A Formal Evaluation of Data Flow Path Selection Criteria

(Technical Report 88-25)

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

June 1988

†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 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), CCR-8704311 from the National Science Foundation with cooperation from the Defense Advanced Research Projects Agency (ARPA Order 6108), 84M103 from Control Data Corporation, and F30602-86-C-0006 from the Rome Air Development Center.

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 provide good program coverage [Stuc73]. This has led to the development of a number of coverage criteria. A path 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 cause 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 usually impossible, because programs often 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 [Howd75,Lask83,Ntaf84,Rapp85,Wood80]. 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 [Lask83,Ntaf84,Rapp85]. 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 solely based on syntactic information and do not consider semantic issues 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 (not necessarily unique) 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 and procedure call 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. 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 s by |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 $(m) \cdot p \cdot (n)$ 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 $DEFINED(n)$. 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 $u_n(x)$[2]. 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, $x \notin DEFINED(n)$. 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 [Oste76], 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, n_{out}, to the control flow graph and associate with it uses of those variables exporting information

[2]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 $u_i(x)$.

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})[3].

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* 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.

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 [Rapp81,Rapp82,Rapp85,Weyu84]. This family includes three well-known control flow criteria and some new path selection criteria based on the concepts of data flow analysis.

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

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. 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 [Ntaf81,Ntaf82,Ntaf84]. 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[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.*

Our definition for a Required k-Tuples criterion differs from that given by Ntafos in that ours require interaction subpaths for every *l-dr* interaction where $l \leq k$, while his merely requires interaction subpaths only for every *k-dr* interaction. Ntafos' Required k-Tuples criterion does not necessarily subsume his Required $(k-1)$-Tuples criterion for a fixed $k > 2$, because 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.

3.3 The Laski and Korel Criteria

Laski and Korel define three path selection criteria based on data flow analysis [Lask83]. 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

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

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 [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 $ODC(n) = [d_1(x_1), d_2(x_2), \ldots, d_k(x_k)]$ 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 $DC(n) = \{d_1(x_1), d_2(x_2), \ldots, d_k(x_k)\}$ some permutation of which is an ordered definition context of n. There is, in general, a

many–to–one relationship between the ordered definition contents of n and the definition contexts of n. A *context subpath* for a $DC(n)$ is a subpath along which $DC(n)$ occurs. Any ordered context subpath for an $ODC(n)$ is a context subpath for the corresponding $DC(n)$.

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 the statement calls 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 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

In this section we investigate the subsumption relationships between the criteria presented in Section 3. We feel that this analysis is important in understanding the differences between the criteria. Clearly no new criteria should be proposed without first demonstrating where those criteria fit into the subsumption hierarchy and the significance of those differences. The analysis is based upon a "reasonable" model of an annotated control flow graph. As noted previously, we took great

care in formulating the criteria in terms of this model so that the results of our analysis would not be biased based on our choice of model or the redefinition of the criteria in terms of that model. For the most part the model is similar to that usually assumed by those doing such analysis but the failure to explicitly define the model has led to misunderstandings or inaccurate analysis in the past. Thus, before presenting the analysis results we remind the reader of the assumptions developed in Section 2.

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;*
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}.

4.1 Evaluating the Rapps and Weyuker Hierarchy

The Rapps and Weyuker path selection criteria defined in Section 3. are presented in [Fran85], [Rapp81], [Rapp82], [Rapp85], [Weyu84]. In these papers, Rapps and Weyuker give the subsumption relationships between their criteria illustrated by graph of Figure 1. 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.

