A FORMAL EVALUATION OF DATA FLOW PATH SELECTION CRITERIA

Lori A. Clarke*
Andy Podgurski*
Debra J. Richardson†
Steven J. Zeil*

COINS Technical Report 88–73
July 1988

**Software Development Laboratory*
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

† Information and Computer Science Department
University of California
Irvine, California 92717

KEYWORDS: Software Testing, Path Selection, Data Flow Analysis

This is a revised version of COINS Technical Report 86–49.

This work was supported by grants DCR–8404217 from the National Science Foundation, CCR–8704478 from the National Science Foundation with cooperation from the Defense Advanced Research Projects Agency (ARPA Order 6104), 84M103 from Control Data Corp[oration, and RADC grant F30602–86–C–0006.

ABSTRACT

A number of path selection criteria have been proposed throughout the years. Unfortunately, little work has been done on comparing these criteria. To determine what would be an effective path selection criterion for revealing errors in programs, we have undertaken an evaluation of these criteria. This paper reports on the results of our evaluation of path selection criteria based on data flow relationships. We show how these criteria relate to each other, thereby demonstrating some of their strengths and weaknesses. In addition, we suggest minor changes to some criteria that improve their performance. We conclude with a discussion of the major limitations of these criteria and directions for future research.

1. INTRODUCTION

One of the concerns of software testing is selecting test data that will adequately exercise the various statements in a program. Stucki showed that, left to their own devices, programmers do a poor job of selecting test data that provides good program coverage. This has led to the development of a number of coverage criteria. A coverage criterion is satisfied by certain sets of paths through a program, where a path is a sequence of statements. An effective criterion requires paths with a high probability of revealing errors — that is, when the program is run with test data that causes the selected paths to be executed, there is a high probability that errors, if they exist, will be exposed by those test runs. Of course, the effectiveness of such a criterion depends not only on the selected paths but also on the test data for those paths. In this paper, we do not address the test data selection problem but look only at the path selection problem.

Testing all the paths in a program is often impossible, because programs sometimes contain an infinite number of paths. Thus, a practical path selection criterion should specify only a finite subset of a program's paths. It is generally agreed that, at a minimum, this subset should require that every branch, and thus every statement, in a program be executed at least once. It has been repeatedly shown that this minimum requirement, although important, is far from effective. Several other factors, such as loop coverage and data relationships, should also be considered. Thus, there have been a number of more thorough path selection criteria proposed throughout the years [**?**, **?**, **?**, **?**, **?**]. Unfortunately, there has been little work done on comparing or evaluating the different criteria. We are currently undertaking a study of path selection criteria, working toward the formulation of a more effective criterion that builds upon the strengths of existing ones. As a first step in this study, we are evaluating the path selection criteria that are based on data flow relationships [?, ?, ?]. This paper reports on how these criteria relate to each other and demonstrates some of their strengths and weaknesses.

The authors of these criteria defined them using different terminologies. To facilitate the comparison and simplify the discussion, we define all the criteria using a single set of terms. Although our definitions of the criteria are usually equivalent in meaning to those originally given, some are not. This occurs for two reasons. First, some of the original definitions are ambiguous. Second, the original, formal definitions of the criteria often differ from the stated intent of their authors. In both cases we have tried to redefine the criteria in ways that strengthen them yet seem consistent with the intent of their authors.

In this paper, we formally compare data flow path selection criteria. The next section defines the terms we use throughout this paper. Section 3 defines the criteria we are evaluating using the terminology presented in Section 2. In Section 4 we compare each criterion to the others and present a subsumption graph showing their relationships. One of the major weaknesses of all these criteria is that they are strictly graph theoretic and do not consider problems such as infeasible paths. The conclusion discusses the infeasible path problem as well as other issues that must be considered in order to evaluate these criteria more meaningfully and, more importantly, in order to formulate a more effective path selection criterion. This paper lays the foundation for such future research.

2. TERMINOLOGY

Our evaluation considers the application of a path selection criterion to a *module*. To simplify the discussion, we assume a module is either a main program or a single subprogram and has only one entry and one exit point. In applying a path selection criterion, a module is represented by a directed graph that describes the possible flow of control through the module. A *control flow graph* of a module M is a directed graph $G(M) = (N, E, n_{start}, n_{final})$, where N is the (finite) set of nodes, $E \subseteq N \times N$ is the set of edges, $n_{start} \in N$ is called the *start node*, and $n_{final} \in N$ is called the *final node*. Each node in N, except the start node and the final node, represents a statement fragment in M, where a statement fragment can be a part of a statement or a whole statement. We assume the control flow graphs are defined so that each assignment statement is represented by a node, as is the predicate from each conditional statement. For each pair of distinct nodes m and n in N for which there is a possible transfer of control from the statement fragment represented by m to that represented by n, there is a single edge (m, n) in E. There is also an edge in E from the start node to the entry point of M and an edge in E from the exit point to the final node. We also assume that E contains no edges of the form (n, n).

A control flow graph defines the paths within a module. Let $G(M) = (N, E, n_{start}, n_{final})$ be a control flow graph. A *subpath* in $G(M)$ is a finite, possibly empty, sequence of nodes $p = (n_1, n_2, \ldots, n_{|p|})$ [1] such that for all i, $1 \leq i < |p|$, $(n_i, n_{i+1}) \in E$. A subpath formed by the concatenation of two subpaths p_1 and p_2 is denoted by $p_1 \cdot p_2$. An *initial subpath* is a subpath whose first node is the start node n_{start}. A *path* is an initial subpath whose last node is the final node, n_{final}. The set of all paths in $G(M)$ is denoted by $PATHS(M)$. The graph $G(M)$ is *well-formed* iff every node in N occurs along some path in $PATHS(M)$. In this paper, we consider only well-formed control flow graphs.

