NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
IP/O-CHAINS COVERAGE CRITERION

by

Wenxin Ma

A M. Sc. Thesis

submitted to the School of Graduate Studies and Research
of the University of Ottawa
in partial fulfillment of the requirements for the degree of

Master of Computer Science*

Department of Computer Science
University of Ottawa
Ottawa, Ontario

* The Master of Computer Science program is a joint
program with Carleton University, administrated by
the Ottawa-Carleton Institute for Computer Science.

© Wenxin Ma, Ottawa, Canada, 1995
THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT À LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE À LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

Abstract

In this thesis, three versions of the \( IP/O_2 \)-chains coverage criterion, namely the original \( IP/O_2 \)-chains coverage criterion, applicable \( IP/O_2 \)-chains coverage criterion and subdomain-based \( IP/O_2 \)-chains coverage criterion, are compared to the other control and data-flow-oriented software testing criteria under "strictly includes" and "properly covers" relations. The precise positions of these three versions of the \( IP/O_2 \)-chains coverage criterion in three hierarchies are given. Then, a new version of \( IP/O_n \)-chains coverage criterion is defined. It is proved that: (i) Applicable new \( IP/O_2 \)-chains coverage criterion strictly includes applicable all-uses criterion; (ii) For any given program \( P \), there exists a number \( n \) such that subdomain-based new \( IP/O_n \)-chains coverage criterion covers subdomain-based all-uses criterion; (iii) For any given program \( P \), there exists a number \( n \) such that for each \( IP/O_j \)-chain \( c \), if one duplicates the subdomain of \( c \) \( l(c) \) times, where \( j \leq n \) and \( l(c) \) is the length of \( c \), then subdomain-based new \( IP/O_n \)-chains coverage criterion is better than subdomain-based all-uses criterion under measure \( M \); (iv) Subdomain-based new \( IP/O_n \)-chains coverage criterion and subdomain-based required \( k \)-tuples criterion are incomparable in "universally properly covers" relation; (v) For any given program \( P \), there exists a number \( n \) such that for each \( IP/O_j \)-chain \( c \), if one duplicates the subdomain of \( c \) \( m(c) \) times, where \( j \leq n \) and \( m(c) \) is the total number of \( df \)-chains on \( c \), then subdomain-based new \( IP/O_n \)-chains coverage criterion properly covers the subdomain-based required \( k \)-tuples criterion.
Dedication

To my wife Xue Yan and my son Yang Yang
Acknowledgments

I would especially like to express my most sincere appreciation and gratitude to my research advisor, Professor Hasan Ural for his continuous support, his guidance and patience, without which this would not have been possible.

Financial assistance obtained for this research work from the Department of Computer Science, Telecommunication Research Institute of Ontario (TRIO) & Natural Sciences and Engineering Research Council of Canada (NSERC) is gratefully acknowledged.

Thanks are also due to my family for their support throughout this work.
## Table of Contents

I  Introduction  
1.1 Background ........................................................................................................... 1  
1.2 Motivation and Objectives of the Thesis .............................................................. 2  
1.3 Contributions of the Thesis .................................................................................. 4  
1.4 Outline of the Thesis ............................................................................................ 5  

II Original and Applicable *IP/O*₂-chains Coverage Criteria  
2.1 Program Model ...................................................................................................... 6  
2.2 Flowgraph Representation of a Program .............................................................. 7  
2.3 Definition of the Original *IP/O*₂-chains Coverage Criterion ............................ 10  
2.4 Hierarchy of Original Criteria with respect to Strictly Includes Relation ........... 16  
2.5 Definition of the Applicable *IP/O*₂-chains Coverage Criterion ....................... 18  
2.6 Hierarchy of Applicable Criteria with respect to Strictly Includes Relation ....... 20  

III Subdomain-based Software Testing Criteria  
3.1 Program Model ..................................................................................................... 24  
3.2 Definition of the Subdomain-based *IP/O*₂-chains Coverage Criterion .......... 24  
3.3 Measure *M* and Properly Covers Relation ......................................................... 29  
3.4 Hierarchy of Subdomain-based Criteria with respect to Universally Properly Covers Relation .......................................................................................................... 30  

IV A New Version of *IP/O*₂-chains Coverage Criterion  
4.1 Definition of New *IP/O*ₙ-chains Coverage Criterion ........................................ 41  
4.2 The Comparison of New *IP/O*ₙ-chains Coverage and All Uses Criteria ......... 57
4.3 The Comparison of New $IP/O_n$-chains Coverage and Required $k$-tuples+ Criteria 64

V Conclusion and Future Research Directions.................................71

References
List of Figures

Fig. 1a Program A ................................................................. 9
Fig. 1b Intermediate Step of Program A ................................. 9
Fig. 1c Flowgraph of Program A .............................................. 9
Fig. 2 Program B and Its Flowgraph ....................................... 15
Fig. 3 The “Strictly Includes” Hierarchy of the Original Criteria ... 17
Fig. 4 Program C and Its Flowgraph ....................................... 20
Fig. 5 The “Strictly Includes” Hierarchy of the Applicable Criteria ... 22
Fig. 6 Program D and Its Flowgraph ....................................... 31
Fig. 7 Program E and Its Flowgraph ....................................... 33
Fig. 8 The “Universally Properly Covers” Hierarchy of the Subdomain-based Criteria 39
Fig. 9 Program F and Its Flowgraph ....................................... 50
Fig. 10 Program G and Its Flowgraph .................................... 60
Fig. 11 Program H and Its Flowgraph .................................... 65
Fig. 12 Program I and Its Flowgraph .................................... 67
List of Tables

Table. 1  $P/O_j$-chains and their subdomains in Program $C$ ($j \leq 2$) ..............................................21
Table. 2  $IP/O_1$ and $IP/O_2$-chains and their subdomains in $P$ .........................................................31
Table. 3  Decisions and their subdomains in $P$ .......................................................................................31
Table. 4  $du$-pairs in Program $E$ ........................................................................................................34
Table. 5  $IP/O_1$-chains in Program $E$ ................................................................................................34
Table. 6  $IP/O_2$-chains in Program $E$ ................................................................................................34
Table. 7  Subdomains and failure-causing rates of $du$-pairs in Program $E$ .............................................35
Table. 8  Subdomains and failure-causing rates of the $IP/O_1$-chains in Program $E$ .........................36
Table. 9  Subdomains and failure-causing rates of $IP/O_2$-chains in Program $E$ ..............................37
Table. 10 Subdomains and failure-causing rates of multiple-conditions in Program $E$ .....................38
Table. 11 $IP/O_1$-chains and their subdomains in Program $F$ ...............................................................50
Table. 12 Original $IP/O_k$-chains in Program $D$ ($k \leq 2$) .................................................................52
Table. 13 Applicable $IP/O_k$-chains in Program $D$ ($k \leq 2$) ..............................................................53
Table. 14 New $IP/O_k$-chains in Program $D$ ($k \leq 2$) .................................................................54
Table. 15 New $IP/O_j$-chains in Program $H$ ($j \leq 2$) .....................................................................65
Chapter 1

Introduction

1.1 Background

Software testing is one of the most widely used methods for software quality assurance. Its general objective is to reveal the failures of a program by executing it in a controlled environment over a finite set of test cases, called a *test suite*. Each testcase is a pair of test input and corresponding expected output. During testing, a testcase is applied by supplying the test input to the program under test and by comparing the expected output to the actual output produced by the program under test.

Over the last several decades, a wide variety of strategies (called *software testing criteria*) for selecting test suites have been proposed. According to the available information about a program under test, these criteria are classified in two groups: *white box testing* and *black box testing*, where white box testing is based on the source code and the specification while black box testing is based on the specification of the program under test only.

Considering the development of white box testing, software testing criteria can be viewed as belonging to three families. Each family has a hierarchy according to a specific relation that identifies the relative merits of the criteria. The first family (henceforth called *the family of original criteria*) consists of criteria that consider only the coverage of different structural aspects of the program under test by a selected set of test paths which yields a test suite. The hierarchy of such criteria is based on the *strictly includes relation* [14]. Accordingly, a criterion A (strictly) includes a criterion B iff for any given program, any set of test paths satisfying A also satisfies B (but not vice versa).
In 1988, Frankl and Weyuker [2] argued that the family of original criteria suffers from the weakness that for programs with non-executable paths it may be impossible for any test suite to satisfy a given criterion. They modified the original criteria and defined a new family of criteria, namely the family of applicable criteria with a new hierarchy based on the strictly includes relation. In 1989, Weiss [18] argued that strictly includes relation is more useful for comparing the cost of two criteria rather than their effectiveness, where the cost of a criterion relates to the number of test paths required to satisfy the criterion while effectiveness of a criterion relates to the fault-detecting ability of the criterion. Hamlet [5] also pointed out that this relation can be “misleading” because it is possible for criterion $C_1$ to include criterion $C_2$, yet for some test suite that satisfies $C_2$ to detect a fault while some test suite that satisfies $C_1$ does not. In 1993 Frankl and Weyuker [3] proposed the measure $M$ and the properly covers relation to compare the fault-detecting abilities of criteria. Under this relation they set up a hierarchy for the family of subdomain-based criteria [4]. Like many other well-known criteria, the $IP/O_2$-chains (i.e., an ordered sequence of definition/reference pairs of variables such that the definition of the first variable is an input of the variable and the reference of the last variable is an output or a predicate) coverage criterion also has three versions, i.e. the original, the applicable, and the subdomain-based $IP/O_2$. The relative position of the original $IP/O_2$-chains coverage criterion has been identified by Ural and Yang [16]. It has been proved that the $IP/O_2$-chains coverage criterion is on the top position of the hierarchy. But, we do not know the relative position of the $IP/O_2$-chains coverage criterion in the families of applicable and subdomain-based criteria respectively.

1.2 Motivation and Objectives of the Thesis

In 1993, Frankl and Weyuker [3] proved that the fault-detecting ability of a subdomain-based criterion will be increased if its subdomains are refined. They also
proved in [4] that, the required $k$-tuples criterion (which requires each $k-dr$ interaction to be covered, where a $k-dr$ interaction is a sequence of $k-1$ definition/reference pairs of variables) has a higher fault-detecting ability when compared with other subdomain-based criteria. This result is quite natural. Because the required $k$-tuples criterion distinguishes the sequence of definitions/references of variables that occur in loops, and thus refines the subdomains of $k-dr$ interactions (i.e., the structural units on which required $k$-tuples criterion is based). It is observed that the $IP/O_n$-chains (i.e., the structural units on which $IP/O_n$-chains coverage criterion is based) actually distinguish the sequences of definitions/references of variables that occur in loops more precisely than $k-dr$ interactions because the number $n$ represents the maximal number of iterations of variables in an $IP/O_n$-chain. Increasing the number $n$ will result in the refinement of the subdomains of an $IP/O_n$-chain while increasing number $k$ will not necessarily refine the subdomains of a $k-dr$ interaction. Moreover, one of the most important disadvantages of required $k$-tuples criterion is the duplication of subdomains which makes the criterion more costly. But, the $IP/O_n$-chains coverage criterion makes less duplications of subdomains in comparing with required $k$-tuples criterion.

Given all the factors above, we believe that there must exist an improved version of the $IP/O_n$-chains coverage criterion such that its fault-detecting ability is as good as the required $k$-tuples criterion, and the duplications of its subdomains is much less than required $k$-tuples criterion. However, as the definition of required-$k$-tuples criterion introduces non-executable $k-dr$ interactions, the definition of the original $IP/O_n$-chains coverage criterion introduces non-executable $IP/O_n$-chains, i.e. the $IP/O_n$-chains that can not be covered by any executable complete path. Therefore, how we treat these non-executable $IP/O_n$-chains is important in defining the new $IP/O_n$-chains coverage
criterion. So, the objective of this thesis is to define a new version of $IP/O_n$-chains coverage criterion such that it is at the top position in both families of applicable and subdomain-based criteria without any additional overhead for handling non-executable $IP/O_n$-chains carefully.

1.3 Contributions of the Thesis

In this thesis, we have identified the $IP/O_2$-chains coverage criterion’s relative position in three different hierarchies, namely the hierarchies of original, applicable and subdomain-based software testing criteria. We defined a new version of the $IP/O_n$-chains coverage criterion and proved the following results:

1. Applicable new $IP/O_2$-chains coverage criterion strictly includes applicable all-uses criterion.

2. For any given program $P$, there exists a number $n$ such that subdomain-based new $IP/O_n$-chains coverage criterion covers subdomain-based all-uses criterion.

3. For any given program $P$, there exists a number $n$ such that for each $IP/O_j$-chain $c$, if one duplicates the subdomain of $c$ $l(c)$ times, where $j \leq n$ and $l(c)$ is the length of $c$, then subdomain-based new $IP/O_n$-chains coverage criterion is better than subdomain-based all-uses criterion under measure $M$.

4. The subdomain-based new $IP/O_n$-chains coverage criterion and subdomain-based required $k$-tuples$^+$ criterion are incomparable in universally properly covers relation.
5. For any given program $P$, there exists a number $n$ such that for each $IP/O_j$-chain $c$, if one duplicates the subdomain of $c$ $m(c)$ times, where $j \leq n$ and $m(c)$ is the total number of $df$-chains on $c$, then subdomain-based new $IP/O_n$-chains coverage criterion properly covers subdomain-based required $k$-tuples$^+$ criterion for program $P$.

1.4 Outline of the Thesis

The rest of the thesis is organized as following:

In Chapter 2, Original and Applicable $IP/O_2$-chains Coverage Criteria, we first review the definitions of original versions of software testing criteria. Then, we review the definitions of the applicable versions of software testing criteria and define the applicable $IP/O_2$-chains coverage criterion. Finally, we identify the position of applicable $IP/O_2$-chains coverage criterion in the hierarchy (Theorem 2.1) of the applicable software testing criteria.

In Chapter 3, The Subdomain-based Software Testing Criteria, we review the definitions of the subdomain-based software testing criteria, properly covers relation, measure $M$ as well as Frankl and Weyuker's results. Then, we define the subdomain-based $IP/O_2$-chains coverage criterion, and identify the position of subdomain-based $IP/O_2$-chains coverage criterion in the hierarchy (Theorem 3.2) of subdomain-based software testing criteria.

