger the structural testing criteria, none available in the literature *"bridges the gap"* in the presence of infeasible paths — a path is infeasible if there is no assignment of values to the input variables which causes the path to be executed [Maldonado[1991]]; for example, Frankl[1987],[1988] pointed out that DFCF does not *"bridge the gap"* in the presence of infeasible paths. Notice that in practice the ocurrence of infeasible paths in programs is quite common.

The Potential Uses Criteria Family (PUCF) although strongly based on the Data Flow Criteria Family is fundamentally different; these criteria use the concept of *potential use* . Potential Uses (PU)Criteria satisfy the three conditions discussed above; more specifically, *"bridge the gap"* between branch testing and path testing even in the presence of infeasible paths. They require essentially the execution of definition clear paths from every node i containing variable definitions, even if there are no uses of these variables through these paths; if a use might occur in a path — a *potential use* — we require this path to be examined during testing activities. Moreover, requiring the execution of definition clear paths w.r.t. a variable x, independently of use occurrences of x, enables us to verify if the value of x is not being modified through these paths, possibly due to collateral effects, for example. Also, they may make easier the detection of faults caused by missing data flow dependences [Podgurski[1990]] originated by a missing variable use, for example, as illustrated in Fig. 1. Additionally, the automation of PU Criteria requires less static analysis than other data flow criteria do, since it is not necessary to identify uses of variables.

Comparison studies of these criteria based on *inclusion relation* and *complexity analysis* have been conducted [Clarke[1985], Rapps[1985], Weyuker[1984], Ntafos[1988]]. The inclusion relation and the complexity of a criteria must be considered in defining or in selecting a criterion. The inclusion relation establishes a hierarchy (partial order) for these criteria and provides information on testing effectiveness and adequacy to classes of errors. Test selection criterion c_1 *includes* test selection criterion c_2 if, for any given control flow graph G, any set of complete paths of G satisfying c_1 also satisfies c_2. Test selection criterion c_1 *strictly includes* criterion c_2, denoted by $c_1 \Rightarrow c_2$, provided c_1 includes c_2 and for some graph G there is a set of complete paths of G satisfying c_2 but not c_1. If neither $c_1 \Rightarrow c_2$ nor $c_2 \Rightarrow c_1$, we say that these criteria are *incomparable*. One of the strongest data-flow based testing [Taylor[1992]], *all-du-paths* [Rapps[1985]] is included by some of the *PU Criteria Family* ; no data-flow based testing criteria include Potential Uses Criteria [Maldonado[1991]].

Complexity of a testing criterion is defined as the upper bound on the number of test cases needed to satisfy the criterion. All data-flow based criteria, including PU Criteria, have complexity greater than or equal to 2^t [Maldonado[1991]], even the one at the bottom of the hierarchy — the all-defs criterion. As a common contributor to the cost of testing activities — generation of a test case, execution of test cases and results evaluation — is the number of test cases needed to satisfy each criterion, the relevance of benchmarking such criteria is obvious; the same benchmark[1] established by Weyuker[1990] has been used with a testing tool named POKE-TOOL [Maldonado[1989a], Chaim[1991]]. POKE-TOOL (a POtential use Criteria for program testing TOOL) consists in a multi-language testing tool that supports the use of PU Criteria; in the present configuration, it is operational for programming language C and is being instantiated for COBOL, FORTRAN and PASCAl.

Results of this benchmark are very promising: the number of test cases required to satisfy any of the Potential Use Criteria is linear in t, where t is the number of decision commands in P; the maximum number of test cases (the empirical worst case) required by the most demanding PU criteria was $4 * t$. These results are an indication that satisfaction of these criteria can be adopted as a practical goal of testing; they also provide data for comparison studies in terms of the "average" number of test cases needed [Maldonado[1991]]. A more complete discussion of this benchmark will appear in a forthcoming paper.

In Section 2, basic terminology and concepts are introduced. Potential Uses (PU) and Feasible Potential Uses (FPU) Criteria Families are introduced in Section 3. The inclusion analysis is presented in Section 4. Complexity analysis of PU and FPU criteria is presented in Section 5; complexity analysis of other data-flow testing criteria is also discused. Conclusions are presented in Section 6.

[1]29 programs from Kernighan[1981], translated into C

2 Potential Uses Criteria — Terminology

A program P is decomposed into a set of disjoint blocks where each block has the property that, whenever the first statement is executed, the other statements are executed in the given order. The representation of a program P as a control flow graph, $G = (N, E, s)$, consists in establishing a correspondence between nodes and blocks, and indicating possible flow of control between blocks through edges; N is the set of nodes, E the set of edges, and s the inicial node. We consider every program graph as a directed, connected graph having a unique inicial node $s \in N$ and a unique exit node $e \in N$.

A *path* is a finite sequence of nodes $(n_1, n_2, \ldots, n_k)$, $k \geq 2$, such that there is an edge from n_i to n_{i+1} for $i = 1, 2, \ldots, k-1$. A path is a *simple path* if all nodes, except possibly the first and the last, are distinct. If all nodes are distinct it is a *loop-free path.* A *complete path* is a path where the first node is the inicial node and the last node is the exit node.

A variable occurrence in a program can be a *variable definition*, a *variable use* (*c-use* or *p-use*) or an *undefinition.* A *c-use* affects directly the computation being performed or allows one to see the result of an earlier definition. A *p-use* affects directly the flow of control through the program.

A path $(i, n_1, \ldots, n_m, j)$, $m \geq 0$, containing no definitions in nodes $n_1, \ldots, n_m$ of a variable x occurring in a program is called a *definition clear path* with respect to (w.r.t.) x from node i to node j and from node i to edge (n_m, j).

