A Comparison of Data Flow Path Selection Criteria [1]

Lori A. Clarke, Andy Podgurski, Debra J. Richardson, Steven J. Zeil

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

Abstract

A number of path selection testing 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 for those 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.

1. INTRODUCTION

Ever since Stucki experimentally showed that programmers select test data that provide very poor coverage of their code [Stuc73], researchers have been concerned with developing effective coverage criteria. A coverage criterion is usually satisfied by a set 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 assume that a reasonable test data selection criterion exists and look only at the path selection problem.

Testing all the paths in a program is often impossible, because programs with loops may contain an infinite number of paths. Thus, a path selection criterion should specify only a 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. Other factors, such as loop coverage and data relationships, should also be considered. A number of path selection criteria have been proposed [Lask83,Ntaf84,Rapp85]. 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.

In this paper we formally compare data flow path selection criteria [Lask83,Ntaf84,Rapp85]. To facilitate this comparison, we define all the criteria using a single set of terms, rather than using the terminology of the criteria's originators. Although our definitions are usually equivalent in meaning to those given by the criteria's originators, some are not. This occurs for two reasons. First, some of the original definitions are ambiguous. Second, as originally defined, some of the criteria 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 originators.

The next section of this paper presents our terminology. Section 3 defines the criteria 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. The conclusion discusses issues that must be considered in order to evaluate these criteria more meaningfully. 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. We assume that E contains no edges of the form (n, n). 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.

The 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 $p = (n_1, n_2, \ldots, n_{|p|})$ [2] 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}.

[1] This work was supported in part by NSF Grants MCS-8303320 and DCR-8404217.

[2] We denote the *length* of (the number of elements in) a sequence s by $|s|$.

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 the subgraph of $G(M)$ corresponding 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 that begins and ends with the same node and that contains at least two edges. 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*. Subpaths through loops need not contain cycles. 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 combination of cycles and traversals encountered during a complete execution of a loop. A *complete loop-subpath* or *cl-subpath* for a loop L is a subpath $(m)\cdot p\cdot(n)$ such that m and n occur outside L, while p is a nonempty subpath lying entirely within L.

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 $DEFINED(n)$. 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 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 simply 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 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 definitions. Since anomalies like these are normally considered to be errors, and since they are easily detectable via 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.

[3] When nodes are subscripted, as in n_i, we abbreviate the notation to $d_i(x)$ and $u_i(x)$.

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, 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_{final}).

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 $PATHS(M)$. A pair (M,P) *satisfies* a criterion C iff $C(M,P) = true$. 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* if 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);
2. *Every control flow graph is well-formed;*
3. *Every control flow graph contains at least one definition;*
4. *Every definition reaches at least one use;*
5. *Every use is reached by at least one definition;*
6. *At least one use is associated with each node representing a predicate;*
7. *No definitions are associated with a node representing a predicate.*

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 [Rapp82,Rapp85,Weyu84]. This family includes three well-established 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 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. 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.

Rapps and Weyuker first define a criterion that requires a path set to contain at least one definition-clear subpath from a 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 $u_n(x)$ associated with node n.*

Next, Rapps and Weyuker define a criterion that requires a path set to 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 nodes is that they force all branches to be taken following a predicate.

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 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 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 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 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 $u_n(x)$, 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 cycle-free and p is definition-clear wrt x.*

3.2 Ntafos's 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 [Ntaf81,Ntaf84]. These criteria require that a path set cover chains of alternating definitions and uses, called *k-dr interactions*. The ith definition of a *k-dr* interaction reaches the ith use, which occurs at the same node as the $(i+1)$st definition. 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 $k \geq 2$. A *2-dr* interaction is simply a pair $[d_m(x), u_n(x)]$ such that $d_m(x)$ reaches $u_n(x)$ and $m \neq n$. An *interaction subpath* for this *2-dr* interaction is a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x. For $k \geq 3$, a *k-dr* interaction is a sequence $\kappa = [d_1(x_1), u_2(x_1), d_2(x_2), \ldots, d_{k-1}(x_{k-1}), u_k(x_{k-1})]$ 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 the variables $x_1, x_2, \ldots, x_{k-1}$ need not be distinct. An interaction subpath for κ is a subpath $p = (n_1) \cdot p_1 \cdot (n_2) \cdot p_2 \cdot \cdots \cdot p_{k-1} \cdot (n_k)$ such that for all i, $1 \leq i < k$, subpath p_i is definition-clear wrt x_i.