A *loop* of a control flow graph $G(M)$ is a strongly-connected subgraph of $G(M)$ corresponding

[1] We denote the *length* of (the number of elements in) a sequence $\mathbf{s}$ by $|\mathbf{s}|$.

to a looping construct in module M. An *entry node* of a loop L is a node n in L such that there is an edge (m, n) in $G(M)$, where m is not in L. An *exit node* for L is a node n outside L such that there is an edge (m, n) in $G(M)$, where m is in L. We assume that all loops have single entry and single exit nodes.

We will frequently need to distinguish between several types of subpaths that visit loops. A *cycle* is a subpath of length ≥ 2 that begins and ends with the same node. A cycle $(n) \cdot p \cdot (n)$ such that the nodes of p are distinct and do not include n is called a *simple cycle.* A *traversal* of a loop L is a subpath within L that begins with the entry node of L, does not return to that node, and ends with a predecessor of either the entry node or the exit node of L. A traversal of a loop represents a single iteration of the loop or possibly a "fall through" execution of the loop. A subpath is said to *traverse* a loop L if the subpath contains a traversal of L. Finally, consider a complete execution of a loop, which consists of one or more consecutive traversals of that loop. A *complete loop-subpath* or *cl-subpath* for a loop L is a subpath $(m) \cdot p \cdot (n)$ [2] such that p is a nonempty subpath lying entirely within L and m and n occur outside L. A cl-subpath represents a fall-through execution of a loop or contains at least one cycle.

The path selection criteria described in this paper are based on data flow analysis and thus are concerned with definitions and uses of variables. Let x be a variable in a module M. A *definition* of x is associated with each node n in $G(M)$ that represents a statement fragment that can assign a value to x; this definition is denoted by $d_n(x)$. The set of variables for which there is a definition associated with a particular node n is denoted by $\mathit{DEFINED}(n)$. A *use* of x is associated with each node n in $G(M)$ that represents a statement fragment that can access the value of x; this use is

[2]Note that n is the loop exit node and could be n_{final}.

denoted by $u_n(x)$ [3] . The set of variables for which there is a use associated with a particular node n is denoted by $USED(n)$.

A use $u_n(x)$ is called a *predicate use* iff node n represents the predicate from a conditional branch statement; otherwise $u_n(x)$ is called a *computation use.* Note that a predicate use is associated with any node having two or more successors. A node representing a predicate is assumed to have at least one variable use but no definitions associated with it.

Data flow analysis is concerned not only with the definitions and uses of variables, but also with subpaths from definitions to statements where those definitions are used. A *definition-clear subpath* with respect to (wrt) a variable x is a subpath p such that for all nodes n in p, $x \notin DEFINED(n)$ and x does not become undefined at the statement fragment represented by n. A definition $d_m(x)$ *reaches* a use $u_n(x)$ iff there is a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x. It is possible that a given definition might not reach any use or that a given use might not be reached by any definition. Since these anomalies are normally considered to be errors and are easily detectable by static analysis [**?**], we assume that every definition of a variable x reaches at least one use of x and that every use of x is reached by at least one definition of x.

When a module receives information from a calling module via parameters or global variables, we add a node, n_{in}, to the control flow graph and associate with it definitions of those variables importing information. The edge (n_{start}, m), where m is the node representing the entry point of the module, is replaced by the edges (n_{start}, n_{in}) and (n_{in}, m). We assume that there is at least one definition associated with a control flow graph, although this definition may be associated with n_{in}. Similarly, when a module returns information via parameters or global variables, we add a node,

[3]When nodes are subscripted, as in n_i, we abbreviate $d_{n_i}(x)$ to $d_i(x)$ and abbreviate $u_{n_i}(x)$ to $u_i(x)$.

n_{out}, to the control flow graph and associate with it uses of those variables exporting information from the module. The edge (m, n_{final}), where m is the node representing the exit point of the module, is replaced by the edges (m, n_{out}) and $(n_{out}, n_{\mathit{final}})$. [4]

A *path selection criterion*, or simply a *criterion*, is a predicate that assigns a truth value to any pair (M, P), where M is a module and P is a subset of $\mathit{PATHS}(M)$. A pair (M, P) *satisfies* a criterion C iff $C(M, P) = \mathit{true}$. A path selection criterion C_1 *subsumes* a criterion C_2 iff every pair (M, P) that satisfies C_1 also satisfies C_2. Two criteria are *equivalent* iff each subsumes the other. A criterion C_1 *strictly subsumes* a criterion C_2 iff C_1 subsumes C_2 but C_2 does not subsume C_1. Two criteria are *incomparable* iff neither criterion subsumes the other. Note that the subsumption relation defines a partial order on any set of path selection criteria.

3. DEFINITIONS OF THE CRITERIA

In this section, we define the family of path selection criteria proposed by Rapps and Weyuker, the Required k-Tuples criteria proposed by Ntafos, and the three criteria proposed by Laski and Korel. We remind the reader that the following assumptions have been made:

1. *There are no edges of the form (n, n_{start}) or (n_{final}, n);*
2. *There are no edges of the form (n, n);*
3. *Every control flow graph is well-formed;*
4. *Every loop has a single entry and single exit node;*
5. *At least one variable use is associated with a node representing a predicate;*
6. *No variable definitions are associated with a node representing a predicate;*

[4]Note that since n_{start} and n_{final} do not represent statement fragments, there are no definitions or uses associated with either node.