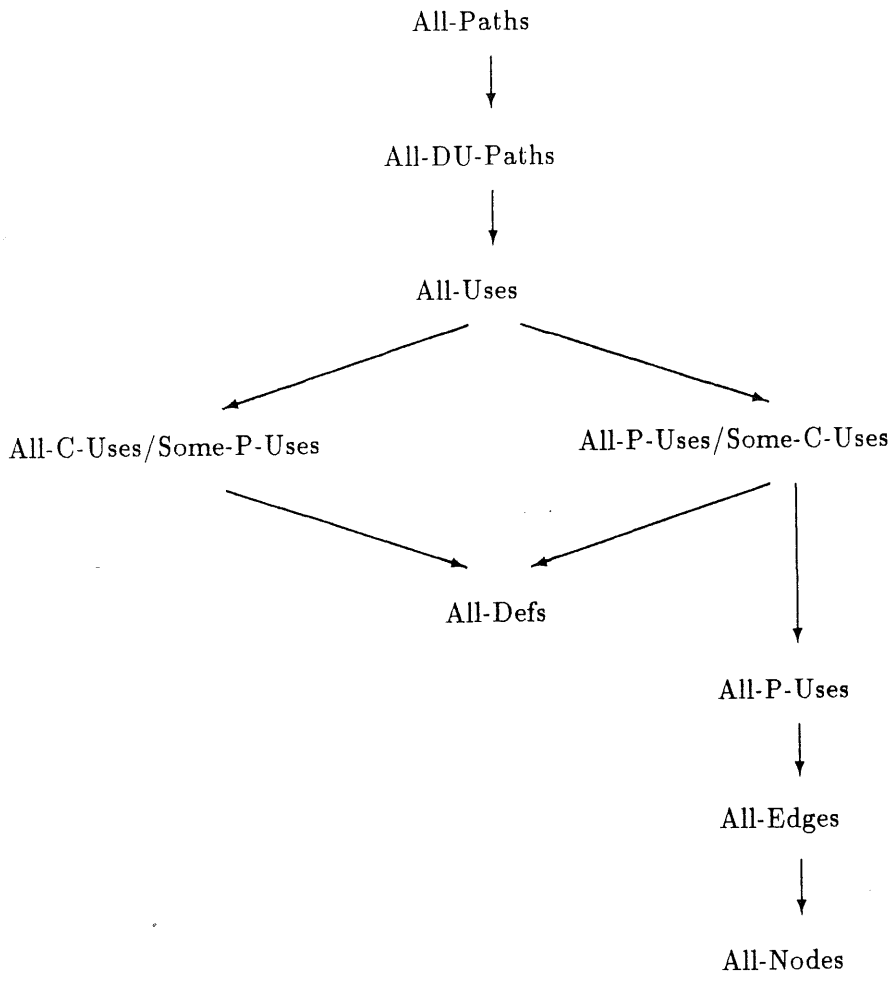

Figure 1: The Rapps and Weyuker Subsumption Hierarchy.

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 $r = (m) \cdot q' \cdot (n)$. □

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. 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

$$(m) \cdot p \cdot (n, n'), \tag{1}$$

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 (1), 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 2. The pair (M_2, P_2) satisfies All-Uses, where

$$P_2 = \{(n_{start}, n_1, n_2, n_3, n_4, n_5, n_6, n_{final}), (n_{start}, n_1, n_2, n_4, n_6, n_{final})\}. \tag{2}$$

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. □

```
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);
```

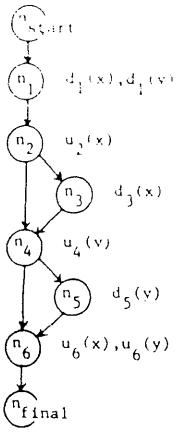

Figure 2: Module M_2 and its control flow graph $G(M_2)$.

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. First, Ntafos' criteria form a hierarchy.

Theorem 2 *Each Required k-Tuples criterion strictly subsumes the Required $(k-1)$-Tuples criterion.*

Proof. Obvious from the definition. □

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

```
        -- initial, blank, and cr are constants.
n1      state := initial;
        repeat
n2        input (char);
n3        if char ∉ {blank, cr} then
            --The procedure fsa takes char
            --and state as inputs and yields
            --state and accept as outputs.
n4          fsa (state, char, accept);
          end if;
n5      until char = cr;
n6      output (accept);
```

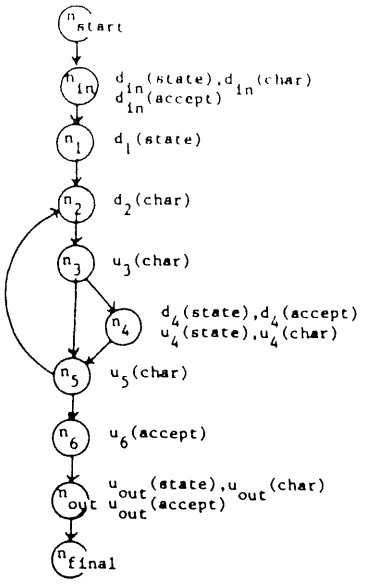

Figure 3: Module M_3 and its control flow graph $G(M_3)$.

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 3 *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 3. The 2-*dr* interactions associated with $G(M_3)$ are as follows:

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

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

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

There are no k-*dr* interactions associated with $G(M_3)$ 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 4 *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 4. The pair (M_4, P) satisfies the All-DU-Paths criterion, where

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

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

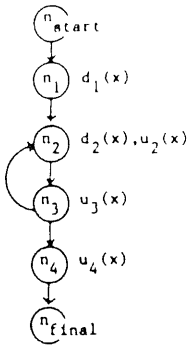

Figure 4: Module M_4 and its control flow graph $G(M_4)$.

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

Proof. Suppose the pair (M, P) 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

$$(m) \cdot p \cdot (n, n'), \tag{3}$$

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 (3), 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 those 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[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 6 *The Context Coverage criterion strictly subsumes the Reach Coverage criterion.*

[5]For reasons pointed out in Section 3.3, this is not true for Laski's and Korel's original definition of a definition context.

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

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

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 definition-clear wrt x. But then $(m) \cdot q_2 \cdot (n)$ is of the form shown in (4), and so (M, P) satisfies Reach Coverage. Thus, Context Coverage subsumes Reach Coverage.

We now show that the Reach Coverage criterion does not subsume the Context Coverage criterion. Consider again the module M_2 and its control flow graph $G(M_2)$, both shown in Figure 2. The pair (M_2, P_2) satisfies Reach Coverage, where P_2 is defined by Equation (2). It does not satisfy Context Coverage, however, because P_2 contains no context subpath for the definition context $DC(n_6) = \{d_1(x), d_5(y)\}$. □

Theorem 7 *The Ordered Context Coverage criterion strictly subsumes the Context Coverage criterion.*

Proof. We first prove that the Ordered Context Coverage criterion subsumes the Context Coverage criterion. Let (M, P) be a pair satisfying Ordered Context Coverage and let $DC(n)$ be a definition context in $G(M)$. To prove that (M, P) satisfies Context Coverage, it is sufficient to show that P must contain at least one context subpath for $DC(n)$. By definition, there is at least one ordered definition context of node n whose definitions are exactly the elements of $DC(n)$; let $ODC(n)$ be such an ordered definition context. Since (M, P) satisfies Ordered Context Coverage, P must contain an ordered context subpath p for $ODC(n)$. But then p is also a context subpath for $DC(n)$. Thus, Ordered Context Coverage subsumes Context Coverage.