As defined by Ntafos, each Required k-Tuples criterion requires only that a path set contain at least one interaction subpath for every *k-dr* interaction in a module's control flow graph, and some additional subpaths if the first definition or last use of a *k-dr* interaction occurs in a loop or if the last use is a predicate use. This means that the Required k-Tuples criterion does not necessarily subsume the Required $(k-1)$-Tuples criterion for a fixed $k > 2$, since for any module there exists a constant n such that there are no *k-dr* interactions for $k > n$. 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 $k > 2$. Our definition of the criteria assures this.

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. This is apparently done to detect data flow anomalies. 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 [4] 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.*

3.3 The Laski and Korel Criteria

Laski and Korel define three path selection criteria based on data flow analysis [Lask83]. We refer to these as the Reach Coverage criterion (Strategy I), the Context Coverage criterion (Strategy II), and the Ordered Context Coverage criterion (modified Strategy II).

The Reach Coverage criterion was originally defined by Herman [Herm76]. 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 $ODC(n) = [d_1(x_1), d_2(x_2), \ldots, d_k(x_k)]$ for which there exists a subpath $p \cdot (n)$, called an *ordered context subpath*, with the following property: for all i, $1 \leq i \leq k$, $p = p_i \cdot (n_i) \cdot q_i$, where 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 the node. The order of the definitions in the sequence is the same as their order along the subpath.

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 a node n is a set of definitions $DC(n) = \{d_1(x_1), d_2(x_2), \ldots, d_k(x_k)\}$ for which there exists a subpath $p \cdot (n)$, called a *context subpath*, with the following property: for all i, $1 \leq i \leq k$, $p = p_i \cdot (n_i) \cdot q_i$, where 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, which reach the node along some initial subpath. Note that for any node n, a definition context $DC(n)$ is the set of definitions in at least one sequence $ODC(n)$, and an ordered context subpath for any such $ODC(n)$ is a context subpath for $DC(n)$.

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 just a 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, in a call to a procedure that 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 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 ordered definition contexts $ODC(n)$, P contains at least one ordered context subpath for $ODC(n)$.*

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

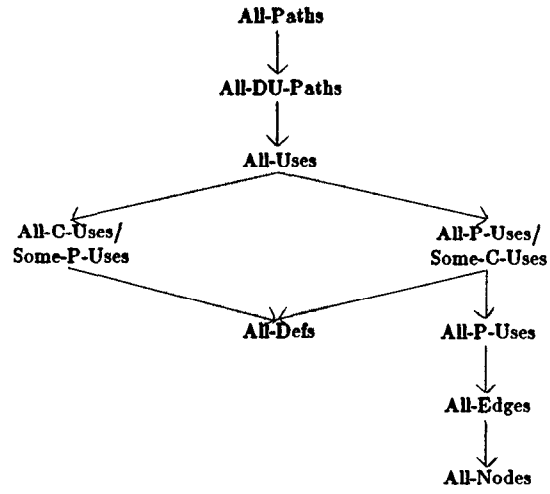

Figure 1: The Rapps and Weyuker Subsumption Hierarchy.

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 those presented in [Rapp85]. In that paper, Rapps and Weyuker propose a partial ordering of their criteria, as illustrated in the subsumption graph of Figure 1. It is interesting to note that the definition of the All-DU-Paths criterion presented in [Rapp85] differs from the earlier definitions in [Rapp82,Weyu84]. The earlier definitions of All-DU-Paths required only cycle-free subpaths, while the newer definition requires simple cycles as well. Without this change, it can be shown that All-DU-Paths does not even subsume All-Defs [Clar85A].

In order to demonstrate that the position of the newer version of the All-DU-Paths criterion in the subsumption hierarchy of Figure 1 is correct, we prove that All-DU-Paths strictly subsumes the All-Uses criterion.

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

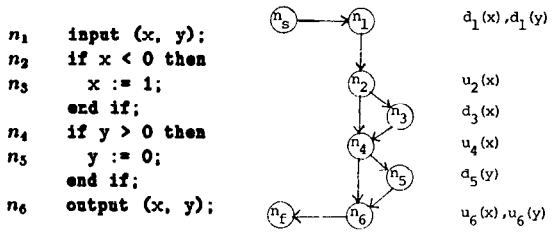

Figure 2: Module M_1 and its control flow graph $G(M_1)$.