7. *Every definition of a variable reaches at least one use of that variable;*
8. *Every use of a variable is reached by at least one definition of that variable;*
9. *Every control flow graph contains at least one variable definition;*
10. *No variable definitions or uses are associated with n_{start} or n_{final}.*

3.1 The Rapps and Weyuker Family of Criteria

Rapps and Weyuker define a family of path selection criteria and analyze these criteria in an attempt to specify the subsumption relationships that exist among the members of the family [**?**, **?**, **?**, **?**]. This family includes three well-known control flow criteria and some new path selection criteria based on the concepts of data flow analysis.

The control flow criteria considered by Rapps and Weyuker are All-Paths (path coverage), All-Edges (branch coverage), and All-Nodes (statement coverage). The All-Paths criterion requires that a path set contain every path through a module's control flow graph. The All-Edges and All-Nodes criteria require that a path set cover every edge and every node, respectively.

> *The pair (M, P) satisfies the* **All-Paths** *criterion iff $P = PATHS(M)$.*
> *The pair (M, P) satisfies the* **All-Edges** *criterion iff for all edges e, there is at least one path in P along which e occurs.*
> *The pair (M, P) satisfies the* **All-Nodes** *criterion iff for all nodes n, there is at least one path in P along which n occurs.*

It is well-known that (for well-formed graphs) All-Paths subsumes All-Edges, which subsumes All-Nodes. For most modules M, the only pairs (M, P) that satisfy the All-Paths criterion are those whose path set P is infinite. In such a case, of course, M will necessarily contain a loop. Thus, All-Paths is not useful for such modules. On the other hand, important combinations of nodes and/or edges might not be required by either All-Edges or All-Nodes. The data flow criteria developed by Rapps and Weyuker distinguish combinations that are important in terms of the flow of data through a module. Their criteria direct the selection of definition-clear subpaths between definitions and uses reached by those definitions.

Rapps and Weyuker first define a criterion that forces each definition to be used. The All-Defs criterion requires that a path set contain at least one definition-clear subpath from each definition to some use reached by that definition.

> *The pair (M, P) satisfies the* **All-Defs** *criterion iff for all definitions $d_m(x)$, there is at least one subpath $(m) \cdot p \cdot (n)$ in P such that p is definition-clear wrt x and there is a use $u_n(x)$.*

Next, Rapps and Weyuker define a criterion that requires that all uses reached by a definition be covered. The All-Uses criterion requires that a path set contain at least one definition-clear subpath from each definition to each use reached by that definition and each successor of the use. The significance of the successor node is that it forces all branches to be taken following a predicate use.

> *The pair (M, P) satisfies the* **All-Uses** *criterion iff for all definitions $d_m(x)$, all uses $u_n(x)$ reached by $d_m(x)$, and all successors n' of node n, P contains at least one subpath $(m) \cdot p \cdot (n, n')$ such that p is definition-clear wrt x.*

Rapps and Weyuker define three criteria that are similar to All-Uses but that distinguish between computation uses and predicate uses. The All-C-Uses/Some-P-Uses criterion requires that a path set contain at least one definition-clear subpath from each definition to each computation use reached by that definition; if the definition reaches only predicate uses, the path set must contain at least one definition-clear subpath from the definition to a predicate use.

> *The pair (M, P) satisfies the* **All-C-Uses/Some-P-Uses** *criterion iff for all definitions $d_m(x)$:*

> 1. *For all computation uses $u_n(x)$ reached by $d_m(x)$, P contains at least one subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x;*
> 2. *If there is no computation use of x reached by $d_m(x)$, then for at least one predicate use $u_n(x)$, P contains a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x.*

The All-P-Uses/Some-C-Uses criterion requires that a path set contain at least one definition-clear subpath from each definition to each predicate use reached by that definition and each successor of that use; if the definition reaches only computation uses, the path set must contain at least one definition-clear subpath from the definition to a computation use.

> *The pair (M, P) satisfies the* **All-P-Uses/Some-C-Uses** *criterion iff for all definitions $d_m(x)$:*
> 1. *For all predicate uses $u_n(x)$ reached by $d_m(x)$ and all successors n' of node n, P contains at least one subpath $(m) \cdot p \cdot (n, n')$ such that p is definition-clear wrt x;*
> 2. *If there is no predicate use of x reached by $d_m(x)$, then for at least one computation use $u_n(x)$, P contains a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x.*

The All-P-Uses criterion requires that a path set contain at least one definition-clear subpath from each definition to each predicate use reached by that definition and each successor of the use.

> *The pair (M, P) satisfies the* **All-P-Uses** *criterion iff for all definitions $d_m(x)$, all predicate uses $u_n(x)$ reached by $d_m(x)$, and all successors n' of node n, P contains at least one subpath $(m) \cdot p \cdot (n, n')$ such that p is definition-clear wrt x.*

The final criterion, All-DU-Paths (DU stands for definition-use), goes a step further than All-Uses; rather than requiring one definition-clear subpath from every definition to all the successor nodes of each use reached by that definition, All-DU-Paths requires every such definition-clear subpath that is a simple cycle or is cycle-free. This limitation on cycles is included to ensure that the path set is finite.

> *The pair (M, P) satisfies the* **All-DU-Paths** *criterion iff for all definitions $d_m(x)$, all uses $u_n(x)$ reached by $d_m(x)$, and all successor nodes n' of n, P contains every subpath*

$(m) \cdot p \cdot (n, n')$ such that $(m) \cdot p \cdot (n)$ is a simple cycle or is cycle-free and p is definition-clear wrt x.