We now prove that Context Coverage does not subsume Ordered Context Coverage. Consider the module M_5 and its control flow graph $G(M_5)$, both shown in Figure 5. The definition contexts associated with $G(M_5)$ are as follows:

$$\begin{array}{ll}
DC_1(n_2) = \{d_1(z)\} & DC_2(n_2) = \{d_3(z)\} \\
DC_1(n_3) = \{d_1(x), d_1(y)\} & DC_2(n_3) = \{d_1(x), d_6(y)\} \\
DC_3(n_3) = \{d_5(x), d_1(y)\} & DC_4(n_3) = \{d_5(x), d_6(y)\} \\
DC_1(n_4) = \{d_3(z)\} & \\
DC_1(n_5) = \{d_3(z)\} & \\
DC_1(n_6) = \{d_3(z)\} & \\
DC_1(n_8) = \{d_1(x), d_1(y)\} & DC_2(n_8) = \{d_1(x), d_6(y)\} \\
DC_3(n_8) = \{d_5(x), d_1(y)\} & DC_4(n_8) = \{d_5(x), d_6(y)\}.
\end{array}$$

```
n1      input (x, y, z);
n2      while z < 100 loop
n3        z := f(x,y);
n4        if z < 50 then
n5          x := z;
          else
n6          y := z
          end if;
n7      end loop;
n8      output (x, y);
```

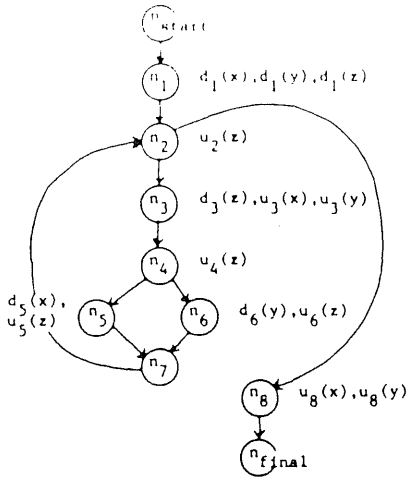

Figure 5: Module M_5 and its control flow graph $G(M_5)$.

The pair (M_5, P) satisfies Context Coverage, where

$$P = \{p_1, p_2, p_3, p_4\} \tag{5}$$

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's and Korel's three criteria relate to each other, we now 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

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

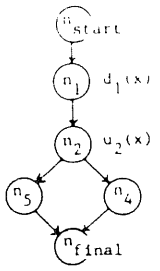

Figure 6: Module M_6 and its control flow graph $G(M_6)$.

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 the above 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-Uses and that subsume All-Nodes.

Theorem 8 *The Ordered Context Coverage criterion does not subsume the All-Nodes criterion.*

Proof. Consider the module M_6 and its control flow graph $G(M_6)$, both shown in Figure 6. The only ordered definition context associated with $G(M_6)$ is $ODC(n_2) = [d_1(x)]$. Thus, the pair (M_6, P) satisfies the Ordered Context 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 9 *The All-DU-Paths criterion does not subsume the Context Coverage criterion.*

Proof. Consider again the module M_5 and its control flow graph $G(M_5)$, both shown in Figure 5. The pair (M_5, P) satisfies the All-DU-Paths criterion, where $P = \{p_1, p_2, p_3\}$ and

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

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

Theorem 10 *There is no Required k-Tuples criterion that subsumes the Context Coverage criterion.*

Proof. Consider again the module M_2 and its control flow graph, both shown in Figure 2. The pair (M_2, P_2) satisfies each Required k-Tuples criterion, where P_2 is defined by Equation (2). It does not satisfy the Context Coverage criterion, however, because P 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 11 *The All-Uses criterion strictly subsumes the Reach Coverage criterion.*

Proof. We first prove that the All-Uses criterion subsumes the Reach Coverage criterion. Suppose the pair (M, P) satisfies All-Uses. Let $u_n(x)$ be a use reached by a definition $d_m(x)$. To prove that (M, P) satisfies Reach Coverage, it is sufficient to show that P must contain at least one subpath of the form $(m) \cdot p \cdot (n)$, where p is definition-clear wrt x. Since (M, P) satisfies All-Uses, P must contain a subpath $(m) \cdot p \cdot (n, n')$, where p is definition-clear with respect to x, for every successor n' of node n. Since every node except the final node has at least one successor and by definition the final

node has no uses, (M, P) satisfies Reach Coverage. Thus, All-Uses subsumes Reach Coverage.

Clearly the Reach Coverage criterion can not subsume the All-Uses criterion, because by Theorem 6 the Context Coverage criterion subsumes Reach coverage and by Corollary 5 Context Coverage does not subsume All-Uses. □

Theorem 12 *The Reach Coverage criterion strictly subsumes the All-C-Uses/Some-P-Uses criterion.*

Proof. We first prove that the Reach Coverage criterion subsumes the All-C-Uses/Some-P-Uses criterion. Let (M, P) be a pair satisfying Reach Coverage and let $d_m(x)$ be a definition. To prove that (M, P) satisfies All-C-Uses/Some-P-Uses, it is sufficient to show that

1. For all computation uses $u_n(x)$ reached by $d_m(x)$, P must contain 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 must contain a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x.

First, suppose $d_m(x)$ reaches a computation use $u_n(x)$. Since (M, P) satisfies Reach Coverage, P must contain a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x. On the other hand, suppose there is no computation use reached by $d_m(x)$. Because every definition must reach at least one use, there is a predicate use $u_n(x)$ reached by $d_m(x)$. Again, since (M, P) satisfies Reach Coverage, P must contain a subpath $(m) \cdot p \cdot (n)$ such that p is definition-clear wrt x. Thus Reach Coverage subsumes All-C-Uses/Some-P-Uses.