Proof. We first prove that All-DU-Paths subsumes All-Uses, by showing that any pair not satifying the All-Uses criterion cannot satisfy the All-DU-Paths criterion either. Let (M, P) be a pair not satisfying the All-Uses criterion. Then there exists a definition $d_m(x)$, a use $u_n(x)$ reached by $d_m(x)$, and a successor n' of node n such that P contains no subpath of the form $(m) \cdot p \cdot (n, n')$, where p is definition-clear wrt x. Assume, by way of contradiction, that (M, P) satisfies the All-DU-Paths criterion. Because $d_m(x)$ reaches $u_n(x)$, there exists a subpath $(m) \cdot p \cdot (n, n')$ in $G(M)$ such that p is definition-clear wrt x. It follows [Clar85A] that $G(M)$ also contains a subpath $q = (m) \cdot p' \cdot (n, n')$ such that $(m) \cdot p' \cdot (n)$ is cycle-free or is a simple cycle, and p' is definition-clear wrt x. Because (M, P) satisfies All-DU-Paths, P must contain q. But this is a contradiction, and we must conclude that (M, P) cannot satisfy All-DU-Paths. Thus, All-DU-Paths subsumes All-Uses.

We now show that All-Uses does not subsume All-DU-Paths. Consider the module M_1 shown in Figure 2. The pair (M_1, P_1) satisfies All-Uses, where

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

It does not satisfy All-DU-Paths, however, because 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. □

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.

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

Proof. Consider the module M_2 shown in Figure 3. The procedure **fsa** called in M_2 implements a finite state automaton.

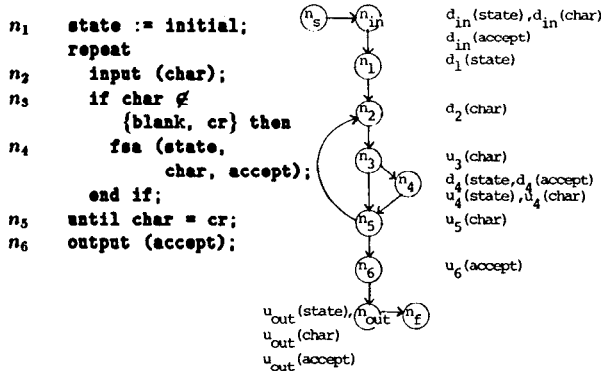

Figure 3: Module M_2 and its control flow graph $G(M_2)$.

It inputs **state** and **char** and outputs **state** and **accept**. The 2-*dr* interactions and 3-*dr* interactions associated with $G(M_2)$ are as follows:

$[d_{in}(accept), u_6(accept)]$, $[d_{in}(accept), u_{out}(accept)]$,
$[d_1(state), u_4(state)]$, $[d_1(state), u_{out}(state)]$,
$[d_2(char), u_3(char)]$, $[d_2(char), u_4(char)]$,
$[d_2(char), u_5(char)]$, $[d_2(char), u_{out}(char)]$,
$[d_4(state), u_{out}(state)]$,
$[d_4(accept), u_6(accept)]$, $[d_4(accept), u_{out}(accept)]$,
$[d_1(state), u_4(state), d_4(state), u_{out}(state)]$,
$[d_1(state), u_4(state), d_4(accept), u_6(accept)]$,
$[d_1(state), u_4(state), d_4(accept), u_{out}(accept)]$,
$[d_2(char), u_4(char), d_4(state), u_{out}(state)]$,
$[d_2(char), u_4(char), d_4(accept), u_6(accept)]$,
$[d_2(char), u_4(char), d_4(accept), u_{out}(accept)]$.

There are no k-*dr* interactions associated with $G(M_2)$ for $k > 3$. The pair (M_2, P) satisfies each Required k-Tuples criterion, where $P = \{p_1, p_2, p_3\}$ and

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

However, (M_2, P) does not satisfy the All-Defs criterion, because P does not contain a definition-clear subpath wrt the variable *state* from the definition $d_4(state)$ to a use of *state* (there is no use u_{out} associated with $G(M_2)$ for All-Defs). □

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_3 shown in Figure 4. The pair (M_3, 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 in P that covers the 2-*dr* interaction

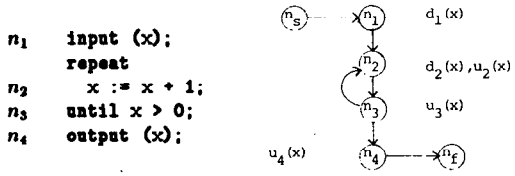

Figure 4: Module M_3 and its control flow graph $G(M_3)$.