3.2 Ntafos' Required k-Tuples Criteria

Ntafos also uses data flow information to overcome the shortcomings of using control flow information alone to select paths. He defines a class of path selection criteria, based on data flow analysis, called Required k-Tuples [**?**, **?**, **?**]. These criteria require that a path set cover chains of alternating definitions and uses, called *k-dr* interactions. In a *k-dr* interaction, each definition reaches the immediately following use, which occurs at the same node as the next definition in the chain. Thus a *k-dr* interaction propagates information along a subpath, which is called an interaction subpath for the *k-dr* interaction.

The Required k-Tuples criteria are only defined for $k \geq 2$. For $k \geq 2$, a *k-dr interaction* is a sequence $\kappa = [d_1(x_1), u_2(x_1), \ldots, d_{k-1}(x_{k-1}), u_k(x_{k-1})]$ of $k-1$ definitions and $k-1$ uses associated with k distinct nodes $n_1, n_2, \ldots, n_k$, where for all i, $1 \leq i < k$, the ith definition $d_i(x_i)$ reaches the ith use $u_{i+1}(x_i)$. Note that, although the nodes must be distinct, the variables $x_1, x_2, \ldots, x_{k-1}$ need not be distinct. An *interaction subpath* for κ is a subpath $p = (n_1) \cdot p_1 \cdot \ldots \cdot (n_{k-1}) \cdot p_{k-1} \cdot (n_k)$ such that for all i, $1 \leq i < k$, subpath p_i is definition-clear wrt x_i.

A Required k-Tuples criterion is satisfied by a pair (M, P) only if there is at least one interaction subpath in P for every *l-dr* interaction in $G(M)$, $2 \leq l \leq k$. In addition, P must exercise certain branches and loops with particular kinds of subpaths. To exercise all branches from a predicate use $u_l(x_{l-1})$ that ends an *l-dr* interaction λ, P must contain a subpath $p \cdot (m)$ for each successor m of node n_l, where p is an interaction subpath for λ. To exercise loops, if L is the innermost loop containing the first definition or last reference of an *l-dr* interaction λ, then P must contain subpaths that both cover λ and exercise L a minimal and larger number of times.

In Ntafos' definition of the Required k-Tuples criteria, definitions and uses of all the variables in a module are associated with a "source" and "sink" node, respectively. To achieve the same effect, we require that: (1) the control flow graphs to which Ntafos' criteria are applied *always* contain the nodes n_{in} and n_{out}, (2) definitions of all variables (not just those that import information) are associated with n_{in}, and (3) uses of all variables (not just those that export information) are associated with n_{out}.

We now formally define the Required k-Tuples criteria. Let k be a fixed integer, $k \geq 2$.

The pair (M, P) satisfies the **Required k-Tuples** *criterion iff for all l-dr interactions λ in $G(M)$, $2 \leq l \leq k$, each of the following conditions holds:*

1. *For all successors m of the node n_l associated with the last use in λ, P contains a subpath $p \cdot (m)$ such that p is an interaction subpath for λ;*
2. *If the node n_1 associated with the first definition in λ occurs in a loop, then P contains subpaths $p = p_1 \cdot (n_1) \cdot p_2 \cdot p_3$ and $p' = p'_1 \cdot (n_1) \cdot p'_2 \cdot p'_3$ such that: $(n_1) \cdot p_2 \cdot p_3$ and $(n_1) \cdot p'_2 \cdot p'_3$ begin with interaction subpaths for λ, $p_1 \cdot (n_1) \cdot p_2$ is a cl-subpath for the loop L immediately containing n_1* [5] *that traverses L a minimal number of times, and $p'_1 \cdot (n_1) \cdot p'_2$ is a cl-subpath for L that traverses L some larger number of times;*
3. *If the node n_l associated with the last use in λ occurs in a loop, then P contains subpaths $p = p_1 \cdot p_2 \cdot (n_l) \cdot p_3$ and $p' = p'_1 \cdot p'_2 \cdot (n_l) \cdot p'_3$ such that: $p_1 \cdot p_2 \cdot (n_l)$ and $p'_1 \cdot p'_2 \cdot (n_l)$ end with interaction subpaths for λ, $p_2 \cdot (n_l) \cdot p_3$ is a cl-subpath for the loop L immediately containing n_l that traverses L a minimal number of times, and $p'_2 \cdot (n_l) \cdot p'_3$ is a cl-subpath for L that traverses L some larger number of times.*

Our definition for a Required k-Tuples criterion differs from that given by Ntafos in that his requires interaction subpaths only for every *k-dr* interaction, while ours require interaction subbpaths

[5] A loop L *immediately contains* a node iff L contains the node and there is no subloop of L that also contains it.

for every l-dr interaction, where $l \leq k$. Since for any module there exists a constant n such that there are no k-dr interactions for $k > n$, Ntafos' Required k-Tuples criterion does not necessarily subsume his Required $(k-1)$-Tuples criterion for a fixed $k > 2$. It is clear from Ntafos' examples, however, that he did intend the Required k-Tuples criterion to subsume the Required $(k-1)$-Tuples criterion for $k > 2$. Note that our definition of the criteria assures this.

3.3 The Laski and Korel Criteria

Laski and Korel define three path selection criteria based on data flow analysis [**?**]. Their criteria emphasize the fact that a given node may contain uses of several different variables, where each use may be reached by several definitions occurring at different nodes. Laski's and Korel's criteria are concerned with selecting subpaths along which the various combinations of definitions reach the node. We refer to their three criteria as the Reach Coverage criterion, the Context Coverage criterion, and the Ordered Context Coverage criterion.