We now show that the All-C-Uses/Some-P-Uses criterion does not subsume the Reach Coverage criterion. Consider the module M_7 and its control flow graph $G(M_7)$, both shown in Figure 7. The pair (M_7, P) satisfies All-C-Uses/Some-P-Uses, where

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

It does not satisfy Reach Coverage, however, because P does not contain the subpath (n_1, n_2, n_3), along which $d_1(x)$ reaches $u_3(x)$. □

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 subsumption hierarchy including all the criteria considered is shown in Figure 8.

```
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;
```

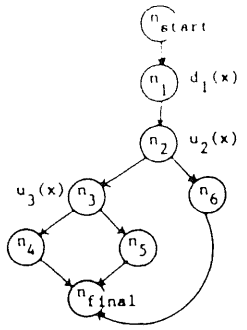

Figure 7: Module M_7 and its control flow graph $G(M_7)$.

5. Modifications to the Criteria

The subsumption hierarchy of Figure 8 shows that some of the path selection criteria we have considered fail to achieve certain minimum program coverage requirements. Specifically, the Required k-Tuples criteria do not ensure that each definition in a program is referenced at least once, and the Reach Coverage, Context Coverage, and Ordered Context Coverage criteria each fail to ensure that all statements are covered. The proofs given in the preceding section suggest that simple modifications to these criteria might correct their deficiencies. In this section, we make those modifications and determine their effect on the subsumption hierarchy.

The first modification we consider is to the Required k-Tuples criteria. By Theorem 3, there is no Required k-Tuples criterion that subsumes the All-Defs criterion. The Required 2-Tuples criterion, however, is clearly similar to the All-Uses criterion, which subsumes All-Defs. Indeed, since Required 2-Tuples calls for tests of loops that All-Uses does not, one might expect that Required 2-Tuples, and hence all the Required k-Tuples criteria, would strictly subsume All-Uses.

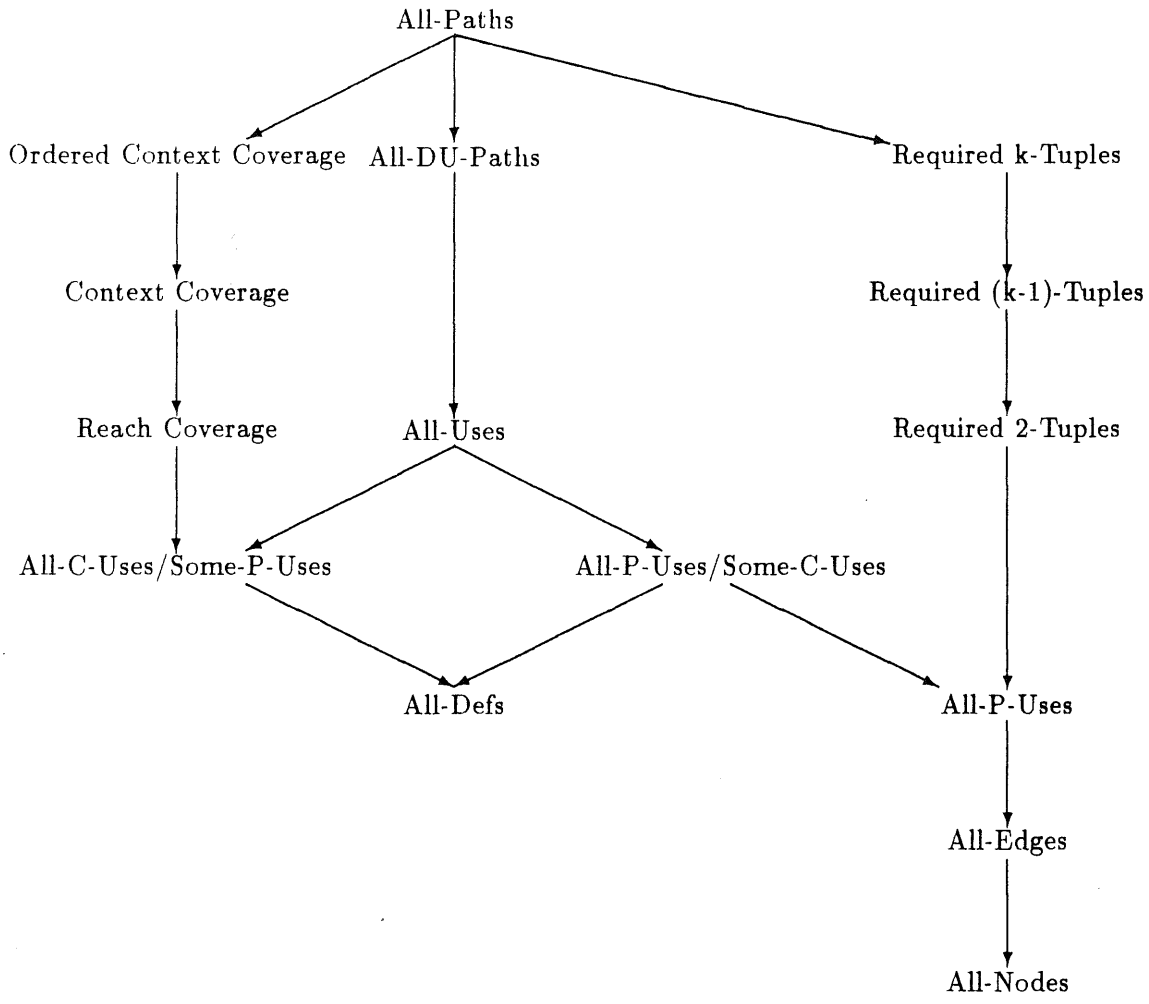

Figure 8: The New Subsumption Hierarchy.