In establishing the Data Flow Criteria Family, Rapps[1985] introduced the concept of *def-use graph.* A *def-use graph* is obtained from the flow graph by associating to each node i the sets *c-use*(i) = { variables x such that there is no definition of x, preceding the $c-use$ of x , within the block i } and *def* (i) = { variables x such that x is defined in block i and there is a definition-clear path from node i to some node containing a $c-use$ or to some edge containing a $p-use$ of x}, and associating to each edge (i, j) the set *p-use* (i, j) = { variables which have p-uses on edge (i, j)}. Also defined are the sets of nodes *dcu* (x, i) = { nodes j such that $x \in$ c-use (j) and there is a definition-clear path with respect to x from i to j} and *dpu* (x, i) = { edges (j, k) such that $x \in$ p-use (j, k) and there is a definition-clear path w.r.t. x from i to (j, k)}. A path $(n_1, n_2, \ldots, n_j, n_k)$ is a *du-path* w.r.t. a variable x if n_1 has a global definition of x and: (1) either n_k has a c-use of x and $(n_1, n_2, \ldots, n_j, n_k)$ is a definition-clear simple path w.r.t. x; or (2) (n_j, n_k) has a p-use of x and $(n_1, n_2, \ldots, n_j, n_k)$ is a definition-clear path w.r.t. x and $n_1, n_2, \ldots, n_j$ is a loop-free path.

A *definition-c-use-association* is a triple (i, j, x) where $x \in def(i)$ and $j \in dcu(x, i)$. A *definition-p-use-association* is a triple $(i, (j, k), x)$ where $x \in def(i)$ and $(j, k) \in dpu(x, i)$. An *association* is a definition c-use association, a definition-p-use association or a du-path.

In the definition of Potential Uses Criteria minor but interesting modifications in these concepts were introduced as follows: *defg*(i) is the set of variables for which node i contains a definition; *pdcu(x,i)* ={nodes j | there is a definition clear path w.r.t x from i to j}; *pdpu(x,i)* ={edges (j, k) | there is a definition-clear path w.r.t x from i to (j, k)}; *def graph* is a graph obtained by associating to each node i of the control flow graph the set defg(i); *potential-du-path* w.r.t. a variable x is a definition-clear path $(n_1, n_2, \ldots, n_j, n_k)$ w.r.t. x from node n_1 to node n_k and to the edge (n_j, n_k) , where path $(n_1, n_2, \ldots, n_j)$ is a loop-free path and n_1 has a definition of x; *potential-definition-c-use association* is a triple $[i,j,x]$ where $x \in defg(i)$ and $j \in pdcu(x,i)$; *potential-definition-p-use association* is a triple $[i,(j,k),x]$ where $x \in defg(i)$ and $(j,k) \in pdpu(x,i)$; and *potential-association* is defined as a potential-definition-c-use association, a potential-definition-p-use association or a potential-du-path. Observe that *every association is a potential-association.* The notation $[i,(j,k),\{v_1,\ldots,v_n\}]$ is also introduced to represent the set of associations $[i,(j,k),v_1],\ldots,[i,(j,k),v_n]$; it indicates that at least one def-clear path w.r.t. $v_1,\ldots,v_n$ from node i to the arc (j,k) exists.

A path $\pi_1 = (i_1,\ldots,i_k)$ is said to be included in a set Π of paths if Π contains a path $\pi_2 = (n_1,\ldots,n_m)$ such that $i_1 = n_j, i_2 = n_{j+1},\ldots,i_k = i_{j+k-1}$, for some j, $1 \leq j \leq m-k+1$. We say π_1 is included in π_2 or that π_1 is a sub-path of π_2.

A complete path π covers a potential-definition-c-use association $[i,j,x]$ [respectively, a potential-definition-p-use association $[i,(j,k),x]$] if it includes a definition clear path w.r.t. x from i to j [respectively, from i to (j,k)]. π covers a potential-du-path π_1 if π_1 is included in π. A set Π of paths covers a potential-association if some element of the set does. Note that, if a path set Π covers an association or a potential-association from node i to node j [respectively, to edge (j,k)], then there exists a definition clear path w.r.t. variable $x \in defg(i)$ from node i to node j [respectively, to edge (j,k)] which is included in Π.

A *complete path is executable or feasible* if there exists some assignment of values to the input variables which cause the path to be executed. A *path is executable* if it is a subpath of an executable complete path. A *potential-association is executable* if there is some executable complete path which covers it; otherwise, it is unexecutable. Two other sets are defined: $fpdcu(x,i) = \{j \in pdcu(x,i) \mid$ *the potential-association* $[i,j,x]$ *is executable* $\}$ and $fpdpu(x,i)=\{(j,k) \in pdpu(x,i) \mid$ *the potential-association* $[i,(j,k),x]$ *is executable* $\}$.

Aiming at keeping, even in the presence of infeasible paths, an hierarchy of criteria that bridges the gap between all-edges and all-paths, new criteria are defined using the cycle-extended concept [Frankl[1988]]. Let $\pi = (n_1,n_2,\ldots,n_k)$ be a du-path (potential-du-path) w.r.t. x; its *cycle-extension* (π,x) is defined as the set of definition clear paths w.r.t. x of the form $(\lambda_1,\lambda_2,\ldots,\lambda_k)$ where each λ_i is a path of length greater than or equal to one, beginning and ending with n_i. Observe that for any du-path [potential-du-path] π w.r.t. x, $\pi \in$ cycle-extension (π,x).