The Reach Coverage criterion was originally defined by Herman [**?**]. It requires that a path set contain at least one subpath between each definition and each use reached by that definition.

> *The pair (M, P) satisfies the* **Reach Coverage** *criterion iff for all definitions $d_m(x)$ and all uses $u_n(x)$ reached by $d_m(x)$, P contains at least one subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x.*

Before defining the remaining two criteria, some additional terminology must be introduced. Let n be a node in a control flow graph $G(M)$, and let $\{x_1, x_2, \ldots, x_k\}$ be a nonempty subset of $USED(n)$. An *ordered definition context* of node n is a sequence of definitions $ODC(n) = [d_1(x_1), d_2(x_2), \ldots, d_k(x_k)]$ associated with nodes $n_1, n_2, \ldots, n_k$ such that there exists a subpath

$p \cdot (n)$, called an *ordered context subpath for* $ODC(n)$, with the following property: for all i, $1 \leq i \leq k$, $p = p_i \cdot (n_i) \cdot q_i$, where $d_i(x_i)$ occurs at n_i and q_i is definition-clear wrt x_i and for all j, $i < j \leq k$, either $n_i = n_j$ or n_j occurs along q_i. Thus, an ordered definition context of a node is a sequence of definitions that occur along the same subpath and that reach uses at that node via the subpath. The order of the definitions in the sequence is the same as their order along the subpath. An ordered context subpath for an ordered definition context is a subpath along which the ordered definition context occurs.

Again, let n be a node in a control flow graph $G(M)$, and let $\{x_1, x_2, \ldots, x_k\}$ be a nonempty subset of $USED(n)$. A *definition context* of node n is a set of definitions $DC(n) = \{d_1(x_1), d_2(x_2), \ldots, d_k(x_k)\}$ associated with nodes $n_1, n_2, \ldots, n_k$ such that there exists a subpath $p \cdot (n)$, called a *context subpath for* $DC(n)$, with the following property: for all i, $1 \leq i \leq k$, $p = p_i \cdot (n_i) \cdot q_i$, where $d_i(x_i)$ occurs at n_i and q_i is definition-clear wrt x_i. Thus, a definition context of a node is a set of definitions of variables used at the node that reach the node along the same initial subpath. A context subpath for a definition context is a subpath along which the definition context occurs. We will later show that for any definition context $DC(n)$, there is at least one ordered definition context $ODC(n)$ whose definitions are exactly the elements of $DC(n)$, and thus any ordered context subpath for $ODC(n)$ is a context subpath for $DC(n)$.

The Context Coverage criterion requires that a path set cover every definition context in a module; the Ordered Context Coverage criterion requires that every ordered definition context be covered. The Context Coverage and Ordered Context Coverage criteria defined here differ somewhat from those originally defined by Laski and Korel, who require a definition context or ordered definition context of a node to include definitions of all variables used at the node, instead of any subset. Thus the criteria we define require paths to a statement even when there is no path that defines all the variables used at the statement — a situation that might legitimately occur, for example, if a called procedure references some of its parameters conditionally. We now formally define the Context Coverage and Ordered Context Coverage criteria:

> *The pair (M, P) satisfies the* **Context Coverage** *criterion iff for all nodes n and for all definition contexts $DC(n)$, P contains at least one context subpath for $DC(n)$.*
> *The pair (M, P) satisfies the* **Ordered Context Coverage** *criterion iff for all nodes n and for all ordered definition contexts $ODC(n)$, P contains at least one ordered context subpath for $ODC(n)$.*

4. ANALYSIS OF THE CRITERIA

4.1 Evaluating the Rapps and Weyuker Hierarchy

The Rapps and Weyuker path selection criteria defined in Section 3 are presented in [**?**, **?**, **?**, **?**, **?**]. In these papers, Rapps and Weyuker propose the subsumption relationships between their criteria illustrated by graph of Figure 1. It is interesting to note that the definition of the All-DU-Paths criterion presented in [**?**, **?**] differs from the earlier definition in [**?**, **?**, **?**]. We refer to the earlier version of the criterion as All-DU-Paths$'$.

> *The pair (M, P) satisfies the* **All-DU-Paths$'$** *criterion iff for all definitions $d_m(x)$, all uses $u_n(x)$, and all successor nodes n' of n, P contains every subpath $(m) \cdot p \cdot (n, n')$ such that $(m) \cdot p \cdot (n)$ is cycle-free and p is definition-clear wrt x.*

The only difference between the two criteria is that All-DU-Paths$'$ requires only cycle-free subpaths while All-DU-Paths requires simple cycles as well.