In Chapter 4, A New Version of $IP/O_2$-chains Coverage Criterion, we define a new $IP/O_n$-chains coverage criterion, and compare the new $IP/O_n$-chains coverage with the all-uses criterion (Theorem 4.1, 4.2) and the required $k$-tuples$^+$ criterion (Theorem 4.3, 4.4).
Chapter 2

Original and Applicable $IP/O_2$-chains Coverage Criteria

In this Chapter, we first review the definition of the original $IP/O_2$-chains coverage criterion and the hierarchy given in Ural and Yang's [16]. Then, we give a definition of the applicable $IP/O_2$-chains coverage criterion and prove its proper position within the hierarchy of applicable software testing criteria. The program model to be used in both cases follows the one given in Frankl and Weyuker [2].

2.1 Program Model

Without loss of generality, we assume that a program ($main$ or $subprogram$) is written in Pascal. We follow the program model given in Frankl and Weyuker [2] and assume that

a) Non-Straight-Line (NSL) Property: every program has at least one conditional or repetitive statement,

b) at least one variable occurs in every Boolean expression controlling a conditional or repetitive statement in a given program (note that this variable occurrence may be implicit, as in the use of the input file variable in the statement, while not $eof$ do $s$),

6
c) **No Feasible Anomalies (NFAUD) property:** in a given program, each feasible path from the program entry to a use of a variable \( v \) must pass through a node having a definition of \( v \) (if programs do not satisfy this property, they may have the possibility of referencing an undefined variable),

d) a given program has no goto statements, no with statements, no variant records, no functions having var parameters, no procedure or functional parameters and no conformance arrays,

e) a given program has a single entry and single exit.

### 2.2 Flowgraph Representation of a Program

For ease of applying a software testing criterion, a program is represented by a digraph \( G(V, E) \), called **flowgraph**, where \( V \) is the set of nodes corresponding to statements in the program and \( E \) is the set of directed edges indicating the possible flow of control between statements in the program (see Fig. 1a). In \( V \), there is

- a) a **statement node** (e.g. nodes 1, 3, 5, 6, and 7 in Fig. 1b) corresponding to each simple statement (e.g. input, output, assignment, or procedure statement) in the program,

- b) a **branching node** (e.g. nodes 2 and 4 in Fig. 1b) corresponding to the branching point in the transfer of control implied by the condition part of each conditional statement (e.g. if-then, if-then-else, or case statement) or each repetitive statement (e.g. while, for, or repeat statement) in the program.

In addition, there are two special nodes in \( V \), namely \( s \) and \( t \), that correspond to the entry and exit points of the program, respectively. Each statement node in \( V \) is assumed to
contain one simple statement. All other nodes (i.e. \(s, t\), and branching nodes) are assumed to be empty.

Every directed edge in \(E\) represents the possible flow of control between the nodes in \(V\) as implied by the transfer of control between statements in the program. Consequently,

1. For each statement node in \(V\), there is a single outgoing edge.
2. For each branching node in \(V\), there are at least two outgoing edges. Each outgoing edge of a branching node is associated with a predicate (i.e. a Boolean expression) whose truth value is equivalent to a specific outcome of the condition that implies the branching point corresponding to the node.
3. There is at most one edge from a node \(i\) to node \(j\) in \(E\) for any pair of nodes \(i\) and \(j\) in \(V\).
4. Node \(s\) has no incoming edge and \(t\) has no outgoing edge.

A path \((n_1, n_2, ..., n_m)\) in \(G(V, E)\) is a sequence of nodes in \(G\) such that \((n_i, n_{i+1}) \in E\), for all \(i, 1 \leq i \leq m - 1, m \geq 2\). A path \((i_1, ..., i_k)\) is a subpath of a path \((n_1, ..., n_m)\) if there exists a \(\delta, 0 \leq \delta \leq m - k\), such that for all \(j, 1 \leq j \leq k\), \(i_j = n_{j+\delta}\). A loop-free path is a path in which all nodes are distinct. A simple path is a path in which all nodes except possibly the first and the last are distinct. A complete path in \(G\) is a path whose first node is \(s\) and whose last node is \(t\). For example, path \((s, 1, 2, t)\) is a complete path for the flowgraph in Fig.1b.
begin
read (x);
while f1(x) do
begin
z := f2(x);
if f3(z)
then x := f4(x)
else x := f5(x);
writeln (z)
end
end.

Fig. 1a Program A

Fig. 1b Intermediate step

Fig. 1c Flowgraph of Program A

Let \( \Pi \) be a set of complete paths for the flowgraph of a given program. A node \( i \) is covered by \( \Pi \) if \( \Pi \) contains a complete path \((n_1, \ldots, n_m)\) such that \( i = n_j \) for some \( j, 1 \leq j \leq m \). An edge \((i_1, i_2)\) is covered by \( \Pi \) if \( \Pi \) contains a complete path \((n_1, \ldots, n_m)\) such that \( i_1 = n_j \) and \( i_2 = n_{j+1} \) for some \( j, 1 \leq j \leq m - 1 \). A path \((i_1, \ldots, i_k)\) is covered by \( \Pi \) if \( \Pi \) contains a complete path \((n_1, \ldots, n_m)\) and path \((i_1, \ldots, i_k)\) is a subpath of path \((n_1, \ldots, n_m)\).
2.3 Definition of the Original $IP/O_2$-chains Coverage Criterion

A definition (or def) of a variable $x$ is an occurrence of $x$ in node $n$ (denoted by $d^x_n$) if $n$ contains a statement in which $x$ is assigned a value. A use of a variable $x$ is an occurrence of $x$ by which the value of $x$ is referenced. Uses of variables are further classified as computational uses ($c$-uses) and predicate uses ($p$-uses) [14]. A $c$-use of a variable $x$ at node $n$ (denoted by $c^x_n$) is a use of $x$ which directly affects the computation being performed (e.g., an occurrence of $x$ on the RHS of an assignment statement, for instance, the use of $x$ at node 3 in Fig.1c) or allows one to see the result of some earlier definition (e.g., an occurrence of $x$ in the list of variables of an output statement, for instance, the use of $z$ at node 7 in Fig.1c). A $p$-use of a variable $x$ on edge $(n,m)$ (denoted by $p^x_{(n,m)}$) is a use of $x$ which directly affects the control flow of the program (e.g., an occurrence of $x$ in the Boolean expression associated with an outgoing edge of a branching node $n$ in a flowgraph, for instance, the use of $x$ on edge $(2,7)$ in Fig.1c). Examples of the classification of variable occurrences in a flowgraph representing a Pascal program can be found in [2, 14]. For the purposes of this thesis, we assume the classification given in [2].

A path $(n_1, n_2, ..., n_{m-1}, n_m)$ is a def-clear path with respect to a variable $x$ from node $n_1$ to node $n_m$ or from node $n_1$ to edge $(n_{m-1}, n_m)$ if either

1. $m = 2$, or
2. $m > 2$ and there are no definitions of $x$ at nodes $n_2$ to $n_{m-1}$.

We say that $d^x_i$ and $c^x_j$ form a du-pair (denoted by tuple $(d^x_i, c^x_j)$) if there is a def-clear path with respect to $x$ from node $i$ to node $j$. Similarly, $d^x_i$ and $p^x_{(j,k)}$ form a du-pair (denoted by $(d^x_i, p^x_{(j,k)})$) if there is a def-clear path with respect to $x$ from node $i$ to edge...
(j,k). For ease of reference, \(u^x_q\) denotes henceforth either \(c^x_q\) if \(q\) represents a node, or \(p^x_q\) if \(q\) represents an edge. A path \((i, \ldots, q)\) is called an activating path for a du-pair \((d^x_i, u^x_q)\) if it is a def-clear path with respect to \(x\) from \(i\) to \(q\).

Let \(\Pi\) be a set of complete paths for the flowgraph \(G(V,E)\) of a given program \(P\). We say that:

a) a du-pair is covered by \(\Pi\) if an activating path for the du-pair is covered by \(\Pi\).

b) \(\Pi\) satisfies the all-uses (all-c-uses, all-p-uses resp.) criterion if an activating path for each du-pair \(((d^x_i, c^x_j), (d^x_i, p^x_{(j,k)})\) resp.) in \(G(V,E)\) is covered by \(\Pi\).

c) \(\Pi\) satisfies the all-c-uses/some-p-uses (all-p-uses/some-c-uses resp.) criterion if \(\Pi\) satisfies all-c-uses (all-p-uses resp.) criterion and in addition, if a variable \(x\) is defined at node \(i\) and has no c-use (p-uses resp.) in \(G(V,E)\) then an activating path for some \((d^x_i, p^x_{(j,k)})\) \((d^x_i, c^x_j)\) resp.) in \(G(V,E)\) is covered by \(\Pi\).

d) \(\Pi\) satisfies the all-defs criterion if an activating path for some \((d^x_i, p^x_{(j,k)})\) or \((d^x_i, c^x_j)\) in \(G(V,E)\) is covered by \(\Pi\).

e) \(\Pi\) satisfies all-nodes (all-edges, all-paths resp.) criterion if each node (each branch, each path resp.) in \(G(V,E)\) is covered by \(\Pi\).

The all-du-paths criterion requires that all the du-paths in a flowgraph be covered at least once. That is, if there exists more than one activating path that cover the same du-pair then the all-du-paths criterion requires all these paths to be covered. A path \((n_1, n_2, \ldots, n_{m-1}, n_m)\) is a du-path with respect to a variable \(x\) if \(n_1\) has a def of \(x\) and either

1. \(n_m\) has a c-use of \(x\) and \((n_1, \ldots, n_m)\) is a def-clear simple path with respect to \(x\), or
2. \( (n_{m-1}, n_m) \) has a p-use of \( x \) and \( (n_1, \ldots, n_{m-1}) \) is a def-clear loop-free path with respect to \( x \) [14]. Formally, a set of complete paths \( \Pi \) satisfies the all-du-paths criterion if every du-path for each du-pair in a flowgraph is covered by \( \Pi \).

The required \( k \)-tuples criteria ask for the coverage of each \( k \)-dr interaction. A \( k \)-dr interaction \( (k \geq 2) \) is a sequence of \( k-1 \) du-pairs \( (d_{1,1}^{x_1}, u_2^{x_1}), (d_{2,2}^{x_2}, u_3^{x_2}), \ldots, (d_{k-1,k}^{x_{k-1}}, u_k^{x_{k-1}}) \) which are associated with \( k \) distinct nodes \( 1, 2, 3, \ldots, k \), where for all \( i, 1 \leq i \leq k \), the \( i \)-th definition \( d_i^{x_i} \) at node \( i \) reaches the \( i \)-th use \( u_{i+1}^{x_i} \) at node \( i+1 \) through def-clear path with respect to \( x_i \) [13]. Formally, a set of complete paths \( \Pi \) satisfies the required \( k \)-tuples criterion for a given \( k \), if an activating path for every \( k \)-dr interaction in a flowgraph is covered by \( \Pi \). An activating path for \( k \)-dr interaction \( [d_{1,1}^{x_1}, u_2^{x_1}, d_{2,2}^{x_2}, u_3^{x_2}, \ldots, d_{k-1,k}^{x_{k-1}}, u_k^{x_{k-1}}] \) is a path \( (n_1, \ldots, n_2, \ldots, n_3, \ldots, n_{k-1}, \ldots, n_k) \) in which \( (n_i, \ldots, n_{i+1}) \) is def-clear path with respect to variable \( x_i \), \( 1 \leq i \leq k-1 \). If the last use is a p-use, the \( k \)-dr interaction is covered twice, once for each outcome of a predicate, to ensure branch coverage. If the first definition or the last use of a \( k \)-dr interaction occurs in a loop, two test paths are generated for that interaction, reflecting two iteration counts for the loop, one for the minimum iteration of the loop [13], the other for a larger number of iterations [13]. In a newer version of this criterion called required \( k \)-tuples* [1], the du-pairs in a \( k \)-dr interaction need not be distinct.

The context coverage criterion requires that each elementary data context in a flowgraph be covered at least once, whereas the ordered context coverage criterion requires that each ordered elementary data context be covered at least once. An (ordered) elementary data context of a node \( i \) in the flowgraph of a given program that uses a vector of variables \( X(i) = [x_1, x_2, \ldots, x_n] \) is a (ordered) set of definitions \( ec(i) = [d(x_1), d(x_2), \ldots, d(x_n)] \) of all variables from \( X(i) \) such that there exists a control path from the
entry node to node $i$ and all definitions from $ec(i)$ reach node $i$ through this path (in the given order) without redefining the variables in $X(i)$ [11]. Formally, a set of complete paths $\Pi$ satisfies the (ordered) context coverage criterion, if an activating path for every (ordered) elementary data context in each node in a flowgraph is covered by $\Pi$. A path $(n_1, ..., n_2, ..., n_m)$ is an activating path for an elementary data context in node $i$, if $[d_{n_1}^{x_1}, d_{n_2}^{x_2}, ..., d_{n_m}^{x_m}]$, if path $(n_i, ..., n_{i+1}, ..., n_m)$ is a def-clear path with respect to variable $x_i$, where $1 \leq i \leq m - 1$.

Before defining the original $IP/O_2$-chains coverage criterion, the motivation behind the criterion will be reviewed. First of all, it has been advocated to link $du$-pairs in a program to form a sequence of $du$-pairs in order to trace a sequence of computations which capture particular control and data dependencies in the program [10, 11]. This observation gives rise to the notion of affect.

A use of a variable $y$, which may either be a $c$-use or a $p$-use, is affected by a definition of a variable $x$ if either

a) $x$ and $y$ are the same variable and the use of $y$ is reached by the def of $x$ through a def-clear path with respect to $x$, or

b) a def of a variable $z$ is given in terms of a use of $x$ at a node which is reached by the def of $x$ through a def-clear path with respect to $x$ and the use of $y$ is affected by the def of $z$.

A df-chain is defined as an ordered sequence $(d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, ..., d_{n_m}^{x_m}, u_{n_{m+1}}^{x_m})$ of $du$-pairs $(d_{n_1}^{x_1}, u_{n_2}^{x_1}), (d_{n_2}^{x_2}, u_{n_3}^{x_2}), ..., (d_{n_m}^{x_m}, u_{n_{m+1}}^{x_m})$, where $m \geq 1$ and $u_{n_{m+1}}^{x_m}$ is affected by $d_{n_1}^{x_1}$.

An input of a program is the definition of a variable which is not given in terms of any other variable. In particular, variable initializations through assignment statements whose RHS is an expression consisting of only constants are also considered as inputs. An
output of a program is a $e$-use of a variable in an output statement of the program. Such an output will be called a variable output. If an output statement contains only constants, then it is called constant output. A constant output can only be indirectly captured by a $df$-chain that starts with a program input and terminates with a $p$-use which leads the flow of control to the constant output. Thus, a constant output in a procedure will be tested by following a $df$-chain from an input to the last $p$-use which leads the control flow to the constant output.