3 Potential Uses and Feasible Potential Uses Criteria Family

Potential Uses (PU) Criteria, initially introduced by Maldonado[1988], examine every possible definition clear paths starting at a definition node (program state change), in order to refute the claim that the program is correct. Some of PU Criteria bridge the gap between $(all-edges)$ and $(all-paths)$ criteria, even in the presence of infeasible paths, establishing a hierarchy of criteria, in addition to satisfying the minimum coverage requirements from the data flow point of view. The other data flow based criteria in the literature do not bridge the gap; none of them includes PU Criteria.

Consider Fig. 1, a modified example from Rapps[1985]; the definition of variable x at node 1 would imply, for example, one potential-association between node 1 and $arc(6,7)$

(as there might be a potential-use of x at $arc(6,7)$), denoted by $[i,(6,7),\{x\}]$, as well as a potential-association between node 1 and node 8 (which has an explicit use of x), denoted by $[1,8,\{x\}]$. The potential-associations $[8,(6,8),\{E\}]$ and $[8,8,\{e\}]$ would also be required despite the occurence of a typo error (a missing data-flow dependence); keep in mind that the aim of software testing is to reveal the presence of defects. Considering the potential-association $[1,(6,7),\{x\}]$, the weakest of the Potential Use Criteria Family — all-potential-uses — would require *at least one path* (loop free or not) from node 1 to edge $(6,7)$ with no definition of variable x to be exercised; the intermediate criterion — all-potential-uses/du — would require at *least one loop free path* from node 1 to edge $(6,7)$; the strongest criterion — all-potential-du-paths — would require *every loop free path* from node 1 to edge $(6,7)$.

Potential Uses Criteria

Definition 1 *All-potential-uses criterion - Requires all associations* $[i,j,x] \mid j \in pdcu(x,i)$ *and all associations* $[i,(j,k),x] \mid (j,k) \in pdpu(x,i)$ *for each node* $i \in G$ *and for each* $x \in defg(i)$.

Definition 2 *All-potential-uses/du criterion - Requires, for each* $i \in G \mid defg(i) \neq \emptyset$, *a potential-du-path from* i *to* j *w.r.t.* x *for all associations* $[i,j,x] \mid j \in pdcu(x,i)$ *and a potential-du-path from* i *to* (j,k) *w.r.t.* x *for all associations* $[i,(j,k),x] \mid (j,k) \in pdpu(x,i)$.

Definition 3 *All-potential-du-paths criterion - Requires, for each* $i \in G \mid defg(i) \neq \emptyset$, *all potential-du-paths from* i *to* j *w.r.t. all variable* $x \in defg(i)$ *for each* $j \in pdcu(x,i)$ *and all potential-du-paths from* i *to* (j,k) *w.r.t. all variable* $x \in defg(i)$ *for each* $(j,k) \in pdpu(x,i)$ *for all* $i \mid defg(i) \neq \emptyset$.

Potential Uses Criteria can be seen as an extension to all-uses and all-du-paths criteria [Rapps[1985]] which require def-clear paths or du-paths w.r.t. variable x to be executed only if there is a use of variable x through these paths; i.e., in these criteria, associations are characterized using $dcu(x,i)$ and $dpu(x,i)$ while $pdcu(x,i)$ and $pdpu(x,i)$ are used in Potential Uses Criteria. Observe that the computational defect introduced in node 8 can be detected only through the execution of path (8,9,5,6,8), which is required by Potential Uses Criteria.

Not all paths in a program are feasible and the identification of feasible and infeasible paths is an undecidable problem [Ural[1988]]; for example, only three programs from the benchmark we have used have no unexecutable path. Thus, PU criteria may require paths and associations that are unexecutable; variations of these criteria have been defined [Maldonado[1989b]], considering infeasible paths, since they do not satisfy the applicability property [Weyuker[1986]]: for every program P there exists some test case set which is *C-adequate* for P. Given a program P and its associated control flow graph (program graph), a set T is said to be *C-adequate* for P (satisfies criterion C for P) if and only if each of the sequences required by C is a sub-path of any path of the set Π, corresponding to the paths executed by the test cases.

Potential Uses Criteria are modified by selecting the required potential-associations from $fpdcu(x,i)$ and $fpdpu(x,i)$ instead of $pdcu(x,i)$ and $pdpu(x,i)$; in other words, the unexecutable associations are eliminated from the required components set, ensuring the applicability property. According to Frankl[1988], transition from a data-flow based criterion to the corresponding feasible data-flow criterion means a trade of the undecidability of the existence question "Is there any test set T which is *C-adequate* for P ?" for the undecidability of the recognition problem "Is a given test set T *C^*-adequate* for P ?" Frankl[1987],[1988] pointed out that, unfortunately, although the FDF criteria satisfy the applicability property, the inclusion relation among FDF criteria has changed significantly compared to DF criteria. In the next section, inclusion analysis of Potential Uses Criteria, taking into account the presence of infeasible paths, is presented.

Attempting to fulfil the three properties discussed in Section 1, even in the presence of infeasible paths, cycle-extended criteria — cycle-extended-potential-du-paths (cyex-potential-du-paths) and cycle-extended-all-potential-uses/du (cyex-all-potential-uses/du) — are defined using the cycle-extension concept [Frankl[1988]]. These criteria require that one element of the cycle-extension (π, x) of a potential-du-path π be selected. Observe that all-potential-uses and cyex-potential-uses/du criteria are equivalent.

The basic PU criteria and the cycle-extended PU criteria constitute the *Potential Uses (PU) Criteria Family*; the correspondent feasible criteria C^* constitute the *Feasible PU Criteria Family (FPU)*. Following, some of these criteria are defined; the other criteria are defined in a similar way.