As shown in the proof of Theorem 3, this is not true because, if the only use reached by $d_n(x)$ occurs at node n, the Required k-Tuples criteria may fail to select any definition-clear subpath wrt the variable x. The reason such a subpath may not be selected is that the nodes associated with a *k-dr* interaction must be distinct [Ntaf82,Ntaf84]; hence, there can be no *k-dr* interaction containing $d_n(x)$.

An obvious way to ensure that the Required k-Tuples criteria subsume the All-Defs criterion is to allow any definitions and uses in a *k-dr* interaction to occur at the same node. To achieve the same result, however, it is sufficient to allow the first definition and last use in a *k-dr* interaction to occur at the same node. We make the latter change, since the modified criteria then require fewer additional paths. We call a sequence of definitions and uses that satisfies this modified definition a *k-dr*$^+$ *interaction*, and we call the corresponding criteria the Required k-Tuples$^+$ criteria (we omit the formal definitions, which are obvious). We then have the following results.

Theorem 13 *Each Required k-Tuples*$^+$ *criterion strictly subsumes the corresponding Required k-Tuples criterion.*

Proof. Obvious from the definitions. □

Theorem 14 *Each Required k-Tuples*$^+$ *criterion strictly subsumes the the Required* $(k-1)$*-Tuples*$^+$ *criterion.*

Proof. Obvious from the definition. □

Theorem 15 *Each Required k-Tuples*$^+$ *criterion strictly subsumes the All-Uses criterion.*

Proof. The proof that each Required k-Tuples$^+$ criterion subsumes the All-Uses criterion is similar to the proof of Theorem 5.
We now show that the All-Uses criterion does not subsume any Required k-Tuples$^+$ criterion. If the All-Uses criterion subsumed the Required k-Tuples$^+$ criterion, where $k \geq 2$, then it would also subsume the Required k-Tuples criterion. But by Corollary 2, each Required k-Tuples criterion is incomparable to All-Uses. Thus there is no Required k-Tuples$^+$ criterion subsumed by All-Uses. □

Like the Required k-Tuples criteria, the Required k-Tuples$^+$ criteria do not in general select all cycle-free subpaths and simple cycles between a definiton-use pair in a k-dr^+ interaction. Thus, there is no Required k-Tuples$^+$ criterion that subsumes the All-DU-Paths criterion. Neither does All-DU-Paths subsume any Required k-Tuples$^+$ criterion, because of the tests of loops needed to satisfy the Required k-Tuples$^+$ criteria.

Theorem 16 *Each Required k-Tuples$^+$ criterion is incomparable to the All-DU-Paths criterion.*

> **Proof.** The All-DU-Paths criterion does not subsume any Required k-Tuples$^+$ criterion, or it would subsume the corresponding Required k-Tuples criterion, which by Corollary 2 can not be true.
> We now show that there is no Required k-Tuples$^+$ criterion that subsumes the All-DU-Paths criterion. Consider again the module M_2 and its control flow graph $G(M_2)$, shown in Figure 2. The 2-dr^+ interactions associated with $G(M_2)$ are as follows:
>
> $$\begin{array}{lll} [d_1(x), u_2(x)], & [d_1(x), u_6(x)], & [d_1(x), u_{out}(x)], \\ [d_1(y), u_4(y)], & [d_1(y), u_6(y)], & [d_1(y), u_{out}(y)], \\ [d_3(x), u_6(x)], & [d_3(x), u_{out}(x)], & \\ [d_5(y), u_6(y)], & [d_5(y), u_{out}(y)]. & \end{array}$$
>
> There are no l-dr^+ interactions associated with $G(M_2)$ for $l > 2$. Thus the pair (M_2, P_2) satisfies each Required k-Tuples$^+$ criterion, where P_2 is defined by equation (2). This pair does not satisfy All-DU-Paths, as shown in the proof of Theorem 1. □

Corollary 7 *The All-Paths criterion strictly subsumes each Required k-Tuples$^+$ criterion.*

We now consider modifications to the Laski and Korel criteria. By corollaries 5 and 6, none of these criteria subsume the All-Nodes criterion. This is because none of the Laski and Korel criteria force the coverage of all successors of a node associated with a predicate use. Therefore, we redefine the Laski and Korel criteria so that they do cover these successors and then determine the place of the modified criteria in the subsumption hierarchy. We call the modified criteria Reach Coverage$^+$, Context Coverage$^+$, and Ordered Context Coverage$^+$. Reach Coverage$^+$ is obviously equivalent to the All-Uses criterion. The definitions of the other criteria are as follows:

The pair (M, P) satisfies the **Context Coverage$^+$** *criterion iff for all nodes n in $G(M)$, all definition contexts $DC(n)$, and all successors n' of node n, P contains at least one subpath $p \cdot (n')$ such that p is a context subpath for $DC(n)$.*
The pair (M, P) satisfies the **Ordered Context Coverage$^+$** *criterion iff for all nodes n in $G(M)$, all ordered definition contexts $ODC(n)$, and all successors n' of node n, P contains at least one subpath $p \cdot (n')$ such that p is an ordered context subpath for $ODC(n)$.*

The Context Coverage$^+$ criterion and the Ordered Context Coverage$^+$ criterion clearly subsume the Context Coverage criterion and the Ordered Context Coverage criterion, respectively. Context Coverage$^+$ is not strong enough to subsume Ordered Context Coverage, however, for the same reason Context Coverage is not: there may be several ordered definition contexts corresponding to a single definition context, each of which must be covered to satisfy Ordered Context Coverage.