It is said that a variable output $o$ (or a predicate $p$) is affected by a program input $i$ if there is a $df$-chain that starts with $i$ and terminates with the $o$ (or $p$). Let $I$, $O$, and $U_p$ be the sets of all inputs, variable outputs, and $p$-uses in a program, respectively. Then, two binary relations with respect to the notion of affect, namely input-output relation and input-predicate relation, are defined as follows.

**Input-output relation** is defined as

$$R_{IO} = \{(i, o) \mid i \in I, o \in O, \text{ and } o \text{ is affected by } i \}.$$  

**Input-predicate relation** is defined as

$$R_{IP} = \{(i, p) \mid i \in I, p \in U_p, \text{ and } p \text{ is affected by } i \}.$$  

A $df$-chain that starts with $i$ and terminates with $o$ or $p$ is called an **IP/O-chain** (Input-Predicate/Output Chain).

An **IP/O$_n$-chain** ($n \geq 1$) is an **IP/O-chain** such that there are $m$ ($1 \leq m \leq n$) occurrences of $u_j^x d_j^y$, and for at least one node $k$, there are exactly $n$ occurrences of $u_k^z d_k^w$, where $x$, $y$, $z$, or $w$ stands for any variable. As a special case, each $du$-pair whose def is
an input and whose use is a variable output or a p-use is considered as an IP/O₁-chain. Thus, in Fig. 2, \([d₁^x, \mu_{(2,3)}], [d₁^x, c₆^x]\) and \([d₅^x, c₅^x d₅^x, c₆^x]\) are IP/O₁-chains, and \([d₁^x, c₅^x d₅^x, c₅^x d₅^x, c₆^x]\) is an IP/O₂-chain.

begin
read \((x)\);
if \(f₁(x)\)
then
begin
while \(f₂(x)\) do
\(x := f₃(x)\);
write ln\((x)\)
end
else
write ln ('invalid input')
end.

Fig. 2. Program B and Its Flowgraph

The IP/O₂-chains coverage criterion is satisfied by a set of complete paths \(\Pi\) in a flowgraph if

a) an activating path for every IP/Oₖ-chain \((1 \leq k \leq 2)\) for each \((i, o) \in R_{Io}\) is covered at least once by \(\Pi\), and

b) an activating path for every IP/O₁-chain for each \((i, p) \in R_{Ip}\) is covered at least once by \(\Pi\).

Where an activating path for an IP/O-chain

\([d_{n₁}^{x₁}, u_{n₂}^{x₂} d_{n₂}^{x₂}, u_{n₃}^{x₃}, ..., d_{nₘ}^{xₘ}, u_{nₘ₁}^{xₘ₁}]\)

is a path \((n₁, ..., n₂, ..., n₃, ..., nₘ, ..., nₘ₁)\) in which \((nᵢ, ..., nᵢ₊₁)\) is def-clear path with respect to variable \(xᵢ\) for \(1 \leq i \leq m\).
2.4 Hierarchy of Original Criteria With Respect To Strictly Includes Relation

The relative strength of software testing criteria has been determined by utilizing the inclusion relation [1, 2, 11, 13, 14, 17]. For two software testing criteria $A$ and $B$, $A$ includes $B$ iff for any given flowgraph, any set of complete paths that satisfies $A$ also satisfies $B$. $A$ strictly includes $B$ iff $A$ includes $B$ but $B$ does not include $A$. $A$ and $B$ are incomparable iff neither $A$ includes $B$ nor $B$ includes $A$.

In [16], the following hierarchy with respect to strictly includes relation was proved by Ural and Yang.
Fig. 3. The "Strictly Includes" Hierarchy of the Original Criteria
2.5 Definition of the Applicable $IP/O_2$-chains Coverage Criterion ($IP/O_2$-chains)*

In this section, we first review the definition of applicable software testing criteria given in Frankl and Weyuker [2], and define the applicable $IP/O_2$-chains coverage criterion. These definitions are based on the assumption that the program under test satisfies $a) - e)$ given in Section 2.1 and the flowgraph of a program is as defined in Section 2.2. Then we give the position of the applicable $IP/O_2$-chains coverage criterion ($IP/O_2$-chains)* in the hierarchy of the applicable software testing criteria in Section 2.6.

A complete path is executable or feasible if there exists some assignment of values to input variables, non-local variables, and parameters which causes the path to be executed. We say a path is executable if it is a subpath of an executable complete path.

A du-pair ($IP/O_2$-chain respectively) is executable if there is some executable complete path which covers it; otherwise, it is non-executable. We define subsets $f_{du}(x, i)$ and $f_{du}(x, i)$ for each variable $x$ and for each node $i$ in the flowgraph $G(V,E)$ of a program $P$:

$$f_{du}(x, i) = \{ \text{nodes } j \text{ such that } x \text{ has } c\text{-use at node } j \text{ and there is an executable def-clear path with respect to } x \text{ from } i \text{ to } j \}.$$  

$$f_{du}(x, i) = \{ \text{edges } (j, k) \text{ such that } x \text{ has a } p\text{-use at edge } (j, k) \text{ and there is an executable def-clear path with respect to } x \text{ from } i \text{ to edge } (j, k) \}.$$  

**Definition 2.1** For each software testing criterion $C$ defined in Section 2.3 on the basis of du-pairs, we define $C^*$ by selecting the required du-pair from $f_{du}(x, i)$ and $f_{du}(x, i)$.  

18
That is, only executable \(du\)-pairs are considered here. So, in the flowgraph \(G(V,E)\) of a program \(P\),

1. \((all-defs)^*\) requires some \((d^X_{m'}, u^X_n)\) such that \(n \in fdcu(x,m)\) or some \((d^X_{m'}, p^X_{(i,j)})\) such that \((i, j) \in fdp(x, m)\) to be covered.
2. \((all-c-uses)^*\) requires all \((d^X_{m'}, u^X_n)\) such that \(n \in fdcu(x,m)\) to be covered.
3. \((all-p-uses)^*\) requires all \((d^X_{m'}, p^X_{(i,j)})\) such that \((i, j) \in fdp(x, m)\) to be covered.
4. \((all-p-uses/some-c-uses)^*\) requires all \((d^X_{m'}, p^X_{(i,j)})\) such that \((i, j) \in fdp(x, m)\) to be covered; In addition, if \(fdpu(x, m)\) is empty and \(fddc(x, m)\) is not empty then some \((d^X_{m'}, u^X_n)\) such that \(n \in fdcu(x,m)\) to be covered.
5. \((all-c-uses/some-p-uses)^*\) requires all \((d^X_{m'}, u^X_n)\) such that \(n \in fdcu(x, m)\) to be covered; In addition, if \(fddc(x, m)\) is empty and \(fdpu(x, m)\) is not empty then some \((d^X_{m'}, p^X_{(i,j)})\) such that \((i, j) \in fdp(x, m)\) to be covered.
6. \((all-uses)^*\) requires all executable \(du\)-pairs to be covered.
7. \((all-du-paths)^*\) requires all executable \(du\)-paths to be covered.
8. applicable \(IP/O_2\)-chains coverage criterion (\(IP/O_2\)-chains)\(^*\) requires all executable \(IP/O_2\)-chains to be covered.

**Definition 2.2** In the flowgraph \(G(V,E)\) of a program \(P\),

1. \((all-edges)^*\) requires all executable edges to be covered, where an edge is executable if there exists an executable complete path that covers it.
2. \((all-nodes)^*\) requires all executable nodes to be covered, where a node is executable if there exists an executable complete path that covers it.
3. \((all-paths)^*\) requires all executable paths to be covered, where a path is executable if there exists an executable complete path that covers it.
2.6 Hierarchy of Applicable Criteria With Respect To Strictly Includes Relation

In this section, we prove the main theorem in this chapter to give the proper position of criterion \((IP/O_2-chains)^*\) in the hierarchy of the applicable family of software testing criteria.

**Theorem 2.1** i) \((all-paths)^*\) strictly includes \((IP/O_2-chains)^*\).

ii) \((IP/O_2-chains)^*\) is incomparable with \((all-du-paths)^*\), \((all-uses)^*\), \((all-c-uses)^*\), \((all-defs)^*\), \((all-p-uses)^*\) and \((all-nodes)^*\) criteria.

**Proof:** Consider the program \(C\) whose flowgraph is given below:

![Flowgraph of Program C](image)

Fig. 4. Program C and Its Flowgraph

So the executable \(IP/O_2\)-chains and their subdomains are:
Table 1: IP/O₁-chains and their subdomains in Program C (i ≤ 2)

<table>
<thead>
<tr>
<th>IP/O₁-CHAINS</th>
<th>SUBDOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[d₂, p₁(3,41)]</td>
<td>all y's</td>
</tr>
<tr>
<td>[d₂, u₁d₂d₂, p₁(3,41)]</td>
<td>all y's</td>
</tr>
<tr>
<td>[d₂, u₁d₂d₂, u₁d₂d₂, p₁(3,41)]</td>
<td>all y's</td>
</tr>
</tbody>
</table>

a) Clearly, (all-paths)* includes (IP/O₁-chains)*.

b) (IP/O₂-chains)* does not include (all-paths)*, (all-p-uses)*, (all-c-uses)*, (all-defs)* and (all-nodes)* since the complete path \( p_1 = (s, 1, 2, 3, 41, 42, 3, 41, 42, 3, 41, 42, 3, 41, 42, 3, 41, 51, 52, 6, 7, i) \) covers all executable IP/O₁ and IP/O₂-chains for any \( y \leq 0 \) and \( p_1 \) does not cover executable path \( (s, 1, 2, 3, 41, 42, 3, 41, 42, 3, 41, 42, 3, 41, 42, 3, 41, 51, 52, 6, 81, 82, i) \) (required by (all-paths)*), \( (d₁⁻¹, p₁⁻¹(6,81)) \) (required by (all-p-uses)*), \( (d₁⁻¹, u₁⁻¹(82)) \) (required by (all-c-uses)*), \( (d₁⁻¹, u₁⁻¹(81)) \) (required by (all-defs)*) and node 81 (required by (all-nodes)*).

c) From the transitivity of the strictly includes relation, b) and Theorem 1 [2] we have that (IP/O₂-chains)* does not include (all-uses)*.

d) It can be proved by the same argument of Theorem 3 [16] that (all-uses)* does not include (IP/O₂-chains)*.

e) It's trivial to see that (all-c-uses)*, (all-defs)*, (all-p-uses)* and (all-nodes)* do not include (IP/O₂-chains)*.
By the same argument of Theorem 4 [16], we can show that \((IP/O_2-chains)^*\) and \((all-du-paths)^*\) are incomparable.

So, by the transitivity of strictly includes relation and Theorem 1 [2], the hierarchy of the family of the applicable criteria is as follows:

\[
\begin{align*}
&\text{(IP/O_2-chains)*} & \text{(all-paths)*} & \text{(all-edges)*} \\
&\text{(all-c-uses/some-p-uses)*} & \text{(all-defs)*} & \text{(all-p-uses)*} \\
&\text{(all-uses)*} & & \\
&\text{(all-p-uses/some-c-uses)*} & \\
\end{align*}
\]

\[\longrightarrow \quad \text{Strictly includes (known results)} \]
\[\longrightarrow \quad \text{Strictly includes (new result)} \]

Fig. 5 The "Strictly Includes" Hierarchy of the Applicable Criteria

Although \textit{required-k-tuples coverage, context-coverage, ordered-context coverage} criteria are not used in the above hierarchy, we define their applicable versions because they will be referred to in Chapter 3.

1. \((\text{required-k-tuples})^*\) requires all executable \(k-dr\) interactions \((k \geq 2)\) in a flowgraph to be covered, where a \(k-dr\) interaction is executable if there is an executable complete path that covers the activating path of the \(k-dr\) interaction.
2. \((context)^*\) requires each executable elementary data context in a flowgraph to be covered at least once, where an elementary data context is executable if there is an executable complete path that covers the activating path of the elementary data context.

3. \((ordered-context)^*\) requires each executable ordered elementary data context in a flowgraph to be covered at least once, where an ordered elementary data context is executable if there is an executable complete path that covers the activating path of the ordered elementary data context.
Chapter 3
Subdomain Based Software Testing Criteria

In this Chapter, we first review the definitions of subdomain-based software testing criteria, measure $M$ and properly covers relation given by Frankl and Weyuker in [3] and define the subdomain-based $IP/O_2$-chains coverage criterion. Then, we identify the position of subdomain based $IP/O_2$-chains coverage criterion ($IP/O_2$-chain)$^5$ in the hierarchy of the family of subdomain based software testing criteria.

3.1 Program Model

As in Section 2.1, in this chapter we assume that programs under test satisfy assumptions a) - e) with the exception that there may be goto statements. We also restrict our attention to programs with finite input domain, but place no bound on the input domain size.

3.2 Definition of the Subdomain Based $IP/O_2$-chains Coverage Criterion

In this section, we review some definitions and results in [3] and define the subdomain-based $IP/O_2$-chains coverage criterion. A multi-set is a collection of objects in
which duplicates may occur. We shall use set notations and set-theoretic operator
symbols to denote the corresponding multi-set and operators throughout. A multi-set $S_1$ is
a sub-multi-set of $S_2$ if the number of occurrences of each element in $S_1$ is less than or
equal to the number of the occurrences of the element in $S_2$.

The input domain of a program is the set of possible inputs. We restrict attention
to programs with finite input domain. A test suite is a multi-set of test cases, each of
which is an element of the input domain.

A testing criterion $C$ is subdomain-based (denoted by $(C)^s$) if for each program $P$
and the corresponding specification $S$, there is a non-empty multi-set of subdomains,
$SD_C(P, S)$, such that $(C)^s$ requires the selection of one element from each subdomain in
$SD_C(P, S)$. That is, for a given pair $(P, S)$, $(C)^s$ divides $D$ into subdomains \{$D_1(C)$, $D_2(C)$,
\ldots, $D_r(C)$\} and forms a test suite $T$ by independently randomly selecting an element of
each $D_i(C) \in SD_C(P, S)$, $1 \leq i \leq r$, according to a uniform distribution. It is assumed that
$SD_C(P, S)$ is non-empty. From this assumption and the fact that one test case is selected
from each subdomain $D_i(C) \in SD_C(P, S)$, the empty test suite does not result by the
application of any $(C)^s$ to $(P, S)$ for every program $P$ and specification $S$.

**Definition 3.1** Subdomain-based IP/O2-chains coverage criterion $(IP/O2-chains)^s$ is
defined by applicable $(IP/O2-chains)^*$ with $SD_{(IP/O2)}(P, S)$ consisting of all subdomains
of executable IP/O2-chains in the flowgraph of $P$.