$[d_1(x), u_2(x)]$ and contains a cl-subpath for the loop in $G(M_3)$ that traverses it a minimum 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.*

The proof of this theorem is straightforward [Clar85A] and is omitted here. □

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 any 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. [5] 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. The proof that the Context Coverage criterion subsumes the Reach Coverage criterion is straightforward [Clar85A] and is omitted here. We show that Reach Coverage does not subsume Context Coverage. Consider again the module M_1 shown in Figure 2. The pair (M_1, P_1) satisfies Reach Coverage, where P_1 is defined by Equation (1). It does not satisfy Context Coverage, however, because P_1 contains no context subpath for the definition context $DC(n_6) = \{d_1(x), d_5(y)\}$. □

Theorem 6 *The Ordered Context Coverage criterion strictly subsumes the Context Coverage criterion.*

Proof. It is easy to see that the Ordered Context Coverage criterion subsumes the Context Coverage criterion. We prove here that Context Coverage does not subsume Ordered Context Coverage. Consider the module M_4 shown in shown in Figure 5. The definition contexts associated with $G(M_4)$ are as follows:

$$DC_1(n_2) = \{d_1(z)\} \qquad DC_2(n_2) = \{d_3(z)\}$$
$$DC_1(n_3) = \{d_1(x), d_1(y)\} \qquad DC_2(n_3) = \{d_1(x), d_6(y)\}$$
$$DC_3(n_3) = \{d_5(x), d_1(y)\} \qquad DC_4(n_3) = \{d_5(x), d_6(y)\}$$

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

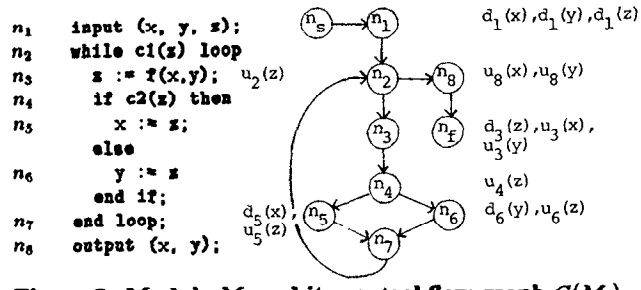

Figure 5: Module M_4 and its control flow graph $G(M_4)$.

$$DC_1(n_4) = \{d_3(z)\}$$
$$DC_1(n_5) = \{d_3(z)\}$$
$$DC_1(n_8) = \{d_3(z)\}$$
$$DC_1(n_8) = \{d_1(x), d_1(y)\} \qquad DC_2(n_8) = \{d_1(x), d_6(y)\}$$
$$DC_3(n_8) = \{d_5(x), d_1(y)\} \qquad DC_4(n_8) = \{d_5(x), d_6(y)\}.$$

The pair (M_4, P) satisfies Context Coverage, where $P = \{p_1, p_2, p_3, p_4\}$ and

$$p_1 = (n_{start}, n_1, n_2, n_3, n_4, n_5, n_7, n_2, n_3, n_4, n_5, n_7, n_2, n_8, n_{final})$$
$$p_2 = (n_{start}, n_1, n_2, n_3, n_4, n_5, n_7, n_2, n_3, n_4, n_6, n_7, n_2, n_3, n_4, n_6, n_7, n_2, n_8, n_{final})$$
$$p_3 = (n_{start}, n_1, n_2, n_3, n_4, n_6, n_7, n_2, n_3, n_4, n_6, n_7, n_2, n_8, n_{final})$$
$$p_4 = (n_{start}, n_1, n_2, n_8, n_{final}).$$

This pair does not satisfy the Ordered Context Coverage criterion, however, because P does not contain an ordered context subpath for the ordered definition context $ODC(n_8) = [d_6(y), d_5(x)]$. □