Definition 4 *$(All-potential-uses/du)^*$ criterion - Requires, for each node $i \in G \mid defg(i) \neq \emptyset$, an executable potential-du-path from i to j w.r.t. x for all associations $[i,j,x] \mid j \in fpdcu(x,i)$ and an executable potential-du-path from i to (j,k) w.r.t. x for all associations $[i,(j,k),x] \mid (j,k) \in fpdpu(x,i)$.*

Definition 5 *Cyex-potential-du-paths: Π satisfies the cyex-potential-du-paths criterion for P if and only if for each variable x and for each potential-du-path π w.r.t. x, Π covers some path $\pi_1 \in$ cycle-extension (π, x).*

4 Partial Order Analysis of Potential Uses Criteria in the Presence of Unexecutable Paths

Analysis of structural testing criteria have been conducted based on the inclusion relation discussed earlier [Clarke[1985], Rapps[1985], Ntafos[1988]]; Frankl[1987],[1988] studied the DF Family in the presence of unexecutable paths. Frankl pointed out that, unfortunately, the inclusion relation has been changed significantly; for example, none of the criteria bridges the gap between $(all-edges)$ and $(all-paths)$. For programs satisfying the No-Anomalies (NA) property — every path from the start node to a use of a variable x must contain a definition —, the FDF criteria would bridge the gap. However, according to Frankl, requiring such a property is overly restrictive since many perfectly good programs fail to satisfy the NA property.

For programs satisfying property *LDEN (At Least one Definition in the Entry Node Property)* — P has at least one variable being defined in the entry node — PU criteria bridge the gap between all- edges and all-paths, even in the presence of infeasible paths;

furthermore, some of them exercise every computation result. Notice that property LDEN is easily fulfilled by the majority of real and practical programs. The partial order for (Feasible) Potential Uses and (Feasible) Data Flow criteria families is presented in Fig. 4 (Fig. 5).

Theorem 1 *The Potential Uses and Data Flow Criteria Families are partially ordered by strict inclusion as shown in Fig. 4. Families FDF and FPU are partially ordered by strict inclusion as shown in Fig. 5. Furthermore, a criterion C_i includes a criterion C_j if and only if it is explicitly shown to do so in Figs. 4 and 5 or if it follows from the transitivity of the relation.*

Proof: Let T be a test case set for a program P (G being the corresponding control flow graph), and let Π be the set of paths executed by T. We present only the most relevant aspects of this proof; for the remainder see Maldonado[1989b],[1991].

•$(all-potential-uses/du)^* \Rightarrow (all-edges)^*$

Suppose T is $(all-potential-uses/du)^*-adequate$ for P. Let edge (i,j) be any executable edge in P. Since edge (i,j) is executable, there is at least one executable complete path $\pi = (I, n_1, n_2, \ldots, i, j, \ldots, F)$ from entry node I to exit node F such that edge (i,j) is included in P. To complete the proof, it must be shown that there is at least one executable potential-du-path from some node n_d to edge (i,j) w.r.t. some variable v defined in n_d. One of these executable potential-du-path will be included in Π because T is $(all-potential-uses/du)^*-adequate$; so edge (i,j) will be included in Π.

i) If node i has a definition of a variable v , then path (i,j) is an executable potential-du-path from node i to edge (i,j).

ii) Consider a path $\pi_i = (I, n_1, n_2, \ldots, n_k, i)$ as a loop-free executable path. Since program P obeys LDEN property, node I has at least one definition of some variable v. If nodes $n_1, n_2, \ldots, n_k$ have no redefinition of variable v defined in node I, path $(I, n_1, n_2, \ldots, n_k, i, j)$ is an executable potential-du-path w.r.t. v from node I to edge (i,j), by definition. If some node n_d, $1 \leq d \leq k$, has a redefinition of v, then path $(n_d, n_{d+1}, \ldots, n_k, i, j)$ is an executable potential-du-path w.r.t. v from node n_d to edge (i,j).

iii) Consider a path $\pi_i = (I, n_1, n_2, \ldots, n_k, i)$ which is not a loop-free path.
Let $(n_l, n_{l+1}, \ldots, n_{l+n}, n_l)$ be the last loop in path π_i before the occurrence of node i, i.e., $\pi_i = (I, n_1, n_2, \ldots, n_l, n_{l+1}, \ldots, n_{l+n}, n_l, n_s, \ldots, n_{s+m}, n_k, i)$. Path $\pi_{i1} = (n_l, n_s, \ldots, n_{s+m}, n_k, i)$ is an executable loop-free path. If some node n_d of path π_{i1} has some definition of a variable v, it is straightforward to see that path $\pi_{ij1} = (n_l, n_s, \ldots, n_{s+m}, n_k, i, j)$ includes an executable potential-du-path w.r.t. v from node n_d to edge (i,j) (part ii). Moreover, it is easy to see that path set $\Pi_i = \{(n_{l+q}, \ldots, n_l, n_s, \ldots, n_k, i) \mid 1 \leq q \leq n\}$ contains only executable loop-free paths. Furthermore, one of the nodes $n_l, n_{l+1}, \ldots, n_{l+n}$ must have a definition to alter the loop condition. Suppose there is such a definition in node $n_d \in \{n_{l+q} \mid 1 \leq q \leq n\}$. Since path $(n_d, \ldots, n_l, n_s, \ldots, n_k, i) \in \Pi_i$, it is straightforward to see that this path includes an executable potential-du-path from node n_d to edge (i,j) (part ii). If node n_l

has such a definition (which alters the loop condition), we have the case of path $\pi_{il} = (n_l, n_s, \ldots, n_k, i)$ with a definition in node n_l.