Similarly, $(all-uses)^s$, $(all-p-uses)^s$, $(context)^s$, $(ordered-context)^s$ and $(required k-
tuples)^s$ criteria are defined respectively by $(all-uses)^*$, $(all-p-uses)^*$, $(context)^*$,
$(ordered-context)^*$ and $(required k-tuples)^*$ criteria.
A decision is a maximal Boolean expression controlling the execution of a conditional statement or loop. For instance, in statement if \((x = 1) \text{ and } (y = 2)\) then writeln('1'), \((x = 1)\) and \((y = 2)\) is a decision. The decision-coverage criterion requires that every decision take on the value true at least once during the testing and also take on value false at least once. Subdomain-based decision-coverage criterion, \((\text{decision})^s\), is defined by \(SD_{DC}(P, S)\) consisting of all subdomains of each decision in the flowgraph of \(P\) (i.e. there are two subdomains for each decision, one consisting of all inputs that cause it to evaluate to true at some point during execution and one consisting of all inputs that cause it to evaluate to false at some point during execution.

A condition is a Boolean variable, a relational expression, or a Boolean function occurring in a decision. For example in the statement if \((x = 1) \text{ and } (y = 2)\) then writeln('1'), both \((x = 1)\) and \((y = 2)\) are conditions. The condition-coverage criterion requires that every condition take on the value true at least once and take on the value false at least once. The subdomain-based condition-coverage criterion, \((\text{condition})^s\), is defined by condition-coverage criterion.

The subdomain-based decision-condition-coverage criterion, \((\text{decision-condition})^s\), is defined by decision-condition-coverage criterion which requires that every decision takes on value true at least once and takes value false at least once and that every condition take on the value true at least once and take on the value false at least once during testing. For example, in the statement if \((x = 1) \text{ and } (y = 2)\) then writeln('1'), decision-condition-coverage criterion would require six testcases for the following: \(((x = 1) \text{ and } (y = 2)) = \text{true}, ((x = 1) \text{ and } (y = 2)) = \text{false}, \((x = 1), (x \neq 1), (y = 2), (y \neq 2)\).

Therefore, the subdomains of \((\text{decision-condition})^s\) are:
1. the set of inputs such that decision \((x = 1) \text{ and } (y = 2)\) = true, i.e. the set \(\{(x, y) \mid x = 1 \text{ and } y = 2\}\),

2. the set of inputs such that decision \((x = 1) \text{ and } (y = 2)\) = false, i.e. the set \(\{(x, y) \mid (x = 1 \text{ and } y \neq 2) \text{ or } (x \neq 1 \text{ and } y = 2) \text{ or } (x \neq 1, y \neq 2)\}\),

3. the set of inputs such that condition \((x = 1)\) is true, i.e. the set \(\{(x, y) \mid x = 1\}\)

4. the set of inputs such that condition \((x = 1)\) is false, i.e. the set \(\{(x, y) \mid x \neq 1\}\)

5. the set of inputs such that condition \((y = 2)\) is true, i.e. the set \(\{(x, y) \mid y = 2\}\)

6. the set of inputs such that condition \((y = 2)\) is false, i.e. the set \(\{(x, y) \mid y \neq 2\}\)

The subdomain-based multiple-condition-coverage criterion, \((\text{multiple-condition})^\circ\), is defined by multiple-condition-coverage criterion which requires that every combination of truth values of conditions occurs at least once during testing. For example, in the statement \(\text{if } (x = 1) \text{ and } (y = 2) \text{ then writeln('1')}, \) multiple-condition-coverage criterion would require four test cases for the following:

\[(x = 1, y = 2), (x \neq 1, y = 2), (x = 1, y \neq 2), (x \neq 1, y \neq 2)\].

The subdomains of \((\text{multiple-condition})^\circ\) are:

1. the set of inputs such that condition \((x = 1)\) is true and condition \((y = 2)\) is true, i.e. the set \(\{(x, y) \mid x = 1 \text{ and } y = 2\}\)

2. the set of inputs such that condition \((x = 1)\) is false and condition \((y = 2)\) is true, i.e. the set \(\{(x, y) \mid x \neq 1 \text{ and } y = 2\}\)

3. the set of inputs such that condition \((x = 1)\) is true and condition \((y = 2)\) is false, i.e. the set \(\{(x, y) \mid x = 1 \text{ and } y \neq 2\}\)

4. the set of inputs such that condition \((x = 1)\) is false and condition \((y = 2)\) is false, i.e. the set \(\{(x, y) \mid x \neq 1, y \neq 2\}\)
The subdomain-based reduced-decision-condition-coverage criterion, (reduced
decision-condition) is defined by decision-condition-coverage criterion which requires
that each decision \(d\) of form \(A\) and \(B\) takes on value true at least once; and condition \(A\)
and \(B\) take on the value false individually at least once; each decision \(d\) of form \(C\) or \(D\)
takes on value false at least once; conditions \(C\) and \(D\) take on the value true individually
at least once; for all other decisions they are the same as the decision-condition-coverage
criterion. For example, in the statement if \((x = 1)\) and \((y = 2)\) then writeln('1'), reduced-
decision-condition-coverage criterion would require three test cases for the following: \(((x
= 1)\) and \((y = 2)\) = true, \((x \neq 1)\), \((y \neq 2)\). The subdomains of (reduced decision-condition) are:

1. the set of inputs such that decision \(((x = 1)\) and \((y = 2)\) = true, i.e. the set \(\{(x, y)\\
\mid x = 1\ \text{and} \ y = 2\}\),
2. the set of inputs such that condition \((x = 1)\) is false, i.e. the set \(\{(x, y) \mid x \neq 1\}\)
3. the set of inputs such that condition \((y = 2)\) is false, i.e. the set \(\{(x, y) \mid y \neq 2\}\)

For statement if \((x = 3)\) or \((y = 4)\) then writeln('2'), reduced-decision-condition-
coverage criterion would require three test cases for the following: \(((x = 3)\) or \((y = 4)\) =
false, \((x = 3)\), \((y = 4)\). The subdomains of (reduced decision-condition) are:

1. the set of inputs such that decision \(((x = 3)\) or \((y = 4)\) = false, i.e. the set \(\{(x, y)\\
\mid (x \neq 3, y \neq 4)\}\),
2. the set of inputs such that condition \((x = 3)\) is true, i.e. the set \(\{(x, y) \mid x = 3\}\)
3. the set of inputs such that condition \((y = 4)\) is true, i.e. the set \(\{(x, y) \mid y = 4\}\)

Clarke et al. [1] pointed out certain problems with the original required k-tuples
criterion and defined the required k-tuples criterion by making the following two
modifications:
1. all l-dr interactions must be exercised for l ≤ k, and
2. the statements s_r in the path need not to be distinct.

Without modification (1), \((\text{required k-tuples})^*\) fails to include \((\text{required (k-1)-tuples})^*\), and without modification (2), \((\text{required 2-tuples})^*\) fails to include \((\text{all-uses})^*\). The subdomain-based required k-tuples coverage criterion, \((\text{required k-tuples}^+)^s\), is defined by the required k-tuples criterion defined above.

3.3 Measure \(M\) and Properly Covers Relation

Given a program \(P\) and a specification \(S\), a failure-causing input \(t\) is one such that the output produced by \(P\) on input \(t\) does not agree with the specified output.

Consider \(SD(C, P, S) = \{D_1(C), D_2(C), \ldots, D_l(C)\}\) for a given triple \((C, P, S)\). Let \(d_i = |D_i(C)|\) and let \(m_i\) be the number of failure-causing inputs in \(D_i(C)\). Then, the probability that a test suite formed by using \(C\) will expose at least one fault in \(P\) is given by measure \(M\), i.e., \(M(C, P, S) = 1 - \prod_{i=1}^{l} (1 - \frac{m_i}{d_i})\).

Let \(C_1, C_2\) be two software testing criteria. \(C_1\) covers \(C_2\) for \((P, S)\) if for every subdomain \(D\) in \(SD(C_2(P, S))\), there is a collection \(\{D_1, \ldots, D_n\}\) of subdomains in \(SD(C_1(P, S))\) such that \(D = D_1 \cup \ldots \cup D_n\). \(C_1\) is universally covers \(C_2\) if for every program, specification pair \((P, S)\), \(C_1\) covers \(C_2\). \(C_1\) properly covers \(C_2\) for \((P, S)\) if the multi-set union of multi-set \(\{D_1, \ldots, D_n\}\) for each \(D\) in \(SD(C_2(P, S))\) is a sub-multi-set of \(SD(C_1(P, S))\). \(C_1\) is universally properly covers \(C_2\) if for every program, specification pair \((P, S)\), \(C_1\) properly covers \(C_2\) for \((P, S)\).

Theorem 3.1 [4] If \(C_1\) properly covers \(C_2\) for program \(P\) and specification \(S\), then
\[ M(C_1, P, S) \geq M(C_2, P, S). \]

3.4 Hierarchy of Subdomain-based Criteria with respect to Universally Properly Covers Relation

In this section, we compare \((IP/O_2\text{-chains})^s\) coverage criterion with other subdomain-based software testing criteria.

**Theorem 3.2**

i) \((IP/O_2\text{-chains})^s\) does not universally properly cover \((decision)^s\) and \((condition)^s\).

Therefore, \((IP/O_2\text{-chains})^s\) does not universally properly cover any criterion in the hierarchy.

ii) \((all\text{-uses})^s\) and \((multiple\text{-condition})^s\) do not universally properly cover \((IP/O_2\text{-chains})^s\). Therefore, \((IP/O_2\text{-chains})^s\) is incomparable with \((all\text{-uses})^s\), \((all\text{-p-uses})^s\), \((decision)^s\), \((multiple\text{-condition})^s\), \((decision\text{-condition})^s\), \((reduced\text{-decision\text{-condition})^s}\) and \((condition)^s\) criteria.

**Proof:** Let \(P\) be program \(D\) with the following flowgraph representation.
Then, the executable $IP/O_1$ and $IP/O_2$-chains of $P$ and their subdomains are given in Table 2:

<table>
<thead>
<tr>
<th>$IP/O$-CHAINS</th>
<th>SUBDOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[d_2^i, p_{(3,41)}^i]$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>$[d_2^i, u_{42}^i d_{42}^i, p_{(3,41)}^i]$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>$[d_2^i, u_{42}^i d_{42}^i, u_{42}^i d_{42}^i, p_{(3,41)}^i]$</td>
<td>all $y$'s</td>
</tr>
</tbody>
</table>

The (decision) requires the subdomains given in Table 3:

<table>
<thead>
<tr>
<th>DECISIONS</th>
<th>SUBDOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \leq 5$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>$i &gt; 5$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>( y \leq 10 )</td>
<td>( y &lt; 0 )</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td>( y &gt; 10 )</td>
<td>( y \geq 0 )</td>
</tr>
</tbody>
</table>

The *(condition)*\(^s\) requires the following subdomains:

Let \( cond_1, cond_2, cond_3, cond_4 \) be condition \( i \leq 5, i > 5, y \leq 10 \) and \( y > 10 \), respectively.
Since *(condition)*\(^s\) requires that every condition takes on the value true at least once and takes on the value false at least once \([4]\), \( SD_{CC} = \{D_1, D_2, D_3, D_4, D_1, D_2, D_3, D_4\} \), where \( D_i \) is the set of inputs such that condition \( cond_i \) = true, for \( i = 1, 2, 3, 4; \) and \( D_2 (D_1 \) resp.) is the set of inputs such that condition \( cond_1 \) = false \( (cond_2 = \text{false resp.}) \), \( D_4 (D_3 \) resp.) is the set of input such that condition \( cond_3 \) = false \( (cond_4 = \text{false resp.}) \), So, \( D_1 = D_2 = \{\text{all } y's\}, D_3 = \{y < 0\}, D_4 = \{y \geq 0\} \).

Let the input domain of program \( P \) is \( D = \{-4, -3, -2, -1, 0, 1, 2, 3, ..., 95\} \) and the set of failure-causing inputs is \( \{-4, -1\} \). Let \( C_1 \) be *(IP/O2-chains)*\(^s\), \( C_2 \) be *(decision)*\(^s\) and \( CC \) be *(condition)*\(^s\). Then \( SD_{C_1} = \{D, D, D\} \) and \( SD_{C_2} = \{D, D, D_3, D_4\} \), where \( D_3 = \{-4, -3, -2, -1\} \) and \( D_4 = \{0, 1, 2, 3, ..., 95\} \). Then

\[
M(C_1, P, S) = 1 - (1 - \frac{2}{100})^3 = 0.0588
\]
\[
M(C_2, P, S) = 1 - (1 - \frac{2}{100})^2(1 - \frac{2}{4}) = 0.5198
\]
\[
M(CC, P, S) = 1 - (1 - \frac{2}{100})^4(1 - \frac{2}{4})^2 = 0.769
\]

So, \( M(C_1, P, S) < M(C_2, P, S) < M(CC, P, S) \) and *(IP/O2-chains)*\(^s\) does not properly cover *(decision)*\(^s\) and *(condition)*\(^s\) by Theorem 3.1. Hence \( i \) holds by the definition of the universally properly covers relation.
To prove that (all-uses)\(s\) and (multiple-decision)\(s\) do not universally properly cover (IP/O\(j\)-chains)\(s\), let us consider the program \(E\) given below:

```
begin
  (1a) read \(n\);
  (1b) \(a := 1\);
  (1c) \(b := 1\);
  (2) if \((n < 2)\) then
  (3) writeln ('Error')
  else
    begin
      \(n := n-2;\)
      \(i := 1;\)
      writeln \((a)\);
      writeln \((b)\);
      (5) while \((i <= n)\) do begin
      (6a) \(c := a+b;\)
      (6b) \(b := b+1;\)
      (6c) writeln \((c)\);
      (6d) \(i := i+1;\)
      end
    end
end.
```

Fig. 7. Program \(E\) and Its Flowgraph

The \(du\)-pairs and the \(IP/Oj\)-chains for \(j = 1, 2\) in Program \(E\) are given in Tables 4, 5, and 6, respectively.
### Table 4: du-pairs in Program E