Having shown how Laski and Korel's three criteria relate to each other, we show how they relate to the other data flow criteria. The Ordered Context Coverage criterion does not subsume the All-Nodes criterion, because Ordered Context Coverage does not require that both branches following a predicate use be taken. The All-DU-Paths criterion does not subsume the Context Coverage criterion, because the presence of a loop between a definition and a node may cause all the context subpaths for a definition context of the node to contain non-simple cycles. None of the Required k-Tuples criteria subsumes Context Coverage either, because the definitions in a definition context are not necessarily linked by an interaction subpath. These three facts imply that Ordered Context Coverage and Context Coverage are incomparable to all the criteria that are subsumed by All-DU-Paths or the Required k-Tuples criteria and that subsume All-Nodes. The All-Uses criterion is similar to the Reach Coverage criterion, but strictly subsumes it because Reach Coverage does not require that all branches following a predicate use be covered as All-Uses does. Finally, Reach Coverage strictly subsumes the All-C-Uses/Some-P-Uses criterion because it requires that every use be exercised at least once. It follows from this and the fact that the All-P-Uses/Some-C-Uses criterion is incomparable to All-C-Uses/Some-P-Uses that Reach Coverage is incomparable to the criteria that are subsumed by All-P-Uses/Some-C-Use and that subsume All-Nodes.

Theorem 7 *The Ordered Context Coverage criterion does not subsume the All-Nodes criterion.*

Proof. Consider the module M_5 shown in Figure 6. The only ordered definition context associated with $G(M_5)$ is $ODC(n_2) = [d_1(x)]$. Thus the pair (M_5, P) satisfies the Ordered Context

```
n1   input (x);
n2   if x = 1 then
n3     output (1);
     else
n4     output (0);
     end if;
```

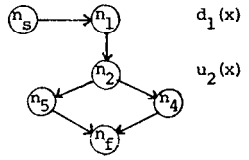

Figure 6: Module M_5 and its control flow graph $G(M_5)$.

Coverage criterion, where

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

It does not satisfy the All-Nodes criterion, however, because node n_4 does not occur along the path in P. □

Corollary 4 *The All-Paths criterion strictly subsumes the Ordered Context Coverage criterion.*

Theorem 8 *The All-DU-Paths criterion does not subsume the Context Coverage criterion.*

Proof. Consider again the module M_4 shown in Figure 5. The pair (M_4, P) satisfies the All-DU-Paths criterion, where $P = \{p_1, p_2, p_3\}$ and

$$p_1 = (n_{start}, n_1, n_2, n_3, n_4, n_5, n_7, n_2, n_3, n_4, n_6, n_7, n_2, n_8, n_{final})$$
$$p_2 = (n_{start}, n_1, n_2, n_8, n_{final}).$$

This pair does not satisfy the Context Coverage criterion, however, because P does not contain a context subpath for the definition context $DC(n_8) = \{d_1(x), d_6(z)\}$. □

Theorem 9 *There is no Required k-Tuples criterion that subsumes the Context Coverage criterion.*

Proof. Consider again the module M_1 shown in Figure 2. The pair (M_1, P_1) satisfies each Required k-Tuples criterion, where P_1 is defined by Equation (1). It does not satisfy the Context Coverage criterion, however, because P_1 contains no context subpath for the definition context $DC(n_6) = \{d_1(x), d_5(y)\}$. □

Corollary 5 *The Context Coverage and Ordered Context Coverage criteria are incomparable to the All-DU-Paths, Required k-Tuples, All-Uses, All-P-Uses/Some-C-Uses, All-P-Uses, All-Edges, and All-Nodes criteria.*

Theorem 10 *The All-Uses criterion strictly subsumes the Reach Coverage criterion.*

Proof. It is clear that the All-Uses criterion subsumes the Reach Coverage criterion. The Reach Coverage criterion cannot subsume the All-Uses criterion, because by Theorem 5 the Context Coverage criterion subsumes Reach coverage and by Corollary 5 Context Coverage does not subsume All-Uses. □

Theorem 11 *The Reach Coverage criterion strictly subsumes the All-C-Uses/Some-P-Uses criterion.*

Proof. Clearly, the Reach Coverage criterion subsumes the All-C-Uses/Some-P-Uses criterion. We prove here that the All-C-Uses/Some-P-Uses criterion does not subsume the Reach Coverage criterion. Consider the module M_6 shown in Figure 7. The pair (M_6, P) satisfies All-C-Uses/Some-P-Uses, where

$$P = \{(n_{start}, n_1, n_2, n_6, n_{final})\}.$$

```
n1   input (x);
n2   if x > 0 then
n3     if x > 1 then
n4       output (0);
       else
n5       output (1);
       end if;
     else
n6     output (2);
     end if;
```

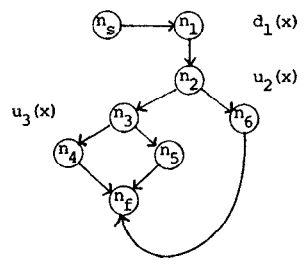

Figure 7: Module M_6 and its control flow graph $G(M_6)$.