$\bullet (all-potential-uses)^* \Rightarrow (all-edges)^*$
Suppose T is $(all-potential-uses)^*-adequate$ for P. Let edge (i,j) be any executable edge in P. Since edge (i,j) is executable, there is at least one executable complete path $\pi_c = (I, n_1, n_2, \ldots, n_k, i, j, \ldots, F)$ from the entry node I to the exit node F such that edge (i,j) is included in π_c. To complete the proof, it must be shown that there is at least one executable definition-clear path from some node n_d to the edge (i,j) w.r.t some variable v defined in n_d. Then, the association $[n_d, (i,j), v]$ must be covered by Π, i.e., Π must include at least one executable definition-clear path from n_d to edge (i,j) w.r.t. v; so edge (i,j) will be included in Π. The reasoning is similar as above.

From Fig. 5, it can be extracted that some of the FPU criteria, for the class of programs with property LDEN, "bridge the gap" between $(all-edges)^*$ and $(all-paths)^*$ criteria and still satisfy the basic requirement, from the point of view of data flow, since all of them include the $(all-defs)^*$ criterion. It must be pointed out that property LDEN is fulfilled by the majority of practical programs. On the other hand, none of the DF criteria "bridges the gap" between $(all-edges)^*$ and $(all-paths)^*$ criteria. Also, a hierarchy of criteria including $(all-paths)^*$ and $(all-edges)^*$ criteria is established.

Concerning the other data flow based testing criteria, consider the modified example from Frankl[1987], shown in Fig. 6, and taking into account Frankl's worst case assumption that edges $(5,6)$ and $(5,7)$ are both executable (consider an environment in which uninitialized variables receive arbitrary values); it is easy to conclude that Hermans', Ntafos', Laski's and Ural's criteria, in the presence of infeasible paths, do not bridge the gap. For example, *all simple O/I criterion* does not require path $(1,2,4,5,7,8,9)$; observe that this path is executable. Considering Figs. 1 and 6 and the criteria definitions the conclusion that PU Criteria are *incomparable* with these criteria is immediate.

5 Complexity Analysis

A serious shortcoming of comparisons in terms of inclusion relation is that the cost of the various strategies is not accounted for. The number of test cases needed to satisfy a criterion is a common contributor to the cost of testing activities; complexity analysis gives an upper bound to this number [Ntafos[1988]].

It can be shown [Maldonado[1991]] that the flow graph of Fig. 2 maximizes the number of du-paths, hence, the number of potential-du-paths is given by $((11/2)t + 9)2^t - 10t - 9$ and 2^t test cases are required to cover these potential-du-paths.

Concerning all-potential-uses criterion, it is known from the inclusion analysis (see Section 4) that all-potential-du-paths criterion includes all-potential-uses criterion; then, 2^t is assumed as an initial upper bound for all-potential-uses criterion. For the example of Fig. 3, 2^t test cases are required to satisfy the all-potential-uses criterion; hence, the complexity of all-potential-uses criterion is 2^t.

Cycle-extension criteria also require at most 2^t test cases as they require only one executable path $\pi_1 \in cycle-extension(\pi, x)$ for each potential-du-path π of the control flow graph. Concerning the Feasible Potential Uses Criteria, it seems obvious that they should be less demanding than the Potential Uses Criteria and that, in the worst case, they would require the same number of test cases as the correspondent Potential Uses Criteria; however, using a control flow graph similar to the one in Fig. 2, with an adequate data definition distribution where every path with more than two loop is infeasible, the Feasible Potential Uses Criteria would require more than 2^t test cases, and so their complexity is greater than 2^t [Maldonado[1991]].

It is important to note that the example of Fig. 3 requires 2^t test cases for both all-uses and all-defs criteria, disagreeing with results presented by Weyuker[1984]; for example, Weyuker says that all-uses criterion require at most $(1/4)(t^2+4t+3)$ test cases. According to Ntafos[1988], required pairs, data-contexts, $2 - dr$ iterations criteria have the same complexity as all-uses criteria; but all of them include all-defs criteria and, consequently, they would have complexity greater than or equal to 2^t. Concerning *all simple O/I paths*, from Fig. 2 it is easy to conclude that this criterion, with an adequate distribution of variable uses and definitions, requires at least $2^{2t-2} + 2^{t-1}$ *simple OI-paths* (in this case, complete paths); thus, 2^t is not the complexity of this criterion as established by Ural[1988].

Given that complexity of a criterion is an important factor in testing activities results presented in this section point out that complexity analysis of criteria must be further investigated.

6 Conclusions

We have introduced and analysed the Potential Uses Criteria Family in the presence of unexecutable paths; since these criteria do not fulfil the applicability property [Weyuker[1986]] we defined the Feasible Potential Uses Criteria, by eliminating those required paths that can never be exercised. All these criteria were compared to Data Flow, Feasible Data Flow and cycle-extended Data Flow Criteria Families, based on an inclusion relation and on complexity analysis. One basic consideration in establishing a structural testing criterion is that branch coverage is a necessary condition to be fulfilled, as well as, from the point of view of data flow analysis, that at least one use of every computation result must be required by the criterion being defined [Clarke[1985]] .

For the class of programs satisfying property LDEN, partial order analysis of Potential Uses criteria shows that some of them bridge the gap between $(all - edges)^*$ and $(all - paths)^*$ criteria, establishing a hierarchy of criteria. Additionally, some of them satisfy the minimum coverage requirements from the data flow point of view. It was also shown that data flow based criteria in the literature do not bridge the gap, considering only property LDEN.

All data-flow based criteria have complexity greater than or equal to 2^t; Potential Uses Criteria (PU) have complexity 2^t. Disagreement with some complexity analysis results of other data flow criteria has been pointed out. As complexity of a criterion is an important factor in testing activities, results presented in Section 5 point out that this topic, complexity analysis, must be further investigated.