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DU-PAIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>((d_{1a}^n, p_{(2,3)}^n), (d_{1a}^n, p_{(2,4a)}^n), (d_{1a}^n, c_{4a}^n), (d_{4a}^n, p_{(5,1)}^n), (d_{4a}^n, p_{(5,6a)}^n))</td>
</tr>
<tr>
<td>(a)</td>
<td>((d_{1b}^a, c_{4c}^a), (d_{1b}^a, c_{6a}^a))</td>
</tr>
<tr>
<td>(b)</td>
<td>((d_{1c}^b, c_{4d}^b), (d_{1c}^b, c_{6a}^b), (d_{1c}^b, c_{6b}^b), (d_{6a}^b, c_{6b}^b))</td>
</tr>
<tr>
<td>(i)</td>
<td>((d_{4b}^i, p_{(5,i)}^i), (d_{4b}^i, p_{(5,6a)}^i), (d_{4b}^i, c_{6d}^i))</td>
</tr>
<tr>
<td>(c)</td>
<td>((d_{6a}^c, c_{6c}^c))</td>
</tr>
</tbody>
</table>

### Table 5: IP/O₁-chains in Program E

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>IP/O₁-CHAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>([d_{1a}^n, u_{(2,3)}^n], [d_{1a}^n, u_{(2,4a)}^n], [d_{1a}^n, u_{4a}^n d_{4a}^n u_{(5,1)}^n], [d_{1a}^n, u_{4a}^n d_{4a}^n u_{(5,6a)}^n])</td>
</tr>
<tr>
<td>(a)</td>
<td>([d_{1b}^a, u_{4c}^a], [d_{1b}^a, u_{6a}^a d_{6b}^a u_{6c}^a])</td>
</tr>
<tr>
<td>(b)</td>
<td>([d_{1c}^b, u_{6d}^b], [d_{1c}^b, u_{6a}^b d_{6d}^b u_{6c}^b], [d_{1c}^b, u_{6b}^b d_{6b}^b u_{6a}^b d_{6d}^b u_{6c}^b])</td>
</tr>
<tr>
<td>(i)</td>
<td>([d_{4b}^i, u_{(5,i)}^i], [d_{4b}^i, u_{(5,6a)}^i], [d_{4b}^i, u_{6d}^i d_{6d}^i u_{(5,i)}^i], [d_{4b}^i, u_{6d}^i d_{6d}^i u_{(5,6a)}^i])</td>
</tr>
</tbody>
</table>

### Table 6: IP/O₂-chains in Program E

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>IP/O₂-CHAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>([d_{1c}^b, u_{6b}^b d_{6b}^b u_{6a}^b d_{6b}^b u_{6c}^b])</td>
</tr>
<tr>
<td>(i)</td>
<td>([d_{4b}^i, u_{6d}^i d_{6d}^i u_{6d}^i d_{6d}^i u_{6d}^i d_{6d}^i u_{(5,i)}^i], [d_{4b}^i, u_{6d}^i d_{6d}^i u_{6d}^i d_{6d}^i u_{(5,6a)}^i])</td>
</tr>
</tbody>
</table>

34
Assume that the input domain of program $E$ is $\{1, 2, ..., 9\}$. Then, the set of failure-causing inputs is $\{5, 6, 7, 8, 9\}$ because program $E$ is intended to calculate the first $n$ Fibonacci numbers with $n \geq 2$ and it only provides correct output for $n \leq 4$ due to a fault in node $5b$, i.e. $a := b$ and $b := c$ are replaced by $b := b + 1$.

<table>
<thead>
<tr>
<th>DU-PAIRS</th>
<th>SUBDOMAINS</th>
<th>FAILURE-CAUSING RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(d_{1a}^n \  p_{(2,3)}^n)$</td>
<td>${1}$</td>
<td>0</td>
</tr>
<tr>
<td>$(d_{1a}^n \  p_{(2,4a)}^n)$</td>
<td>${2, 3, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{4a}^n)$</td>
<td>${2, 3, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$(d_{4a}^n \  p_{(5,4)}^n)$</td>
<td>${2, 3, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$(d_{4a}^n \  p_{(5,6a)}^n)$</td>
<td>${3, 4, ..., 9}$</td>
<td>$\frac{5}{7}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{4c}^n)$</td>
<td>${2, 3, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{6a}^n)$</td>
<td>${3, 4, ..., 9}$</td>
<td>$\frac{5}{7}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{4d}^n)$</td>
<td>${2, 3, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{6a}^n)$</td>
<td>${3, 4, ..., 9}$</td>
<td>$\frac{5}{7}$</td>
</tr>
<tr>
<td>$(d_{1c}^n \  c_{6b}^n)$</td>
<td>${3, 4, ..., 9}$</td>
<td>$\frac{5}{7}$</td>
</tr>
<tr>
<td>$(d_{6b}^n \  c_{6a}^n)$</td>
<td>${4, 5, ..., 9}$</td>
<td>$\frac{5}{6}$</td>
</tr>
<tr>
<td>Subdomain</td>
<td>Failure-Causing Rate</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>((d^{b}<em>{6b}, c^{b}</em>{6b}))</td>
<td>([4, 5, ..., 9])</td>
<td>(\frac{5}{6})</td>
</tr>
<tr>
<td>((d^{i}<em>{4b}, p^{i}</em>{(5,i)}))</td>
<td>([2])</td>
<td>0</td>
</tr>
<tr>
<td>((d^{i}<em>{4b}, p^{i}</em>{(5,6a)}))</td>
<td>([3, 4, ..., 9])</td>
<td>(\frac{5}{7})</td>
</tr>
<tr>
<td>((d^{i}<em>{4b}, c^{i}</em>{6d}))</td>
<td>([3, 4, ..., 9])</td>
<td>(\frac{5}{7})</td>
</tr>
<tr>
<td>((d^{i}<em>{6d}, p^{i}</em>{(5,i)}))</td>
<td>([3])</td>
<td>0</td>
</tr>
<tr>
<td>((d^{i}<em>{6d}, p^{i}</em>{(5,6a)}))</td>
<td>([4, 5, ..., 9])</td>
<td>(\frac{5}{6})</td>
</tr>
<tr>
<td>((d^{i}<em>{6d}, c^{i}</em>{6d}))</td>
<td>([4, 5, ..., 9])</td>
<td>(\frac{5}{6})</td>
</tr>
<tr>
<td>((d^{c}<em>{6d}, c^{c}</em>{6c}))</td>
<td>([3, 4, ..., 9])</td>
<td>(\frac{5}{7})</td>
</tr>
</tbody>
</table>

Table 8: Subdomains and failure-causing rates of the IP/O₁-chains in Program E

<table>
<thead>
<tr>
<th>IP/O₁-CHAIN</th>
<th>SUBDOMAINS</th>
<th>FAILURE-CAUSING RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>([d^{n}<em>{1a}, u^{n}</em>{(2,3)}])</td>
<td>([1])</td>
<td>0</td>
</tr>
<tr>
<td>([d^{n}<em>{1a}, u^{n}</em>{(2,4a)}])</td>
<td>([2, 3, ..., 9])</td>
<td>(\frac{5}{8})</td>
</tr>
<tr>
<td>([d^{n}<em>{1a}, u^{n}</em>{4a}, d^{n}<em>{4a}, u^{n}</em>{(5,i)}])</td>
<td>([2])</td>
<td>0</td>
</tr>
<tr>
<td>([d^{n}<em>{1a}, u^{n}</em>{4a}, d^{n}<em>{4a}, u^{n}</em>{(5,6a)}])</td>
<td>([3, 4, ..., 9])</td>
<td>(\frac{5}{7})</td>
</tr>
<tr>
<td>([d^{a}<em>{1b}, u^{a}</em>{4c}])</td>
<td>([2, 3, ..., 9])</td>
<td>(\frac{5}{8})</td>
</tr>
<tr>
<td>Subdomain</td>
<td>Failure-Causing Rate</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>$[d_{1c}^b u_{6a}^b u_{6a}^c u_{6c}^c]$</td>
<td>( {3, 4, \ldots, 9} )</td>
<td>( \frac{5}{7} )</td>
</tr>
<tr>
<td>$[d_{1c}^b u_{4d}^b]$</td>
<td>( {2, 3, \ldots, 9} )</td>
<td>( \frac{5}{8} )</td>
</tr>
<tr>
<td>$[d_{1c}^b u_{6a}^b u_{6a}^c u_{6c}^c]$</td>
<td>( {3, 4, \ldots, 9} )</td>
<td>( \frac{5}{7} )</td>
</tr>
<tr>
<td>$[d_{1c}^b u_{6b}^b u_{6b}^b u_{6d}^b u_{6d}^c u_{6c}^c]$</td>
<td>( {4, 5, \ldots, 9} )</td>
<td>( \frac{5}{6} )</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{6d}^i u_{6d}^i u_{(5,6)}^i]$</td>
<td>( {2} )</td>
<td>0</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{5d}^i]$</td>
<td>( {3, 4, \ldots, 9} )</td>
<td>( \frac{5}{7} )</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{6d}^i d_{6d}^i u_{(5,6)}^i]$</td>
<td>( {3} )</td>
<td>0</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{6d}^i d_{6d}^i u_{(5,6)}^i]$</td>
<td>( {4, 5, \ldots, 9} )</td>
<td>( \frac{5}{6} )</td>
</tr>
</tbody>
</table>

Table 9: Subdomains and failure-causing rates of \(IP/O_2\)-chains in Program \(E\)

<table>
<thead>
<tr>
<th>(IP/O_2)-CHAIN</th>
<th>SUBDOMAINS</th>
<th>FAILURE-CAUSING RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[d_{1c}^b u_{6b}^b u_{6b}^b u_{6d}^b u_{6d}^b u_{6d}^c u_{6c}^c]$</td>
<td>( {5, 6, \ldots, 9} )</td>
<td>( \frac{5}{5} )</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{6d}^i u_{6d}^i u_{(5,6)}^i]$</td>
<td>( {4} )</td>
<td>0</td>
</tr>
<tr>
<td>$[d_{4b}^i u_{6d}^i u_{6d}^i u_{(5,6)}^i]$</td>
<td>( {5, 6, \ldots, 9} )</td>
<td>( \frac{5}{5} )</td>
</tr>
</tbody>
</table>
Table 10: Subdomains and failure-causing rates of multiple-conditions in Program $E$

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>SUBDOMAINS</th>
<th>FAILURE-CAUSING RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n &lt; 2$</td>
<td>${1, 2}$</td>
<td>0</td>
</tr>
<tr>
<td>$n \geq 2$</td>
<td>${2, ..., 9}$</td>
<td>$\frac{5}{8}$</td>
</tr>
<tr>
<td>$n &lt; i$</td>
<td>${1, 2}$</td>
<td>0</td>
</tr>
<tr>
<td>$n \geq i$</td>
<td>${3, 4, ..., 9}$</td>
<td>$\frac{5}{7}$</td>
</tr>
</tbody>
</table>

Thus,

$M(\text{all-uses}, P, S) = 1 - (1 - \frac{5}{8})^5(1 - \frac{5}{7})^7(1 - \frac{5}{6})^4$,

$M(\text{multiple-condition } P, S) = 1 - (1 - \frac{5}{8})(1 - \frac{5}{7})$,

$M(\text{IP/O2-chains}, P, S) = 1 - (1 - \frac{5}{8})^4(1 - \frac{5}{7})^4(1 - \frac{5}{6})^2 (1 - \frac{5}{5})^2 = 1$,


So, \textit{(all-uses)} and \textit{(multiple-condition)} do not properly cover \textit{(IP/O2-chains)} by Theorem 3.1. Hence, \textit{ii)} holds by the definition of the universally properly covers relation, the transitivity of universally properly covers relation and the Theorem 5 [4].

From Theorem 3.2, we have the following hierarchy:
Fig. 8. The "Universally Properly Covers" Hierarchy of the Subdomain-based Criteria
Chapter 4

A New Version of $IP/O_2$-chains Coverage Criteria

Like required $k$-tuples criterion, one of the original objectives of $IP/O_2$-chains coverage criterion is to refine subdomains so that the $du$-pairs in loops will be examined more than once if the loop containing that $du$-pair iterates more than once. It has been proved later by Frankl and Weyuker in Lemma 3 [3] that to refine subdomains related to a subdomain-based criterion will increase the fault detecting ability of that criterion. But, as the definition of required $k$-tuples criterion introduces non-executable $k$-$dr$ interactions, the definition of $IP/O_2$-chains coverage criterion introduces non-executable $IP/O_2$-chains, i.e. the $IP/O_2$-chains that can not be covered by any executable complete path. More precisely, there are two problems we have to consider

1. How do we identify the non-executable $IP/O_2$-chains
2. How do we treat these non-executable $IP/O_2$-chains

These problems have also been noted by Ntufos [13] and by Frankl and Weyuker [2]. Since the problem of determining whether or not an executable complete path that covers an $IP/O_2$-chain exists is, in general, undecidable, some $IP/O_2$-chains will have to be considered at the test data generation phase. In many cases, symbolic execution [9] may be useful in determining whether or not an $IP/O_2$-chain is executable. The solutions for treating those $k$-$dr$ interactions that can not be covered by any executable complete path
given both in [13] and [4] are to discard them simply at the test data generation phase. If we were to adopt this solution and simply discard these non-executable $IP/O_2$-chains at the test data generation phase, then we would lose some chances to refine some subdomains. Even worse, some executable $du$-pairs on these $IP/O_2$-chains would never be exercised during testing. A solution for required $k$-tuples criterion given in the definition of the required $k$-tuples* criterion is to require that all $l-dr$ interactions be exercised for $l \leq k$. This results in that every executable $du$-pair will be covered in the test suite even all required $k-dr$ interactions that can not be covered by any executable complete path are discarded. However, we will see later by an example that this solution produces duplicate subdomains. So, we will not follow this counter-productive way to improve $IP/O_2$-chains coverage criterion. Roughly speaking, our approach will be the following:

1. We will not introduce any additional requirement for original $IP/O_2$-chains coverage criterion.

2. We will not simply throw away these non-executable $IP/O_2$-chains. Instead, we will improve them to be executable and therefore to use them to refine subdomains.

3. We will utilize the concept of prefix to reduce the duplications of subdomains.

4.1 Definition Of New $IP/O_2$-chains Coverage Criterion

In this section, we define the new $IP/O_2$-chains coverage criterion to achieve the objectives mentioned at the end of the last section. But first, we modify the definition of $df$-chains given in Chapter 2 to define executable $df$-chains.

41
Definition 4.1

1. A length 1 df-chain is an executable du-pair. On the other hand, every executable du-pair is a length 1 df-chain.

2. A length m df-chain \((m > 1)\) is a sequence of \(m\) executable du-pairs such that
   
   a) The first \(m-1\) executable du-pairs form a length \(m-1\) df-chain.
   
   b) If the use of a variable in \((m-1)\)-th executable du-pair occurs at node \(n_m\) then the definition of a variable in \(m\)-th executable du-pair occurs at node \(n_m\).
   
   c) There exists at least one executable complete path to cover all these \(m\) executable du-pairs simultaneously.