Theorem 17 *The Context Coverage$^+$ criterion does not subsume the Ordered Context Coverage criterion.*

Proof. Consider again the module M_5 and its control flow graph $G(M_5)$, shown in Figure 5. The pair (M_5, P) satisfies Context Coverage$^+$, where P is defined by equation (5) in the proof of Theorem 7. This pair does not satisfy Ordered Context Coverage, however, as shown in the the proof of Theorem 7. □

We now relate the modified Laski and Korel criteria to each other and to the remaining criteria. As one might expect, the change we have made to Laski's and Korel's criteria preserves their relationship to each other. This is because for every definition context of a node there is at least one ordered definition context of the same node having the same definitions.

Theorem 18 *The Ordered Context Coverage$^+$ criterion strictly subsumes the Context Coverage$^+$ criterion.*

Proof. The proof that the Ordered Context Coverage$^+$ criterion subsumes the Context Coverage$^+$ criterion is similar to the proof of Theorem 7.

If Context Coverage$^+$ subsumed Ordered Context Coverage$^+$, it would subsume Ordered Context Coverage. But by Theorem 17, this is not the case. Thus, Context Coverage$^+$ does not subsume Ordered Context Coverage$^+$.

The modified Laski and Korel criteria relate to the Rapps and Weyuker criteria differently than do the original ones, taking a more central position in the subsumption hierarchy. As desired, they subsume the All-Nodes and All-Edges criteria, because the new criteria cover all successors of a predicate use. More significantly, they subsume All-Uses, because a definition occurs in at least one definition context of each node associated with a use reached by that definition. The subsumption is strict because All-Uses does not require combinations of definitions be tested. The modified Laski and Korel criteria do not subsume the All-DU-Paths criterion, because any single ordered context subpath for a ordered definition context will suffice to cover the ordered definition context; thus all cycle-free subpaths and simple cycles between a definition and use may not be covered.

Theorem 19 *The Context Coverage$^+$ criterion strictly subsumes the Reach Coverage$^+$ (All-Uses) criterion.*

Proof. The proof is similar to the proof of Theorem 6.

Corollary 8 *The Context-Coverage$^+$ criterion is incomparable to the Ordered Context Coverage criterion.*

Theorem 20 *The Ordered Context Coverage$^+$ criterion does not subsume the All-DU-Paths criterion.*

Proof. Consider the module M_9 and its control flow graph $G(M_9)$, shown in Figure 9. The ordered definition contexts associated with $G(M_9)$ are as follows:

$$ODC_1(n_2) = [d_1(x)] \qquad ODC_2(n_6) = [d_1(y), d_1(x)]$$
$$ODC_1(n_5) = [d_1(y)]$$
$$ODC_1(n_6) = [d_1(x), d_1(y)]$$

The pair (M_9, P) satisfies the Ordered Context Coverage$^+$ criterion, where

$$P = \{(n_{start}, n_1, n_2, n_3, n_5, n_6, n_{final}), (n_{start}, n_1, n_2, n_4, n_5, n_{final})\}$$

```
n1      input (x, y);
n2      if F(x) then
n3        output (1);
        else
n4        output (0);
        end if;
n5      if y < 0 then
n6        output (x * y);
        end if;
```

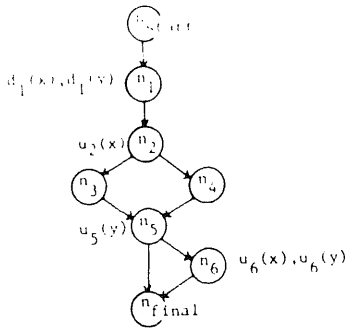

Figure 9: Module M_9 and its control flow graph $G(M_9)$.

This pair does not satisfy the All-DU-Paths criterion, 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 x. In particular, P does not contain the subpath $(n_1, n_2, n_4, n_5, n_6, n_{final})$. Thus, Ordered Context Coverage$^+$ does not subsume All-DU-Paths. □

Corollary 9 *The All-Paths criterion strictly subsumes the Ordered Context Coverage$^+$ criterion.*

Corollary 10 *The Context Coverage$^+$ criterion and the Ordered Context Coverage$^+$ criterion are both incomparable to the All-DU-Paths criterion.*

Like the original context coverage criteria, the modified Laski and Korel criteria do not subsume any Required k-Tuples criterion, because they do not incorporate the tests of loops that the Required k-Tuples criteria do.

Theorem 21 *The Ordered Context Coverage$^+$ criterion does not subsume any Required k-Tuples criterion.*

Proof. Consider again the module M_4 and its control flow graph $G(M_4)$, shown in Figure 4. The ordered definition contexts associated with $G(M_4)$ are as follows:

$$ODC_1(n_2) = [d_1(x)] \quad ODC_2(n_2) = [d_2(x)]$$
$$ODC_1(n_3) = [d_2(x)]$$
$$ODC_1(n_4) = [d_2(x)]$$

The pair (M_4, P) satisfies the Ordered Context Coverage$^+$ criterion, where

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

This pair does not satisfy any Required k-Tuples criterion, however, because P does not contain a subpath $p_1 \cdot p_2 \cdot (n_2) \cdot p_3$ 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). Thus, Ordered Context Coverage$^+$ does not subsume any Required k-Tuples criterion. □

Corollary 11 *The Context Coverage$^+$ criterion and the Ordered Context Coverage$^+$ criterion are incomparable to each Required k-Tuples criterion and to each Required k-Tuples$^+$ criterion.*

The subsumption hierarchy including the modified criteria is shown in Figure 10.