The introduction of the potential use concept enables us to relax the properties programs have to satisfy, i.e., for the class of programs P which satisfy property LDEN, for some FPU criteria, essentially the same partial order for the corresponding PU criteria holds. Furthermore, even in the presence of unexecutable paths, FPU criteria provide a hierarchy including $(all-edges)^*$ and $(all-paths)^*$ criteria, in addition to satisfying the minimum coverage requirements from the data flow point of view. Note that the property LDEN is easily fulfilled by the majority of real and practical programs.

Results of benchmarking Potential Uses Criteria, using POKE-TOOL, are very promising; for example, the empirical worst case was found to be $4 * t$. POKE-TOOL design is being reviewed to include all the criteria studied in this work; in this way, practical use and comparisons of these criteria will be possible. Facilities to support testing activities in the presence of unexecutable paths have been incorporated into POKE-TOOL (for example, the heuristics proposed by Frankl[1987].

Another direction of future work is the combination of criteria, as each of the data flow based criteria introduces an interesting concept. For example, the *all simple O/I paths* criterion [URA88], aims at capturing the effects of programs inputs to programs output; such identification may be helpful in providing better understanding of a program and in checking consistency of a program with its specifications. This effort will potentially lead to the establishement of stronger criteria by combining the best characteristics of each one.

References:

L. Clarke, A. Podgurski, D. J. Richardson and S. J. Zeil [1985], "A Comparison of Data Flow Path Selection Criteria," in *Proc. of the 8th Int'l Conf. on Software Engineering*, pp. 244-251, Aug. 1985.

M.L. Chaim[1991], "POKE-TOOL — Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado Em Análise de Fluxo de Dados", Master Thesis, DCA/FEE/-UNICAMP, Campinas, SP, Brazil, 1991.

F. G. Frankl[1987], "The Use of Data Flow Information for the Selection and Evaluation of Software Test Data," *Ph.D Dissertation*, New York Univ., New York, Oct. 1987.

F. G. Frankl and E.J. Weyuker[1988], "An Applicable Family of Data Flow Testing Criteria," *IEEE Trans. on Software Eng.*, Vol. 14, No. 10, pp. 1483-1498, Oct. 1988.

P. M. Herman[1976], "A Data Flow Analysis Approach to Program Testing," *The Australian Computer Journal*, Vol. 8, No.3, pp.92-96, Nov.1976.

B. W. Kernighan e P. J. Plauger[1981], *Software Tools in Pascal*, Massachusetts: Addison-Wesley Publishing Company, Reading, 1981.

J. W. Laski e B. Korel[1983], "A Data Flow Oriented Program Testing Strategy," *IEEE Trans. Software. Eng.*, Vol. SE - 9, No. 3, pp. 347-354, May 1983.

J. C. Maldonado, M. L. Chaim, M. Jino[1988], "Seleção de Casos de Testes Baseada nos Critérios Potenciais Usos", in *Proc. II Simpósio Brasileiro de Engenharia de Software*, Canela, RS, Brazil, pp. 24-35, Oct. 1988.

J. C. Maldonado, M. L. Chaim, M. Jino[1989a], "Arquitetura de uma Ferramenta de Teste de Apoio aos Critérios Potenciais Usos ", *Proc. XXII Congresso Nacional de Informática*, São Paulo, SP, Brazil, Sept. 1989.

J. C. Maldonado, M. L. Chaim, M. Jino[1989b], "Feasible Potential Uses Criteria Analysis," *Technical Report* - DCA/FEE/UNICAMP - RT/DCA-001/89 - Campinas, SP, Brazil, 1989.

J. C. Maldonado[1991], "Critérios Potenciais Usos: uma Contribuição ao Teste Estrutural de Software", Doctoral Dissertation, DCA/FEE/UNICAMP - Campinas, SP, Brazil, 1991.

S. C. Ntafos[1984], "On Required Element Testing," *IEEE Trans. Software Eng.*, Vol. SE - 10, pp. 795-803, Nov. 1984.

S. C. Ntafos[1988], "A Comparison of Some Structural Testing Strategies," *IEEE Trans. Software Eng.*, Vol. 14, No. 6, pp. 868-873, Jun. 1988.

A. Podgurski and L. A. Clarke[1990], "A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance," *IEEE Trans. on Software Eng.*, Vol. SE - 16, No. 9, pp. 965-979, Sept. 1990.

S. Rapps and E. J. Weyuker[1985], "Selecting Software Test Data Using Data Flow Information," *IEEE Trans. Software Eng.*, Vol. SE - 11, pp. 367-375, Apr. 1985.

R. N. Taylor, D. L. Levine and C. D. Kelly[1992], "Structural Testing of Concurrent Programs," *IEEE Trans. on Software Eng.*, Vol. 18, No. 3, pp. 206-215, March 1992.

E. J. Weyuker[1984], "The Complexity of Data Flow Criteria for Test Data Selection," *Information Processing Letters*, Vol. 19, No.2, pp. 103-109, Aug. 1984.

E. J. Weyuker[1986], "Axiomatizing Software Test Data Adequacy," *IEEE Trans. on Software Eng.*, Vol. SE - 12, No. 12, pp. 1128-1138, Dec. 1986.

E. J. Weyuker[1990], "The Cost of Data Flow Testing: An Empirical Study," *IEEE Trans. on Software Eng.*, Vol. SE - 16, No. 2, pp. 121-128, Feb. 1990.

H. Ural and B. Yang[1988], "A Structural Test Selection Criterion," *Information Processing Letters*, Vol. 28, pp. 157-163 Jul. 1988.

Acknowledgment
The authors wish to thank E. J. Weyuker and C. J. P. Lucena for their valuable comments which helped to improve this paper.