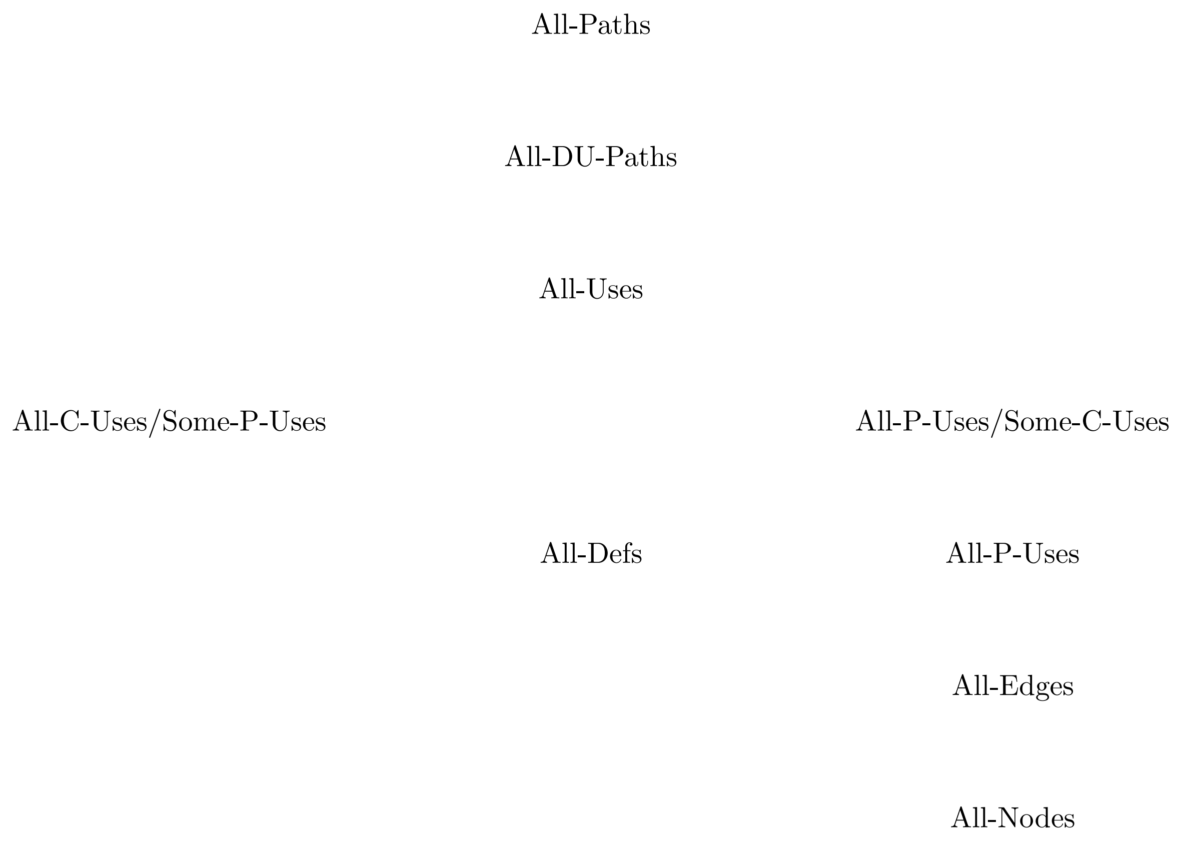

Figure 1: The Rapps and Weyuker Subsumption Hierarchy.

```
n1      input (x, y);
        loop
n2        y := y + 1;
n3        exit when y > 0;
n4        x := x + 2;
n5      end loop;
n6      output (x, y);
```

Figure 2: Module M_1 and its control flow graph $G(M_1)$.

Rapps and Weyuker provide no motivation for this change, but it is, in fact, important to the position of the All-DU-Paths criterion in the subsumption hierarchy. Module M_1 in Figure 2 illustrates that the (original) All-DU-Paths$'$ criterion does not subsume the All-Uses criterion. The pair (M_1, P_1) satifies All-DU-Paths$'$, where

$$P_1 = \{(n_{start}, n_1, n_2, n_3, n_4, n_5, n_2, n_3, n_6, n_{final}), (n_{start}, n_1, n_2, n_3, n_6, n_{final})\}. \tag{1}$$

However, (M_1, P_1) does not satisfy All-Uses, because P_1 does not cover all uses of $d_4(x)$. In particular, P_1 contains no definition-clear subpath wrt x from the definition $d_4(x)$ to the use $u_4(x)$. In fact, All-DU-Paths$'$ does not even subsume All-Defs. If $d_4(x)$ reached no uses of x other than $u_4(x)$ (if, for example, x was not used at n_6), then the prohibition against cycles in All-DU-Paths$'$ would mean that no subpath from this definition to any use would be required. Rapps and Weyuker solved this problem by requiring the selection of simple cycles as well as cycle-free subpaths. Because Rapps and Weyuker do not offer a proof that All-DU-Paths strictly subsumes All-Uses, we prove it here. First, however, we must prove a graph-theoretic lemma.

Lemma 1 *Let $(m) \cdot p \cdot (n)$ be a subpath in a control flow graph $G(M)$, such that there is a definition $d_m(x)$ and p is definition-clear wrt x. Then there exists a subpath $r = (m) \cdot p' \cdot (n)$ in $G(M)$ such that r is cycle-free or is a simple cycle, and p' is definition-clear wrt x.*

Proof. Since p is definition-clear wrt x, m does not occur along p, and if n occurs along p then n has a first occurrence. Thus, there is a subpath $(m) \cdot q \cdot (n)$ in $G(M)$ such that neither m nor n occurs along q and q is definition-clear wrt x. If q is cycle-free then we may let $r = (m) \cdot q \cdot (n)$.