6. CONCLUSION

This paper shows the subsumption relationships that exist among the data flow path selection criteria proposed by Rapps and Weyuker, Ntafos, and Laski and Korel. These relationships are not at all obvious; in fact, we discovered errors in previous statements about them [Ntaf82,Ntaf84,Ntaf85,Ntaf88,Rapp81,Rapp82]. Understanding the relationships among these criteria points out some of their strengths and weaknesses. For example, the original Laski and Korel criteria fail to satisfy the minimum requirement of selecting paths that cover all edges and all nodes. On the other hand, Laski's and Korel's use of combinations of definitions captures certain data flow relationships not considered by the other criteria.

We consider this evaluation to be a first step toward formulating an effective path selection criterion. In this study, we have primarily considered data flow path selection criteria. Since these criteria have related goals, we chose them first for evaluation. Other, more diverse path selection criteria must also be considered and their place in the subsumption hierarchy determined. Not only

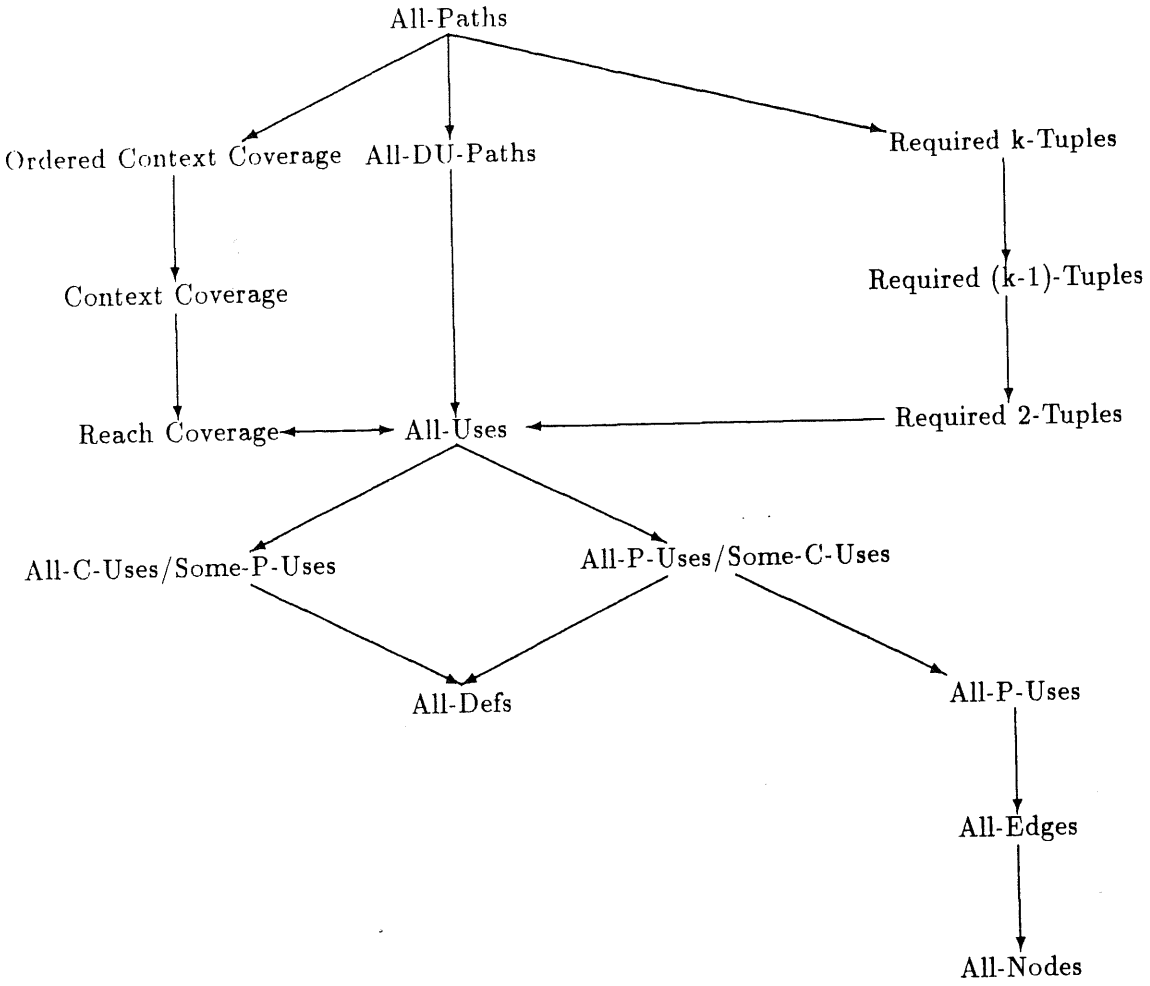

Figure 10: The Final Subsumption Hierarchy.

do the subsumption relationships need to be clearly understood, but a number of other important issues must also be addressed. In particular, we intend to consider the differences between the criteria in terms of error detection capabilities and to investigate the effect of infeasible paths.