A length \(m\) df-chain \(c\) is denoted by \((d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_m}^{x_m}, u_{n_{m+1}}^{x_m})\) and the length of \(c\) is denoted by \(l(c)\).

From Definition 4.1 we know that every df-chain is executable, i.e. there exists an executable complete path to cover it.

Because some du-pairs may occur several times in a df-chain, in order to distinguish these du-pairs in a df-chain, we introduce the concept of segment.

Given a df-chain \(c = (d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_m}^{x_m}, u_{n_{m+1}}^{x_m})\), if \(s = u_{n_k}^{x_{k-1}} d_{n_k}^{x_k} u_{n_{k+1}}^{x_k} \ldots u_{n_r}^{x_{r-1}} d_{n_r}^{x_r}\), where \(2 \leq k \leq r \leq m\) then \(s\) is called a segment of \(c\).

It should be noted that the multiple occurrences of \(u_j^x d_j^y\) in a df-chain indicates that the df-chain traverses a loop in the given program. However, the reverse is not always true, i.e. there may exist a df-chain that traverses a loop in a program but every \(u_j^x d_j^y\) occurs only once in it.
Definition 4.2 A cycle in a df-chain is a segment of the df-chain such that one \( u^x_j d^y_j \) occurs at the beginning and at the end of the segment; and \( u^x_j d^y_j \) does not occur in any other place in the segment.

A cycle may occur several times in a df-chain. Some of them may be deleted from the df-chain without changing the executability of the df-chain while others may have to be there for the df-chain to be executable. The original IP/O2-chains coverage criterion requires at most one occurrence of a cycle in a df-chain to be considered. This results in non-executable df-chains. For instance, program D in Section 3.4 has the following IP/O2-chains:

\[
[d_1^y, u_1^y d_1^y, u_1^y d_1^y, u_5^y d_5^y, p_{(6,7)}],
[d_1^y, u_1^y d_1^y, u_4^y d_4^y, u_5^y d_5^y, p_{(6,8)}],
[d_1^y, u_4^y d_4^y, u_4^y d_4^y, u_5^y d_5^y, u_8^y],
[d_2^i, u_2^i d_2^i, u_4^i d_4^i, u_5^i d_5^i, p_{(6,7)}],
[d_2^i, u_4^i d_4^i, u_4^i d_4^i, u_5^i d_5^i, u_8^y] \]

But, none of them is executable. Discarding these non-executable IP/O2-chains will result in du-pairs that occur on them not been covered. Therefore, we have to overcome this weakness of the original IP/O2-chains coverage criterion in defining a new version of IP/O2-chains coverage criterion. First, we introduce the following concepts:

Definition 4.3 A df-chain \( c \) is irreducible if one of the following conditions is satisfied:

1. There is no cycle in \( c \).

2. Any attempt to decrease the number of occurrences of any cycle that occurs in \( c \) will disqualify \( c \) from being a df-chain.

43
Definition 4.4

1. Let $c$ and $c'$ be df-chains. We say that $c'$ is deduced from $c$ if $c'$ can be obtained from $c$ by increasing the number (which may be zero) of occurrences of a cycle.

2. A cycle in an irreducible df-chain $c$ is said to have height 1. If $c'$ is a df-chain deduced from $c$ by increasing the number of occurrences of a cycle by $n$ then we say that the cycle has height $n + 1$ in $c'$.

3. A df-chain $c$ is said to be height 1 iff $c$ is irreducible. If $c$ is deduced from another df-chain, then the maximal height of the cycles in $c$ is called the height of the df-chain. The height of $c$ is denoted by $h(c)$.

For instance, if $c_0 = (d_1^{x_2}, u_2^{x_2})$ is a $du$-pair and $c = (d_1^{x_2}, u_3^{x_2} d_3^{x_3}, u_4^{x_3})$ is a df-chain then $c_0$ and $c$ are irreducible. Therefore both $c_0$ and $c$ are of height 1. But, $c' = (d_1^{x_2}, u_3^{x_2} d_3^{x_3}, u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, u_3^{x_2} d_3^{x_3}, u_4^{x_3})$ is deduced from $c$. The height of the cycle $u_3^{x_2} d_3^{x_3}, u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, u_3^{x_2} d_3^{x_3}$ is 3 in $c'$ and the height of cycle $u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, u_3^{x_2} d_3^{x_3}$ is 2 in $c'$. Hence, the height of $c'$ is 3. If $c_1 = (d_1^{x_2}, u_3^{x_2} d_3^{x_3}, u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, p_5^{x_2}(2,6))$ is a df-chain and $(d_1^{x_2}, u_3^{x_2} d_3^{x_3}, u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, p_5^{x_2}(2,6))$ is not a df-chain then $c_1$ is irreducible. Thus the height of the cycle $u_3^{x_2} d_3^{x_3}, u_4^{x_3} d_4^{x_4}, u_5^{x_4} d_5^{x_5}, u_3^{x_2} d_3^{x_3}$ is 1 in $c_1$ although it occurs twice in $c_1$. But the height of $c_1$ is also 1 from Definition 4.4 because $c_1$ is irreducible.

Definition 4.5

1. An IP/O-chain is a df-chain such that the first definition of a variable on the df-chain is an input of that variable and the last use of a variable is a $p$-use or an output of that variable. An IP/O-chain is of height $n$ if it is a df-chain of height $n$.

2. An IP/O$_n$-chain is a IP/O-chain of height $n$. 

44
Because every \( IP/O_{n} \)-chain is a \( df \)-chain and each \( df \)-chain is executable, every \( IP/O_{n} \)-chain is executable by Definition 4.1.

Generally speaking, a Boolean expression will produce a large number of \( p \)-uses in which many of them may simply produce duplications of subdomains. In order to reduce the duplication of subdomains, we utilize the concept of prefix of \( IP/O \)-chains which was introduced by Schoot and Ural in [15].

**Definition 4.6** An \( IP/O_{j} \)-chain \( c_{1} \) of length \( m \) which ends with a \( p \)-use is a prefix of an \( IP/O_{k} \)-chain \( c_{2} \) if the following conditions are satisfied:

1. The sequence of the first \( m \) - 1 \( dt \)-pairs in \( c_{1} \) and \( c_{2} \) and the definition in the \( m \)-th \( dt \)-pair in \( c_{1} \) and \( c_{2} \) are the same, i.e.

\[
c_{1} = (d_{n_{1}}^{x_{1}}, u_{n_{2}}^{x_{1}}, d_{n_{2}}^{x_{2}}, u_{n_{3}}^{x_{2}}, \ldots, d_{n_{m}}^{x_{m}}, p_{\{n_{m+1}, n_{m+2}\}^{x_{m}}})
\]
\[
c_{2} = (d_{n_{1}}^{x_{1}}, u_{n_{2}}^{x_{1}}, d_{n_{2}}^{x_{2}}, u_{n_{3}}^{x_{2}}, \ldots, d_{n_{m}}^{x_{m}}, u_{r_{m+1}}^{x_{m}} d_{r_{m+1}}^{x_{m+1}}, \ldots, u_{r_{s}}^{x_{s}})
\]

2. The activating paths of \( c_{2} \) have the following form

\[
(n_{1} \ldots n_{2} \ldots n_{m-1} \ldots n_{m+1} \ldots n_{m+2} \ldots r_{m+1} \ldots r_{m+2} \ldots r_{s-1} \ldots r_{s}),
\]

where \( n_{j} \ldots n_{j+1} \) and \( r_{k} \ldots r_{k+1} \) are \( def \)-clear paths with respect to \( x_{j} \) and \( x_{k} \) respectively for \( j = 1, \ldots, m - 1 \) and \( k = m + 1, \ldots, s - 1 \); \( n_{m} \ldots n_{m+1} \ldots n_{m+2} \ldots r_{m+1} \) is \( def \)-clear path with respect to \( x_{m} \), i.e. all activating paths of \( c_{2} \) traverse the edge \( \{n_{m+1}, n_{m+2}\} \).

**Proposition 4.1** If \( IP/O_{j} \)-chain \( c_{1} \) is a prefix of \( IP/O_{k} \)-chain \( c_{2} \) then \( D(c_{2}) \subseteq D(c_{1}) \).

**Proof:** Let

\[
c_{1} = (d_{n_{1}}^{x_{1}}, u_{n_{2}}^{x_{1}}, d_{n_{2}}^{x_{2}}, u_{n_{3}}^{x_{2}}, \ldots, d_{n_{m}}^{x_{m}}, p_{\{n_{m+1}, n_{m+2}\}^{x_{m}}})
\]

45
\[ c_2 = (d_n^{x_1}, u_n^{x_1} d_n^{x_2}, u_n^{x_2} \ldots d_n^{x_m}, u_n^{x_m} d_n^{x_{m+1}}, \ldots, u_n^{x_s}) \]

If \( c_1 \) is a prefix of \( IP/O_k \)-chain \( c_2 \) then the activating paths of \( c_1 \) and \( c_2 \) are of form

\[
(n_1 \ldots n_2 \ldots n_m \ldots n_{m+1} \ldots n_{m+2})
\]

\[
(n_1 \ldots n_2 \ldots n_m \ldots n_{m+1} \ldots n_{m+2} \ldots r_{m+1} \ldots r_{m+2} \ldots r_{s-1} \ldots r_s)
\]

respectively. If \( d \) is an input data such that the activating path \((n_1 \ldots n_2 \ldots n_m \ldots n_{m+1} \ldots n_{m+2} \ldots r_{m+1} \ldots r_{m+2} \ldots r_{s-1} \ldots r_s)\) will be traversed, then \( d \) is also an input data such that the activating path \((n_1 \ldots n_2 \ldots n_m \ldots n_{m+1} \ldots n_{m+2})\) will be traversed. Hence \( D(c_2) \subseteq D(c_1) \).

From the proof of Proposition 4.1 we know that every activating path of \( c_1 \) is a subpath of an activating path of \( c_2 \) and every activating path of \( c_2 \) must contain a subpath that is an activating path of \( c_1 \) because they have the same sequence of the first \( m - 1 \) \( du \)-pairs and \( c_2 \) also traverses the edge of the \( p \)-use in last \( du \)-pair of \( c_1 \).

Now we can define new version of \( IP/O_n \)-chains coverage criterion. The objectives are to release those \( IP/O \)-chains that may have the same subdomains as the other \( IP/O \)-chains or their subdomains are the unions of subdomains of other \( IP/O \)-chains.

**Definition 4.7** The **new \( IP/O_n \)-chains coverage criterion** requires that following \( IP/O_k \)-chains will be covered at least once, where \( k \leq n \):

1. every \( IP/O_k \)-chain that ends with an output of a variable
2. every \( IP/O_k \)-chain \( c \) that ends with a \( p \)-use of a variable and satisfies one of the following conditions:
   2.1. \( c \) is not a prefix of any other \( IP/O_j \)-chain, where \( j \leq n \).
2.2. \( c \) is a prefix of another \( IP/O_j \)-chain and there exists at least one executable complete path \( p \) such that \( p \) covers \( c \) and \( c \) is not a prefix of any \( IP/O \)-chain covered by \( p \), where \( j \leq n \).

We denote the applicable new \( IP/O_n \)-chains coverage criterion by (new \( IP/O_n \)-chains)* and the subdomain-based new \( IP/O_n \)-chains coverage criterion by (new \( IP/O_n \)-chains)*.

It should be noted that 2.2 in Definition 4.7 is equivalent to say that if an \( IP/O_k \)-chain \( c \) ending with a \( p \)-use of a variable is a prefix of another \( IP/O_k \)-chain such that for every executable complete path \( p \) that covers \( c \), there exists an \( IP/O \)-chain \( c_1 \) on \( p \) and \( c \) is a prefix of \( c_1 \) then \( c \) can be ignored. But, it will be very expensive if one uses this fact to check that if a \( IP/O \)-chain need to be covered in a test suite for a given program. Actually, we need not to check every executable complete path \( p \) that covers \( c \) to see if \( c \) is a prefix of some \( IP/O \)-chain on \( p \). Suppose \( c = (d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_m}^{x_m}, p_{(n_{m+1}, n_{m+2})}^{x_m}) \) is a prefix of another \( IP/O_k \)-chain, then we use algorithm A given below to determine if \( c \) need to be covered. But first we define the extension of a \( df \)-chain.

**Definition 4.8** Given two \( df \)-chains \( c_1 = (d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_m}^{x_m}, u_{n_m}^{x_m}) \) and \( c_2 = (d_{k_1}^{x_1}, u_{k_2}^{x_1}, d_{k_2}^{x_2}, u_{k_3}^{x_2}, \ldots, d_{k_r}^{x_r}, u_{k_r}^{x_r}) \), \( c_2 \) is called a tail extension of \( c_1 \) (or \( c_1 \) is tail-extended to \( c_2 \)) if \( m < r \) and \( n_j = k_j \), for \( j = 1, \ldots, m \). Given two \( df \)-chains \( c_3 = (d_{n_1}^{x_1}, u_{n_2}^{x_1}, d_{n_2}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_{i-1}}^{x_{i-1}}, u_{n_i}^{x_i}, d_{n_i}^{x_{i+1}}, u_{n_{i+1}}^{x_{i+1}}, \ldots, d_{n_r}^{x_r}, u_{n_r}^{x_r}) \) and \( c_4 = (d_{k_1}^{x_1}, u_{k_2}^{x_1}, d_{k_2}^{x_2}, u_{k_3}^{x_2}, \ldots, d_{k_{r-1}}^{x_{r-1}}, u_{k_r}^{x_r}) \), \( c_4 \) is called a head extension of \( c_3 \) (or \( c_3 \) is head-extended to \( c_4 \)) if \( 1 < i \) and \( n_j = k_j \), for \( j = i, i + 1, \ldots, r \). If a \( df \)-chain \( c_1 \) is tail or head-extended to \( c_2 \) then we say that \( c_1 \) is extended to \( c_2 \). Also, if a \( df \)-chain \( c_2 \) is tail or head extension of \( c_1 \) then we say that \( c_2 \) is extension of \( c_1 \).
Note that \( c = (d_{n_1}^{x_1}, u_{n_2}^{x_2}, d_{n_3}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_{m-1}}^{x_{m-1}}, u_{n_m}^{x_m}, d_{n_{m+1}}^{x_{m+1}}, u_{n_{m+2}}^{x_{m+2}}) \) being a prefix of an IP/O-chain on \( p \) is equivalent to that \( c_0 = (d_{n_1}^{x_1}, u_{n_2}^{x_2}, d_{n_3}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_{m-1}}^{x_{m-1}}, u_{n_m}^{x_{m-1}}) \) can be tail-extended to an IP/O-chain on \( p \) with \( (d_{n_m}^{x_m}, u_{r}^{x_m}) \) as the first du-pair to add to \( c_0 \) for some node \( r \) on \( p \).