Suppose, however, that q contains one or more cycles. It is a well-known result that if there exists a subpath $(m) \cdot q \cdot (n)$ in a graph such that q is not cycle-free, then there exists a subpath $(m) \cdot q' \cdot (n)$ in the graph, where q' is cycle-free and is obtained by deletion of nodes from q. Since no definitions have been added to q', q' is definition clear wrt x. Thus, we may let $r = (m) \cdot q' \cdot (n)$. $\square$

Theorem 1 *The All-DU-Paths criterion strictly subsumes the All-Uses criterion.*

Proof. We first prove that the All-DU-Paths criterion subsumes the All-Uses criterion. It must be shown that any pair that satisfies All-DU-Paths also satisfies All-Uses. Suppose the pair (M, P) satisfies All-DU-Paths. Let $d_m(x)$ be a definition, let $u_n(x)$ be a use reached by $d_m(x)$, and let n' be a successor of node n. To prove that (M, P) satisfies All-Uses, it is sufficient to show that P must contain at least one subpath of the form

$$(m) \cdot p \cdot (n, n'), \qquad (2)$$

where p is definition-clear wrt x. Since $d_m(x)$ reaches $u_n(x)$, there is a subpath $(m) \cdot q \cdot (n, n')$ in $G(M)$ such that q is definition-clear wrt x. Thus, by Lemma 1, there is a subpath $r = (m) \cdot q' \cdot (n, n')$ in $G(M)$ such that $(m) \cdot q' \cdot (n)$ is cycle-free or is a simple cycle and q' is definition-clear wrt x. Since (M, P) satisfies All-DU-Paths, P must contain r. But r is of the form shown in (2), and so (M, P) satisfies All-Uses. Since (M, P) was any pair satisfying All-DU-Paths, that criterion subsumes All-Uses.

We now show that All-Uses does not subsume All-DU-Paths. Consider the module M_2 and its control flow graph $G(M_2)$, both shown in Figure 3. The pair (M_2, P_2) satisfies All-Uses, where

$$P_2 = \{(n_{start}, n_1, n_2, n_3, n_4, n_5, n_6, n_{final}), (n_{start}, n_1, n_2, n_4, n_6, n_{final})\}. \qquad (3)$$

It does not satisfy All-DU-Paths, however, because P does not contain all subpaths of the form $(n_1) \cdot p \cdot (n_6, n_{final})$, where p is definition-clear wrt y. In particular, P does not contain the subpath $(n_1, n_2, n_3, n_4, n_6, n_{final})$. Thus, All-Uses does not subsume All-DU-Paths. $\square$

```
n1      input (x, y);
n2      if x < 0 then
n3        x := 1;
        end if;
n4      if y > 0 then
n5        y := 0;
        end if;
n6      output (x, y);
```

Figure 3: Module M_2 and its control flow graph $G(M_2)$.

4.2 Incorporating Ntafos's Required k-Tuples Criteria

In this section, we compare Ntafos's Required k-Tuples criteria to the Rapps and Weyuker criteria. The All-Paths criterion obviously subsumes each of the Required k-Tuples criteria. None of the Required k-Tuples criteria subsume the All-Defs criterion, because the Required k-Tuples criteria do not require that a variable definition be covered if its only use is at the node where the definition occurs. The All-DU-Paths criterion does not subsume any of the Required k-Tuples criteria, because All-DU-Paths does not require each loop containing a definition or use to be tested with at least two cl-subpaths as the Required k-Tuples criteria do. These last two facts imply that the Required k-Tuples criteria are incomparable to all the criteria that are subsumed by All-DU-Paths and that subsume All-Defs. Because the Required k-Tuples criteria require that both edges from a branch predicate be covered, they do subsume the All-P-Uses criterion. We now formally state and prove each of these relationships.

```
          -- initial, blank, and cr are constants.
$n_1$     state := initial;
          repeat
$n_2$       input (char);
$n_3$       if char $\notin$ {blank, cr} then
              --The procedure fsa takes char
              --and state as inputs and yields
              --state and accept as outputs.
$n_4$         fsa (state, char, accept);
            end if;
$n_5$     until char = cr;
$n_6$     output (accept);
```

Figure 4: Module M_3 and its control flow graph $G(M_3)$.

Theorem 2 *There is no Required k-Tuples criterion that subsumes the All-Defs criterion.*

Proof. Consider the module M_3 and its control flow graph $G(M_3)$, both shown in Figure 4. The 2-*dr* interactions associated with $G(M_3)$ are as follows:

$$\begin{array}{ll}
[d_{in}(accept), u_6(accept)], & [d_{in}(accept), u_{out}(accept)], \\
[d_1(state), u_4(state)], & [d_1(state), u_{out}(state)], \\
[d_2(char), u_3(char)], & [d_2(char), u_4(char)], \\
[d_2(char), u_5(char)], & [d_2(char), u_{out}(char)], \\
[d_4(state), u_{out}(state)], & \\
[d_4(accept), u_6(accept)], & [d_4(accept), u_{out}(accept)].
\end{array}$$

The 3-*dr* interactions associated with $G(M_3)$ are:

$$\begin{array}{l}
[d_1(state), u_4(state), d_4(state), u_{out}(state)], \\
[d_1(state), u_4(state), d_4(accept), u_6(accept)],
\end{array}$$

$[d_1(state), u_4(state), d_4(accept), u_{out}(accept)]$,

$[d_2(char), u_4(char), d_4(state), u_{out}(state)]$,
$[d_2(char), u_4(char), d_4(accept), u_6(accept)]$,
$[d_2(char), u_4(char), d_4(accept), u_{out}(accept)]$.