We have shown that minor enhancements to some criteria improve their location in the subsumption hierarchy. Before considering more substantial modifications, the effects of the differences between the various criteria must be better understood. We are specifically interested in differences in the error detection capabilities of the criteria. If one criterion subsumes or is incomparable to another, then what types of errors could be revealed by executing that criterion's paths that could not be revealed by executing the paths of the other? Formulating such classes of errors may prove difficult. The error detection capabilities must be understood, however, in order to meaningfully evaluate the different criteria or to assess the value of an enhancement. For example, we could further extend the Required k-Tuples criteria to require all interaction subpaths that contain only cycle-free or simple-cycle subpaths between the nodes of a k-dr interaction. This extension would be similar to the way in which Rapps and Weyuker extended the All-Uses criterion in formulating the All-DU-Paths criterion. It is not clear, however, what additional types of errors these additional subpaths might reveal. Thus, we have decided not to propose a criterion to subsume the three highest-ranked, incomparable criteria until this issue is satisfactorily addressed.

Perhaps the biggest drawback of the use of these data flow criteria in testing is that none attempt to deal with infeasible paths, which are a common phenomena of programs [Wood80]. Because of the semantics of a program, certain paths in that program may not be executable. Figure 11 shows a module where any path that iterates the loop less than ten times is infeasible. If we use the All-DU-Paths criterion as an example, there is no executable subpath from the definition of x to

```
n1      input (x);
n2      for i in 1 .. 10 loop
n3        output (' ');
n4      end loop;
n5      output (x);
```

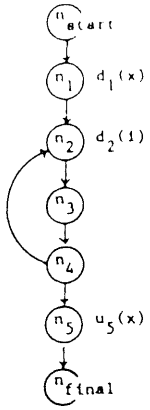

Figure 11: Module M_{10} and its control flow graph $G(M_{10})$.

its only use that is cycle free or is a simple cycle. Consequently, no executable path would be required to test that data flow relationship. In this example, the selected executable paths would not even satisfy the All-Nodes criterion. In general, the existence of a feasible path to a specific statement in a module can not be determined, and so we can not guarantee that any of these path selection criteria can be satisfied by a set of feasible paths for that module. In practice, however, we can often detect infeasible paths, either using simple static analysis or, when necessary, symbolic evaluation techniques [Clar85].

To be useful, a path selection criterion should prescribe alternative guidelines when infeasible paths would be selected. Frankl and Weyuker recognize the unsatisfiability problem that arises due to infeasible paths and circumvent it by deriving a new family of criteria from the Rapps and Weyuker family [Fran86]. These new criteria require the selection of feasible paths that cover

definition-clear subpaths between definitions and uses reached by those definitions. These criteria are always satisfiable but, in general, the question of whether or not a set of paths satisfies any one of these criteria is undecidable. Thus, although Frankl and Weyuker have moved the question of undecidability, it nonetheless remains.

Our long-term 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.

REFERENCES

[Clar85] Lori A. Clarke and Debra J. Richardson, "Applications of Symbolic Evaluation", *Journal of Systems and Software*, Vol.5, No.1, pp.15-35, January 1985.

[Fran85] Phyllis G. Frankl, Stewart N. Weiss, and Elaine J. Weyuker, "ASSET: A System to Select and Evaluate Tests", Technical Report No.148, Department of Computer Science, Courant Institute of Mathematical Sciences, January 10, 1985.

[Fran86] Phyllis G. Frankl and Elaine J. Weyuker, "Data Flow Testing in the Presence of Unexecutable Paths", *Proceedings of the Workshop on Software Testing*, pp.4–13, July 1986.

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

[Howd75] William E. Howden, "Methodology for the Generation of Program Test Data", *IEEE Transactions on Computers*, Vol.C-24, No.5, pp.554–559, May 1975.

[Lask83] Janusz W. Laski and Bogdan Korel, "A Data Flow Oriented Program Testing Strategy", *IEEE Transactions on Software Engineering*, Vol.SE-9, No.3, pp.347–354, May 1983.

[Ntaf81] Simeon C. Ntafos, "On Testing With Required Elements", *Proceedings of COMPSAC '81*, pp.132–139, November 1981.

[Ntaf82] Simeon C. Ntafos, "On Required Element Testing", Technical Report No.123, Computer Science Program, University of Texas at Dallas, November 1982.

[Ntaf84] Simeon C. Ntafos, "On Required Element Testing", *IEEE Transactions on Software Engineering*, Vol.SE-10, No.6, pp.795–803, November 1984.

[Ntaf85] Simeon C. Ntafos, "A Comparison of Some Structural Testing Strategies," Technical Report No.210, Computer Science Program, University of Texas at Dallas, June 1985.

[Ntaf88] Simeon C. Ntafos, "A Comparison of Some Structural Testing Strategies", *IEEE Transactions on Software Engineering*, Vol.14, No.6, pp.868–874, June 1988.

[Oste76] Lee J. Osterweil and Lloyd D. Fosdick, "DAVE — A Validation, Error Detection, and Documentation System for FORTRAN Programs", *Software Practice and Experience*, Vol.6, pp.473–486, 1976.

[Rapp81] Sandra Rapps and Elaine J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection", New York University Department of Computer Science Technical Report No.023, December 1981.

[Rapp82] Sandra Rapps and Elaine J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection", *Proceedings of the Sixth International Conference on Software Engineering*, pp.272–277, September 1982.

[Rapp85] Sandra Rapps and Elaine J. Weyuker, "Selecting Software Test Data Using Data Flow Information", *IEEE Transactions on Software Engineering*, SE-11, 4, pp.367–375, April 1985.

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

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

[Wood80] Martin R. Woodward, David Hedley, and Michael A. Hennel, "Experience with Path Analysis and Testing of Programs", *IEEE Transactions on Software Engineering*, Vol.SE-6, No.3, pp.278–286, May 1980.