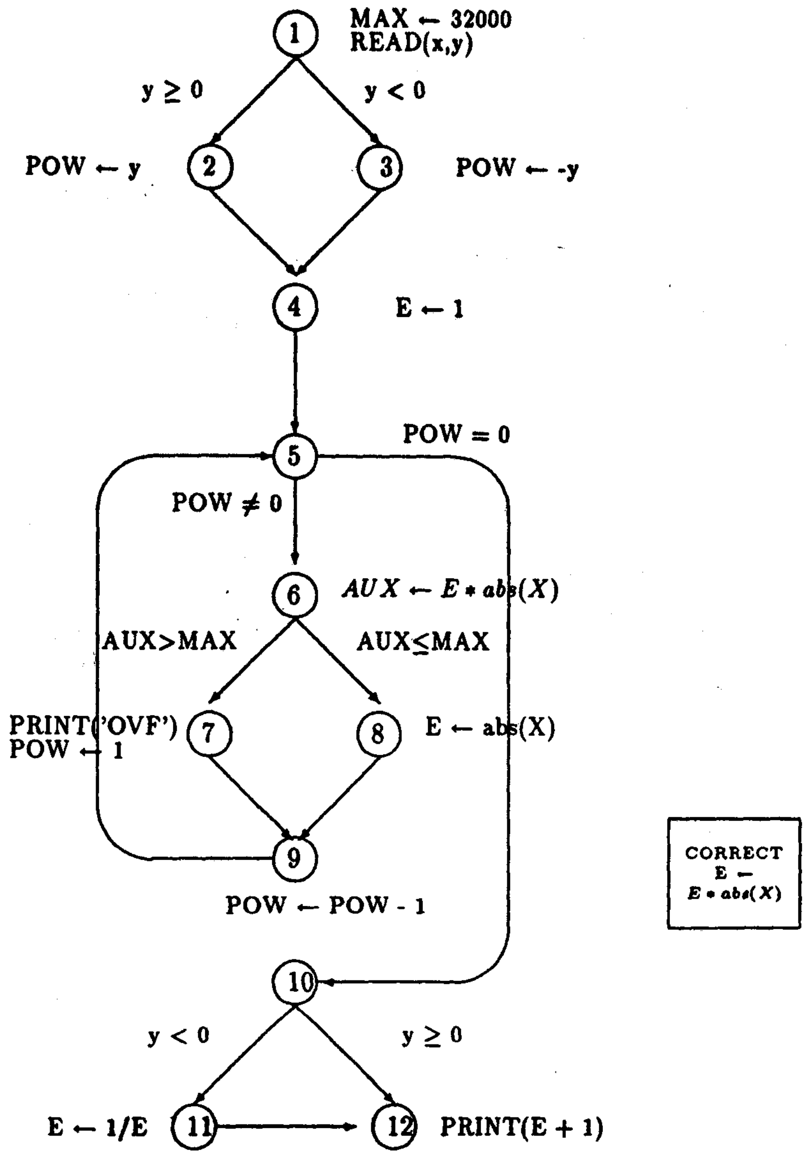

Figure 1: Example to Illustrate Potential Uses Criteria Application

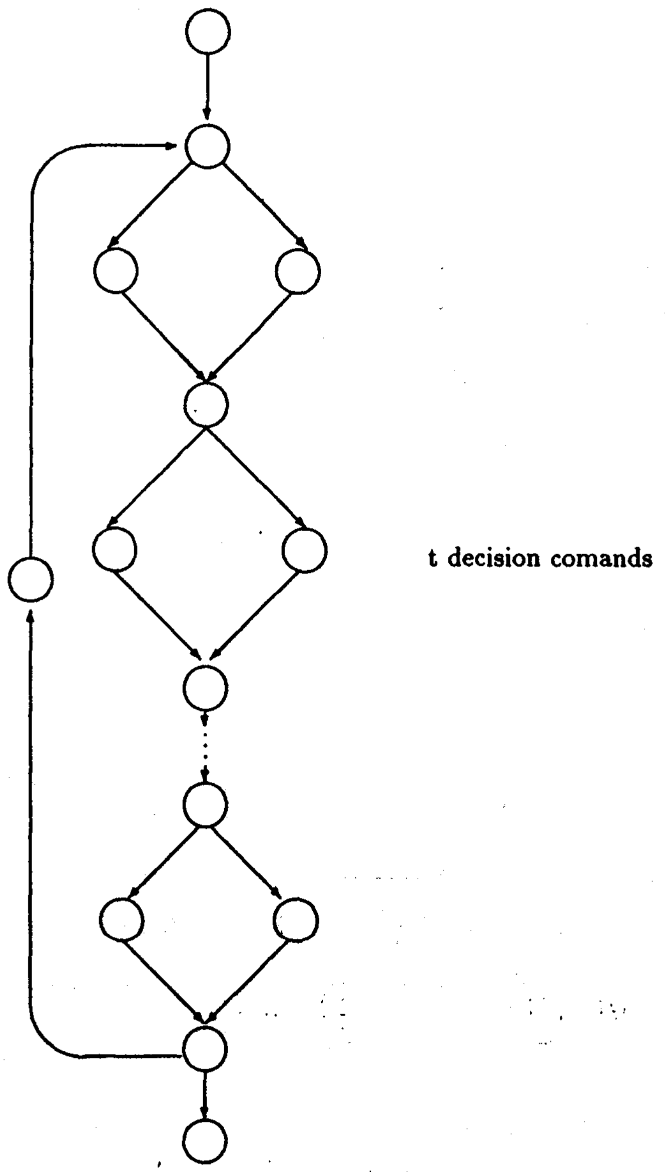

Figure 2: Control Flow Graph that Maximizes the Number of Potential Du-paths

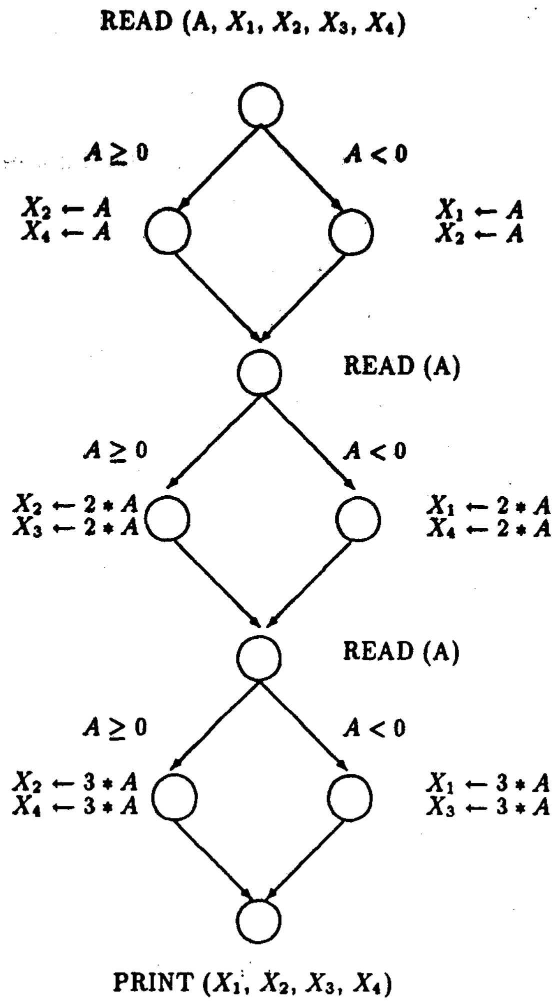

Figure 3: An Example Used in the Complexity Analysis

$$\begin{tikzcd}
 & & \text{All-paths} \arrow[d] & & \\
 & & \text{All-potential-du-paths} \arrow[dll] \arrow[d] \arrow[drr] & & \\
\text{All-du-paths} \arrow[dr] & & \text{Cycle-extended-potential-du-paths} \arrow[dl] \arrow[dr] & & \text{All-potential-uses/du} \arrow[dl] \\
 & \text{Cycle-extended-du-paths} \arrow[dr] & & \text{All-potential-uses} \arrow[dl] & \\
 & & \text{All-uses} \arrow[dl] \arrow[dr] & & \\
 & \text{All-c-uses/some-p-uses} \arrow[dr] & & \text{All-p-uses/some-c-uses} \arrow[dl] \arrow[d] & \\
 & & \text{All-defs} & \text{All-p-uses} \arrow[d] & \\
 & & & \text{All-edges} \arrow[d] & \\
 & & & \text{All-nodes} &
\end{tikzcd}$$