There are no k-*dr* interactions associated with $G(M_3)$ for $k > 3$. The pair (M_3, P) satisfies each Required k-Tuples criterion, where $P = \{p_1, p_2, p_3\}$ and

$$p_1 = (n_{start}, n_{in}, n_1, n_2, n_3, n_4, n_5, n_2, n_3, n_5, n_6, n_{out}, n_{final}),$$
$$p_2 = (n_{start}, n_{in}, n_1, n_2, n_3, n_4, n_5, n_6, n_{out}, n_{final}),$$
$$p_3 = (n_{start}, n_{in}, n_1, n_2, n_3, n_5, n_6, n_{out}, n_{final}).$$

This pair contains interaction subpaths for the 2-*dr* and 3-*dr* interactions associated with $G(M_3)$ and contains the additional subpaths required because some of these interactions begin or end in loops and because some end in branch predicates. Although the Required k-Tuples criteria artificially introduce a use of *state* at n_{out}, for All-Defs the only use of $d_4(state)$ is at n_4. Thus, (M_3, P) does not satisfy the All-Defs criterion, because P does not contain a subpath $(n_4) \cdot p \cdot (n_4)$ such that p is definition-clear wrt the variable *state*. □

Corollary 1 *The All-Paths criterion strictly subsumes each of the Required k-Tuples criteria.*

Theorem 3 *The All-DU-Paths criterion does not subsume the Required 2-Tuples criterion.*

Proof. Consider the module M_4 and its control flow graph, both shown in Figure 5. The pair (M_4, P) satisfies the All-DU-Paths criterion, where

$$P = \{(n_{start}, n_1, n_2, n_3, n_2, n_3, n_4, n_{final})\}.$$

It does not satisfy the Required 2-Tuples criterion, however, because there is no subpath $p_1 \cdot p_2 \cdot (n_2) \cdot p_3$ in P such that $p_1 \cdot p_2 \cdot (n_2)$ ends with an interaction subpath for the 2-*dr* interaction $[d_1(x), u_2(x)]$, and $p_2 \cdot (n_2) \cdot p_3$ is a cl-subpath for the loop in $G(M_4)$ that traverses it a minimal number of times (in this case once). □

Corollary 2 *Each Required k-Tuples criterion is incomparable to the the All-DU-Paths criterion, the All-Uses criterion, the All-C-Uses/Some-P-Uses criterion, the All-P-Uses/Some-C-Uses criterion, and the All-Defs criterion.*

Theorem 4 *Each Required k-Tuples criterion subsumes the All-P-Uses criterion.*

Proof. It must be shown that for each Required k-Tuples criterion, if a pair satisfies that criterion then it also satisfies the All-P-Uses criterion. Suppose the pair (M, P)

```
n1      input (x);
        repeat
n2        x := x + 1;
n3      until x > 0;
n4      output (x);
```

Figure 5: Module M_4 and its control flow graph $G(M_4)$.

satisfies the Required k-Tuples criterion, where $k \geq 2$. Let $u_n(x)$ be a predicate use reached by a definition $d_m(x)$, and let n' be a successor of node n. To prove that (M, P) satisfies All-P-Uses, it is sufficient to show that P must contain at least one subpath of the form

$$(m) \cdot p \cdot (n, n'), \tag{4}$$

where p is definition-clear wrt x. Note that $m \neq n$, because a definition and a predicate use can not be associated with the same node. Thus $\lambda = [d_m(x), u_n(x)]$ is a 2-*dr* interaction in $G(M)$. Since (M, P) satisfies the Required k-Tuples criterion, P must contain a subpath $q \cdot (n')$, where q is an interaction subpath for λ. By definition, $q = (m) \cdot q' \cdot (n)$, where q' is definition-clear wrt x. But then $q \cdot (n')$ is of the form shown in (4), and so (M, P) satisfies All-P-Uses. Since (M, P) was any pair satisfying the Required k-Tuples criterion, that criterion subsumes All-P-Uses. Because k was any integer greater than or equal to 2, each Required k-Tuples criterion subsumes All-P-Uses. □

Corollary 3 *Each Required k-Tuples criterion strictly subsumes the All-P-Uses criterion, the All-Edges criterion, and the All-Nodes criterion.*

4.3 Incorporating the Laski and Korel Criteria

In this section, we demonstrate the subsumption relationships that exist between the Laski and Korel criteria and those of Rapps and Weyuker and of Ntafos. We first show that Laski and Korel's criteria form a hierarchy. The Ordered Context Coverage criterion subsumes the Context Coverage criterion because all ordered context subpaths for an ordered definition context $ODC(n)$ are context subpaths for the definition context containing the same definitions as $ODC(n)$. The subsumption is strict because a context subpath for a definition context $DC(n)$ is not necessarily an ordered context subpath for all the ordered definition contexts containing the same definitions as $DC(n)$. The Context Coverage criterion subsumes the Reach Coverage criterion because every definition reaching a use at a node must appear in some definition context of that node [6] . This subsumption is strict because the Reach Coverage criterion does not require paths exercising combinations of definitions as the Context Coverage criterion does. We now formally state and prove these relationships.

Theorem 5 *The Context Coverage criterion strictly subsumes the Reach Coverage criterion.*

Proof. We first prove that the Context Coverage criterion subsumes the Reach Coverage criterion. Suppose the pair (M, P) satisfies Context Coverage. Let $d_m(x)$ be a definition and let $u_n(x)$ be a use reached by $d_m(x)$. To prove that (M, P) satisfies Reach Coverage, it is sufficient to show that P must contain at least one subpath of the form

$$(m) \cdot p \cdot (n), \tag{5}$$

where p is definition-clear wrt x. Since $d_m(x)$ reaches $u_n(x)$, $DC(n) = \{d_m(x)\}$ is a definition-context of node n. Because (M, P) satisfies Context Coverage, P must contain a context subpath q for $DC(n)$. By definition, $q = q_1 \cdot (m) \cdot q_2 \cdot (n)$, where q_2 is

[6]Note that, as pointed out in Section 3.2, this is not true for Laski and Korel's original definition of a definition context.