**Algorithm A:**

\[
c_0 := (d_{n_1}^{x_1}, u_{n_2}^{x_2}, d_{n_3}^{x_2}, u_{n_3}^{x_2}, \ldots, d_{n_{m-1}}^{x_{m-1}}, u_{n_m}^{x_{m-1}});
B := \{ \text{executable subpaths } (n_{m+2}, n_0) \};
c\text{\_need\_to\_be\_covered} := \text{false};
\]

while \( (B \not<\not> \text{empty}) \text{ and } (\not\not c\text{\_need\_to\_be\_covered} ) \) do

\[
\text{remove one subpath from } B;
\]

\[
c_1 := c_0;
D := \{ \text{du-pairs of form } (d_{n_m}^{x_m}, u_{n_0}^{x_1}) \text{ whose def-clear path w.r.t. } x_m \text{ terminates at some node } n_0 \text{ on the subpath and } c_1 \text{ can be tail-extended by using t he du-pair} \};
\]

while \((D \not<\not> \text{empty} \text{ and } c_1 \not<\not> \text{IP/O-chain})\) do

\[
\text{remove one du-pair from } D;
\]

\[
c_1 := \text{tail-extension of } c_0 \text{ obtained by using the du-pair};
\]

if \( c_1 \not<\not> \text{IP/O-chain} \) then

\[
\text{repeat}
\]

\[
\text{tail-extend } c_1 \text{ on one subpath starting from node } n_0 \text{ (depth first)};
\]

if the tail-extension of \( c_1 \) traverses into a loop then exit the loop as soon as the tail-extension is executable;

48
(* since if there is no use of a variable in the first iteration then neither in other iterations *)

until an IP/O-chain is formed or the exit node t is reached;

endif;
endwhile;

if c₁ <> IP/O-chain then

c_need_to_be_covered := true;

endif;
endwhile;

It should be also noted that a particular example of an IP/Oₖ-chain c that satisfies 2.1 in Definition 4.7 is the IP/Oₖ-chain that ends with a p-use of a variable and steps toward a constant output (there is no IP/O-chain that has c as its prefix on that executable complete path).

Next, we give a very simple program and apply (new IP/O₂-chains)⁷ to it to show that the objectives above are achieved.

Example 4.1 Let F be a program with the following flowgraph representation
Fig. 9 Program $F$ and Its Flowgraph

Let $x, y, z = 1, 2$ be the inputs of program $F$. If $(a, b, c)$ denotes the input $x = a$, $y = b$ and $z = c$ then the executable $IP/O_1$-chains of $F$ and their subdomains are given in Table 11:

Table 11: $IP/O_1$-chains and their subdomains in program $F$

<table>
<thead>
<tr>
<th>ID</th>
<th>$IP/O_1$-chains</th>
<th>Subdomains</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$[d^x_1, p^x_{(8, 9)}]$</td>
<td>${(1,1,1), \ (1,1,2)}$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$[d^x_1, p^x_{(8, 10)}]$</td>
<td>${(2,1,1), \ (2,1,2)}$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$[d^x_1, u^x_{10}]$</td>
<td>${(2,1,1), \ (2,1,2)}$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$[d^y_2, u^y_3, d^x_5, p^y_{(7, 8)}]$</td>
<td>${(1,1,1), \ (2,1,1)}$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$[d^y_2, u^y_3, d^y_5, p^y_{(7,4)}]$</td>
<td>${(1,2,1), (2,2,1)}$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$[d^y_2, u^y_4, d^y_5, u^y_0]$</td>
<td>${(1,1,1)}$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$[d^y_2, p^y_{(7,8)}]$</td>
<td>${(1,1,2), (2,1,2)}$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$[d^z_2, p^z_{(7,4)}]$</td>
<td>${(1,2,2), (2,2,2)}$</td>
</tr>
<tr>
<td>$c_9$</td>
<td>$[d^z_3, p^z_{(4,5)}]$</td>
<td>${(1,1,1), (2,1,1), (1,2,1), (2,2,1)}$</td>
</tr>
<tr>
<td>$c_{10}$</td>
<td>$[d^z_3, p^z_{(4,6)}]$</td>
<td>${(1,1,2), (2,1,2), (1,2,2), (2,2,2)}$</td>
</tr>
<tr>
<td>$c_{11}$</td>
<td>$[d^z_3, u^z_0]$</td>
<td>${(1,1,2), (2,1,2), (1,2,2), (2,2,2)}$</td>
</tr>
</tbody>
</table>

Among the IP/O-chains above, $c_2$ is a prefix of $c_3$, $c_4$ is a prefix of $c_6$ and $c_{10}$ is a prefix of $c_{11}$. According to definition 4.7 we can only discard $c_2$ and $c_{10}$. For example $c_2$ can be discarded because there are only two executable complete paths $p_1 = (s, 1, 2, 3, 4, 5, 7, 8, 10, t), p_2 = (s, 1, 2, 3, 4, 6, 7, 8, 10, t)$ that cover $c_2$ and $c_2$ is a prefix of $c_3$ that is on both $p_1$ and $p_2$. So, $c_2$ does not satisfy 2 in Definition 4.7. But $c_4$ satisfies 2.2 in Definition 4.7 because the executable complete path $p_1$ covers $c_4$, and $c_4$ is not a prefix of any IP/O-chain on $p_1$. Therefore $c_4$ must be included in the set of IP/O-chains to be covered.

Note that the subdomains of $c_2$ and $c_{10}$ are the unions of other subdomains. We can release $c_2$ and $c_{10}$ without any problem. But the subdomain $\{(1,1,1), (2,1,1)\}$ of $c_4$ is not a union of any subdomains although it is a prefix of $c_6$. Therefore the requirement 2 in Definition 4.7 is necessary to achieve the objective of the new IP/O2-chains coverage.
criterion. On the other hand, we will prove in Lemma 4.2.3 that the requirements in Definition 4.7 are also sufficient.

Obviously, \((new \ IP/O_2-chains)^*\) requires that \(c_1, c_3, c_4, c_5, c_6, c_7, c_8, c_9\) and \(c_{11}\) to be covered at least once in a test suite.

In the following example, we will see the differences between the new \(IP/O_2\)-chains and the original \(IP/O_2\)-chains. We will also see the applications of the original \(IP/O_2\)-chains coverage criterion, \((IP/O_2-chains)^*, (IP/O_2-chains)^8, (new \ IP/O_2-chains)^*\) and \((new \ IP/O_2-chains)^8\) to the same program.

**Example 4.2** Consider program \(D\) and its flowgraph given in Fig. 6 in Section 3.4. The original \(IP/O_2\)-chains, applicable \(IP/O_2\)-chains and the new \(IP/O_2\)-chains and their subdomains are given in Tables 12, 13 and 14 respectively:

<table>
<thead>
<tr>
<th>ID</th>
<th>ORIGINAL (IP/O_k)-CHAINS</th>
<th>SUBDOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_1)</td>
<td>({d_2^i, p_{(3,41)}^i})</td>
<td>all (y's)</td>
</tr>
<tr>
<td>(c_2)</td>
<td>({d_2^i, u_4^i d_4^i, p_{(3,41)}^i})</td>
<td>all (y's)</td>
</tr>
<tr>
<td>(c_3)</td>
<td>({d_2^i, u_4^i d_4^i, u_{42}^i d_{42}^i, p_{(3,41)}^i})</td>
<td>all (y's)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>({d_2^i, p_{(3,5)}^i})</td>
<td>empty</td>
</tr>
<tr>
<td>(q_2)</td>
<td>({d_2^i, u_4^i d_4^i, p_{(3,5)}^i})</td>
<td>empty</td>
</tr>
<tr>
<td>(q_3)</td>
<td>({d_2^i, u_4^i d_4^i, u_{42}^i d_{42}^i, p_{(3,5)}^i})</td>
<td>empty</td>
</tr>
</tbody>
</table>
Table 13: Applicable $IP/O_K$-chains in Program $D$ ($k \leq 2$)

<table>
<thead>
<tr>
<th>ID</th>
<th>APPLICABLE $IP/O_K$-CHAINS</th>
<th>SUBDOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$[d_2^i, p_{(3,41)}^i]$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$[u_{(4,2)}^i d_{(42)}^i p_{(3,41)}^i]$</td>
<td>all $y$'s</td>
</tr>
<tr>
<td>ID</td>
<td>NEW IP/O₂-CHAINS</td>
<td>SUBDOMAINS</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>c₁</td>
<td>([d_{2}^{i}, p_{(3,41)}^{i}])</td>
<td>all y's</td>
</tr>
<tr>
<td>c₂</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, p_{(3,41)}^{i}])</td>
<td>all y's</td>
</tr>
<tr>
<td>c₃</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, p_{(3,41)}^{i}])</td>
<td>all y's</td>
</tr>
<tr>
<td>c₄</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, p_{(3,5)}^{i}])</td>
<td>all y's</td>
</tr>
<tr>
<td>c₅</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{5}^{i}d_{5}^{y}, p_{(6,7)}^{y}])</td>
<td>y ≤ - 1</td>
</tr>
<tr>
<td>c₆</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{5}^{i}d_{5}^{y}, p_{(6,8)}^{y}])</td>
<td>y &gt; - 1</td>
</tr>
<tr>
<td>c₇</td>
<td>([d_{2}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i}, u_{5}^{i}d_{5}^{y}, u_{8}^{y}])</td>
<td>y &gt; - 1</td>
</tr>
<tr>
<td>c₈</td>
<td>([d_{1}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{5}^{y}d_{5}^{y}, p_{(6,7)}^{y}])</td>
<td>y ≤ - 1</td>
</tr>
<tr>
<td>c₉</td>
<td>([d_{1}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{5}^{y}d_{5}^{y}, p_{(6,8)}^{y}])</td>
<td>y &gt; - 1</td>
</tr>
<tr>
<td>c₁₀</td>
<td>([d_{1}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{41}^{y}d_{41}^{y}, u_{5}^{y}d_{5}^{y}, u_{8}^{y}])</td>
<td>y &gt; - 1</td>
</tr>
</tbody>
</table>

It should be noted that c₄, c₅, c₆ and c₇ are IP/O₁-chains while c₃ is IP/O₂-chain because the number of the occurrences of the cycle \( u_{42}^{i}d_{42}^{i}, u_{42}^{i}d_{42}^{i} \) in c₄, c₅, c₆ and c₇ can not be decreased, but it can be decreased in c₃.
The original \( IP/O_2 \)-chains coverage criterion requires that \( c_1, c_2, c_3, q_1, ..., q_{15} \) to be covered at least once. \((IP/O_2\text{-chains})^*\) requires that \( c_1, c_2, c_3 \) to be covered at least once.

Among the new \( IP/O_1 \)-chains above, \( c_1, ..., c_{66} \) and \( c_9 \) are prefixes of others. But, we claim that only \( c_5 \) need to be covered. Therefore, \((new IP/O_2\text{-chains})^*\) requires that \( c_5, c_7, c_8 \) and \( c_{10} \) to be covered at least once.

Since \( c_1 \) is a prefix of \( c_2 \) and \( c_2 \) is on the executable complete paths \( p_1 = (s, 1, 2, (3, 41, 42), 3, 5, 6, 7, t) \) and \( p_2 = (s, 1, 2, (3, 41, 42), 5, 3, 5, 6, 8, t) \) that cover \( c_1; c_2 \) is a prefix of \( c_3 \) and \( c_3 \) is on both \( p_1 \) and \( p_2 \) that cover \( c_2; c_1 \) and \( c_2 \) do not satisfy the condition 2.2 in Definition 4.7. Similarly, \( c_3 \) is a prefix of \( c_4 \) and \( c_4 \) is on both \( p_1 \) and \( p_2 \) which imply that \( c_3 \) does not satisfy 2.2 in Definition 4.7. There are only two executable complete paths \( p_1 \) and \( p_2 \) that cover \( c_4, c_5 \) and \( c_6 \) are on \( p_1 \) and \( p_2 \), respectively. \( c_4 \) is a prefix of \( c_5 \) and \( c_6 \). Thus, \( c_4 \) does not satisfy 2.2 in Definition 4.7. Finally, \( c_6 \) is a prefix of \( c_7 \) and \( c_9 \) is a prefix of \( c_{10} \). But, \( p_1 \) is the only executable complete path that covers \( c_6 \) and \( p_2 \) is the only executable complete path that covers \( c_9 \). \( c_7 \) is on \( p_1 \) and \( c_{10} \) is on \( p_2 \) imply that \( c_6 \) and \( c_9 \) do not satisfy 2.2 in Definition 4.7. So, the claim is correct.

Let \( Sub_{IP/O_2} \) (\( Sub_{N-IP/O_2} \)) denote the subdomains of \((IP/O_2\text{-chains})^*\) and \((new IP/O_2\text{-chains})^*\) respectively. Then

\[
Sub_{IP/O_2} = \{ \{ all \ y's \} , \{ all \ y's \} , \{ all \ y's \} \}
\]
\[
Sub_{N-IP/O_2} = \{ \{ y \leq -1 \} , \{ y > -1 \} , \{ y \leq -1 \} , \{ y > -1 \} \}.
\]

Now, we can see from Example 4.2 why we do not require that every \( IP/O_k \)-chain that ends with a \( p \)-use of a variable to be covered at least once in Definition 4.7. The selection of \( IP/O_k \)-chains in 2 of Definition 4.7 minimizes duplication of subdomains. For instance, in the example above, \( IP/O_1 \)-chains \( c_6 \) and \( c_7 \) have exactly the same subdomains,
while the subdomain of \( c_4 \) is a union of the subdomains of \( c_5 \) and \( c_7 \). Therefore, it is redundant to require that \( c_4 \) and \( c_6 \) to be covered at least once during testing.

On the other hand, the purpose of 2.1 in Definition 4.7 is to capture indirectly the effects of a program input on a constant output. Therefore, it is necessary to require that each \( I P/O_1 \)-chain in 2.1 be covered at least once during testing.

We also can see from Example 4.2 that every non-executable \( I P/O \)-chain \((q_1, ..., q_{15})\) required by the original \( I P/O_2 \)-chains coverage criterion is modified to an executable \( I P/O \)-chain \((c_4, ..., c_{10})\) in the new version of \( I P/O_2 \)-chains. The subdomains are refined in the new \( I P/O_2 \)-chains coverage criterion without introducing too many duplications of subdomains. In fact, this can also be seen from the following Example 4.3 where the program has no non-executable \( I P/O \)-chains.