Figure 4: Partial Order for Potential Uses and Data Flow Criteria Families

$$\begin{tikzcd}[column sep=tiny]
 & & & \text{(All-paths)}^{*} \arrow[dl] \arrow[dr] & & & \\
 & & \text{(Cycle-extended-potential-du-paths)}^{*} \arrow[dl] \arrow[dr] & & \text{(All-potential-du-paths)}^{*} \arrow[dl] \arrow[dr] & & \\
 & \text{(Cycle-extended du-paths)}^{*} \arrow[dr] & & \text{(All-potential-uses)}^{*} \arrow[dl] \arrow[dr] & \text{(All-potential-uses/du)}^{*} \arrow[d] & \text{(All-du-paths)}^{*} & \\
 & & \text{(All-uses)}^{*} \arrow[dl] \arrow[dr] & & \text{(All-edges)}^{*} \arrow[d] & & \\
 & \text{(All-c-uses/some-p-uses)}^{*} \arrow[dr] & & \text{(All-p-uses/some-c-uses)}^{*} \arrow[dl] \arrow[dr] & \text{(All-nodes)}^{*} & & \\
 & & \text{(All-defs)}^{*} & & \text{(All-p-uses)}^{*} & &
\end{tikzcd}$$

Figure 5: Partial Order for Feasible Potential Uses and Feasible Data Flow Criteria Families

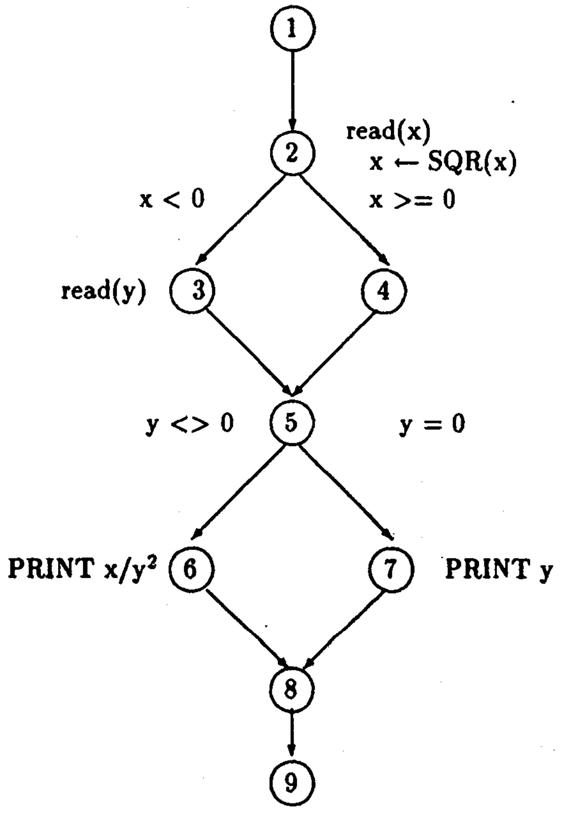

Figure 6: Example of Infeasible Paths