It does not satisfy Reach Coverage, however, because P does not contain the subpath (n_1, n_2, n_3). □

Corollary 6 *The Reach Coverage criterion is incomparable to the All-P-Uses/Some-C-Uses, All-P-Uses, All-Edges, and All-Nodes criteria.*

The final subsumption hierarchy, which includes all the criteria considered, is shown in Figure 8.

5. CONCLUSION

This paper demonstrates the subsumption relationships that exist among the data flow path selection criteria proposed by Rapps and Weyuker, Ntafos, and Laski and Korel. Since these criteria have related goals, we chose them first for evaluation. Other types of path selection criteria must also be considered and their place in the subsumption hierarchy determined. Once the subsumption relationships are clearly understood, a number of important issues will still remain to be addressed. In particular, we intend to continue this investigation by considering the effect of minor enhancements to the existing criteria, the difference between the criteria in terms of their error detection capabilities, and the effect of infeasible paths as well as other troublesome features of programming languages.

Our overall goal is to formulate an effective path selection criterion. We expect that this criterion will exploit the data flow relationships used by the three families of data flow path selection criteria considered in this paper. From this study, it is clear that all three families of criteria have a unique contribution to make, although there is substantial overlap among them. Now that their relationships are better understood, we intend to continue our investigation, focusing on the differences in error detection capabilities among the criteria and on flexible guidelines for replacing infeasible paths with executable ones when appropriate.

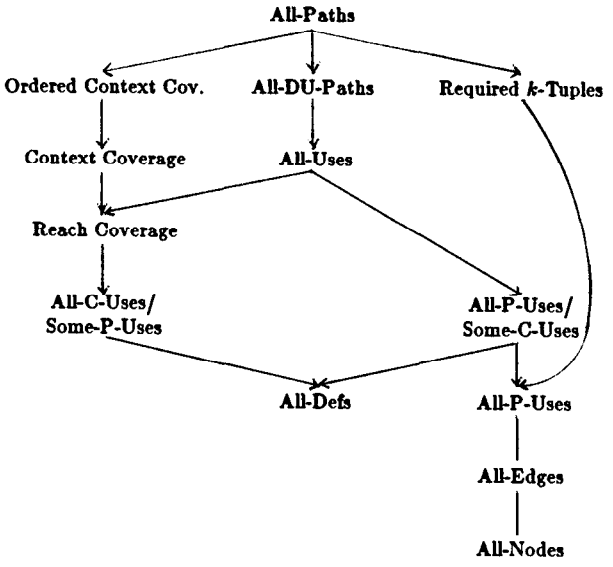

Figure 8: The Final Subsumption Hierarchy.

REFERENCES

[Clar85A] L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil, "A Comparison of Data Flow Path Selection Criteria," COINS Tech. Rep. no. 85-16, Dept. of Comp. and Information Sci., University of Mass., Amherst, June 1985.

[Herm76] P. M. Herman, "A Data Flow Analysis Approach to Program Testing," *The Australian Computer Journal*, vol. 8. no. 3, Nov. 1976.

[Lask83] J. W. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy," *IEEE Trans. on Software Eng.*, vol. SE-9, no. 3, pp. 347-354, May 1983.

[Ntaf81] S. C. Ntafos, "On Testing With Required Elements," *Proc. IEEE COMPSAC 81*, pp. 132-139, Nov. 1981.

[Ntaf84] S. C. Ntafos, "On Required Element Testing," *IEEE Trans. on Software Eng.*, vol. SE-10, no. 6, pp. 795-803, Nov. 1984.

[Rapp82] S. Rapps and E. J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection," *Proc. 6th Int. Conf. Software Eng.*, pp. 272-277, Sept. 1982.

[Rapp85] S. Rapps and E, J. Weyuker, "Selecting Software Test Data Using Data Flow Information," *IEEE Trans. on Software Eng.*, vol. SE-11, 4, pp. 367-375, April 1985.

[Stuc73] L. G. Stucki, "Automatic Generation of Self-Metric Software," *Recordings 1973 IEEE Symp. Software Reliability*, pp. 94-100, April 1973.

[Weyu84] E. J. Weyuker, "The Complexity of Data Flow Criteria for Test Data Selection," *Information Processing Letters*, vol. 19, pp. 103-109, North-Holland, August 1984.