**Example 4.3** Consider program \( E \) and its flowgraph given in Fig. 7 in Section 3.4.

Because there is no non-executable \( I P/O \)-chains in Program \( E \), the original \( I P/O_2 \)-chains, applicable \( I P/O_2 \)-chains and new \( I P/O_2 \)-chains are the same. They were given in Tables 2 and 3. The subdomains of \( I P/O_1 \)-chains and \( I P/O_2 \)-chains were given in Tables 5 and 6.

The subdomains of \( du \)-pairs were given in Table 4. Because required \( k \)-tuples\(^+ \) criterion insists that every \( du \)-pair is covered at least once, all subdomains in Table 4 belong to the subdomains of required \( k \)-tuples\(^+ \) criterion. As we can see from there that there are too many duplications.

But, the new \( I P/O_2 \)-chains coverage criterion is more efficient. In Tables 5 and 6, \( I P/O_1 \)-chain \([d_{1a}^n, p_{(2,4a)}^n] \) is a prefix of \([d_{1a}^n, u_{4a}^n d_{4a}^n p_{(5,6a)}^n] \) and \([d_{1a}^n, u_{4a}^n d_{4a}^n p_{(5,6a)}^n] \) which
are on the executable complete paths that cover \([d_{i6}^{i}p_{(2,4a)}^{i}]\). The IP/O\(_1\)-chain \([d_{i4b}^{i}p_{(5,6a)}^{i}]\) is a prefix of \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\) and \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\) which are on the executable complete paths that cover \([d_{i4b}^{i}p_{(5,6a)}^{i}]\). The IP/O\(_1\)-chain \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\) is a prefix of \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\) and \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\) which are on the executable complete paths that cover \([d_{i4b}^{i}u_{i6d}^{i}d_{i6d}^{i}p_{(5,6a)}^{i}]\). Thus, we can remove subdomains \([2, 3, \ldots, 9]\), \([3, 4, \ldots, 9]\) and \([4, 5, \ldots, 9]\) which are refined by other subdomains.

According to the definition of new IP/O\(_2\)-chains coverage criterion, there is no special requirement for the following IP/O-chains:

1. the first definition of an IP/O-chain occurs in a loop
2. the last use of an IP/O-chain occurs in a loop
3. the first definition or the last use of IP/O-chain occurs within more than one loop

Whereas required \(k\)-tuples\(^+\) criterion produces two required elements that specify two distinct iteration counts for the loop considered.

### 4.2 Comparison of New IP/O\(_2\)-chains Coverage and All-uses Criteria

In this section, we assume that all programs satisfy the requirements given in Section 3.1. In addition, we require the program under test to satisfy: f) \(NFAUU\) property: every feasible path from the entry node traversing a node having definition of a variable \(v\) must reach a node having a use of \(v\) (if programs do not satisfy this property, then may have the possibility of defining an unused variable).
As we have seen in the hierarchies given in Chapter 2 and Chapter 3 respectively that \((IP/O_2\text{-chains})^*\) is not as good as \((all\text{-}uses)^*\) and \((required\ k\text{-}tuples^*)^*\) criterion for some programs and their specifications under the strictly includes relation and the universally properly covers relation. In this section, we show that \((new\ IP/O_2\text{-chains})^*\) strictly includes \((all\text{-}uses)^*\) in Theorem 4.1. We also compare \((new\ IP/O_2\text{-chains})^*\) with \((all\text{-}uses)^*\) under measure \(M\) in Theorem 4.2 and its corollaries. But first we prove the following lemma.

**Lemma 4.2.1** Let \(c\) be an arbitrary df-chain in a given program \(P\) and its flowgraph \(G(V,E)\). If \(p\) is an executable complete path that covers \(c\) then \(c\) can be extended to an IP/O-chain covered by \(p\).

**Proof:** Suppose the lemma is not true for a given program \(P\) and its flowgraph \(G(V,E)\). We will show that this will result in a contradiction. Let \(S\) be the set of all df-chains that cannot be extended to an IP/O-chain in \(G(V,E)\) for some executable complete paths covering them. Then \(S\) is not empty. Because we assume that the input domain of program \(P\) is finite, the df-chains in \(G(V,E)\) is finite. Hence, \(S\) is finite and the df-chain that has maximal length in \(S\) exists. Let \(c = (d_m^x, ..., d_n^x)\) be a df-chain in \(S\) with maximal length. Because \(c\) cannot be extended to an IP/O-chain covered by an executable complete path \(p\), \(c\) itself is not an IP/O-chain. So, \(d_m^x\) is not an input of variable \(x\) or \(d_n^x\) is neither an output of variable \(y\) nor a \(p\)-use of variable \(y\).

Case 1: \(d_m^x\) is not an input of variable \(x\)

If \(d_m^x\) is not an input of variable \(x\) then \(x\) must be defined by using variable \(v\) at node \(m\). Since the executable complete path \(p\) covers \(c\), \(p\) reaches node \(m\) which has a use of \(v\). Thus the variable \(v\) is defined at node \(l\) on \(p\) by NFAUD property. So, we have a df-
chain $c' = (d^y_i, u^y_m d^x_m, \ldots, u^y_n)$. Since the length of $c'$ is greater than the length of $c$ and $c$ is the $df$-chain in $S$ with maximal length, $c' \notin S$. Therefore, $c'$ can be extended to an $IP/O$-chain covered by $p$. But $c$ is head-extended to $c'$. So $df$-chain $c$ can be extended to an $IP/O$-chain covered by $p$. This is a contradiction.

Case 2: $u^y_n$ is neither an output of variable $y$ nor a $p$-use of variable $y$:

In this case, we can also deduce a contradiction by tail-extend $c$ to $c'$ whose length is greater than the length of $c$. Let $p$ be a complete path that covers $c$. If $u^y_n$ is not an output of variable $y$ nor a $p$-use of variable $y$, then $y$ is used to define some variable $z$ at node $n$ on path $p$, i.e. there exists a feasible path from the entry node to a node having a definition of variable $z$. Thus, by $NFAUU$ property, the feasible path reaches some use of $z$ via some $def$-clear path with respect to $z$. Hence, $c' = (d^y_i, u^y_m d^x_m, \ldots, u^y_n d^z_n, u^z_i)$ is a $df$-chain. Now we use the same argument as in Case 1 to deduce a contradiction. Thus the lemma is proved.

Note that there may exist an executable complete path $p$ that covers a $df$-chain $c$, but $c$ can not be extended to an $IP/O$-chain on $p$ if the program does not satisfy $NFAUU$ property. One example of this case is the executable complete path $(s, 1, 2, 3, 4, 6, t)$ that covers $df$-chain $c = (d^y_2, u^y_3)$ in Program $G$ given in Fig. 10. The program does not satisfy $NFAUU$ property because the feasible path $(s, 1, 2, 3, 4, 6)$ from the entry node traversing node 3 having a definition of variable $y$ can not reach a use of $y$. 

59
Because every du-pair is a df-chain from the definition of df-chains, Lemma 4.2.1 implies that every du-pair can be extended to an IP/O-chain c. However, if the height of c is greater than 2, then by removing some cycles in c we can get an IP/O-chain c' of the height less than 3. Moreover, c' is still an extension of the du-pair. Thus, we have proved the following

**Lemma 4.2.2** Every du-pair can be extended to an IP/O₁-chain or an IP/O₂-chain.

Now we can prove the first Theorem in this section.

**Theorem 4.1** (new IP/O₂-chains)* strictly includes (all-uses)*.

**Proof:** From Lemma 4.2.2, we know that (new IP/O₂-chains)* includes (all-uses)*. So, we need only to prove the strictness. Let us consider the program E and its flowgraph given in Section 3.4. The paths

\[ p₁ = (s, 1, 2, 3, t) \]
\[ p₂ = (s, 1, 2, 4, 5, t) \]
\[ p_3 = (s, 1, 2, 4, 5, 6, 5, 1) \]
\[ p_4 = (s, 1, 2, 4, 5, 6, 5, 6, 5, 1) \]

satisfy \((\text{all-uses})^*\) but do not satisfy \((\text{new \, IP/O2-chains})^*\). Therefore, \((\text{all-uses})^*\) does not include \((\text{new \, IP/O2-chains})^*\) (Note: this proof was given by Ural and Yang [16]. It works here because there is no non-executable IP/O-chains in the example). So, Theorem 4.1 is proved.

To compare \((\text{new \, IP/O2-chains})^*\) with \((\text{all-uses})^*\) under measure \(M\), we need the following lemma.

**Lemma 4.2.3** Let \(c_1\) be an IP/O-chain that ends with a \(p\)-use. For every executable complete path \(p\) that covers \(c_1\), if \(c_1\) is a prefix of an IP/O-chain on \(p\) then \(D(c_1) = \bigcup_{c \in I} D(c)\), where \(I = \{c \mid c\ \text{is an IP/O-chain such that} \ c_1 \ \text{is a prefix of} \ c\}\) and \(D(c)\) is the subdomain of \(c\).

**Proof:** From Proposition 4.1 we know that \(D(c) \subseteq D(c_1)\) if \(c \in I\). On the other hand, if \(x \in D(c_1)\) then there exists a complete path \(p\) such that \(p\) covers \(c_1\) during the application of \(x\). Because \(c_1\) is a prefix of an IP/O-chain \(c_3\) on \(p\) by the assumption, \(x \in D(c_3)\) by definition of subdomains. Hence, \(D(c_1) = \bigcup_{c \in I} D(c)\).

**Theorem 4.2** For every given program \(P\), there exists a number \(n\) such that \((\text{new \, IP/O}_n\text{-chains})^*\) covers \((\text{all-uses})^*\).

**Proof:** Let \(d\) be a du-pair and \(D(d)\) denote the subdomain of \(d\). We need to prove that \(D(d)\) is a union of subdomains of some IP/O\(_n\)-chains required in Definition 4.7. But first, we
show that $D(d)$ is a union of subdomains of some $IP/O_n$-chains. Let $I$ be the set of all $IP/O$-chains $c$ that are extensions of $d$, then $\bigcup_{c \in I} D(c) \subseteq D(d)$ by the definition of subdomains.

On the other hand, if $x \in D(d)$ then there exists a complete path $p$ that traverses $d$ during the application of $x$. From Lemma 4.2.1, there exists an $IP/O$-chain $c_0$ on $p$ such that $c_0$ is an extension of $d$. Thus $c_0 \in I$ and $x \in D(c_0)$. So, we have $D(d) = \bigcup_{c \in I} D(c)$.

Let $n$ be the maximal height of all $IP/O$-chains in $P$. Then $D(d)$ is a union of subdomains of some $IP/O_k$-chains in $P$, where $k \leq n$. If one $IP/O_k$-chain $c_1$ in $I$ is not required in Definition 4.7, then $c_1$ satisfies the condition in Lemma 4.2.3 and hence $D(c_1)$ is a union of subdomains $D(c_l)$ of some $IP/O_l$-chains, where $l \leq n$. We substitute these $D(c_l)$ for $D(c_1)$ and repeat if there still exist some $IP/O_k$-chain that are not required in Definition 4.7, where $k \leq n$. Because the number of $IP/O_k$-chains is finite in $P$, via finite steps we have that $D(d)$ is a union of subdomains of some $IP/O_k$-chains which are required in Definition 4.7, where $k \leq n$. Theorem is thus proved.

From the proof of Theorem 4.2, we know that a $D(c)$ occurs in the union iff $c$ is an extension of $d$. In other words, $d$ is a $du$-pair on $c$. Thus, the number of the occurrences of each $D(c)$ in all unions is the number of $du$-pairs on $c$ which is less or equal to the length of $c$. Therefore, we have

**Corollary 4.1** For a given program $P$ there exists a number $n$ such that for each $IP/O_j$-chain $c$ if one duplicates the subdomain of $c l(c)$ times, where $j \leq n$ and $l(c)$ is the length of $c$, then (new $IP/O_n$-chains)$^8$ properly covers (all-uses)$^8$.

From Theorem 3.1, we have

62
Corollary 4.2 For a given program $P$ there exists a number $n$ such that for each $IP/O_j$-chain $c$ if one duplicates the subdomain of $c$ $l(c)$ times, where $j \leq n$ and $l(c)$ is the length of $c$, then $(new \ IP/O_n-chains)^s$ is better than $(all-uses)^s$ under measure $M$.

Generally speaking, to find all $IP/O_n$-chains is more expensive than to find all du-pairs. But, this does not mean that one has possibility of being trapped in an infinite loop. Although $IP/O$-chains are defined based on $df$-chains, to apply $(new \ IP/O_n-chains)^s$ to a program we need not to find all $df$-chains of the given program first because we do not need $IP/O_j$-chains for $j > n$. The following algorithm determines the all $IP/O_n$-chains of a given program.

Algorithm B:

find the set $I$ of all du-pairs of the program;

$C := [du$-pairs that start with an input and not end with a p-use];$

$I := I - C;$

while $(C <> empty)$ do

for each du-pairs in $I$ do

    tail-extend the $df$-chains in $C$ that can be tail-extended by using the du-pair;

    if the tail extension of the $df$-chain is an $IP/O$-chain or the height of the tail-extension is greater than $n$ then delete it from $C$;

endfor;

endwhile
4.3 Comparison of New $IP/O_n$-chains Coverage and Required $k$-tuples+ Criteria

In this section, we compare $(new \ IP/O_n-chains)^s$ and $(required \ k-tuples^+)^s$ under measure $M$. It is trivial to see that $(new \ IP/O_n-chains)^*$ and $(required \ k-tuples^+)^*$ are incomparable in strictly includes relation (see Theorem 5 [16]). We prove the following theorem.

**Theorem 4.3** $(new \ IP/O_n-chains)^s$ and $(required \ k-tuples^+)^s$ are incomparable in universally properly covers relation.

Theorem 4.3 will be proved if we have proved the following two lemmas

**Lemma 4.3.1** For any given integer $k$, there exists a program $P$ and a specification $S$ such that $M(N-IP/O_2,P,S) > M(k-dr^+,P,S)$, where $N-IP/O_2$ and $k-dr^+$ denote $(new \ IP/O_n-chains)^s$ and $(required \ k-tuples^+)^s$, respectively.

**Proof:** For a given integer $k$, let's consider program $H$ and its flowgraph given in Fig. 11.
In program $H$, after first iteration $x_1 = 1 + \frac{k(k+1)}{2}$. After second iteration $x_1 = 1 + k(k+1)$, and after third iteration $x_1 = 1 + \frac{3k(k+1)}{2}$. So, only iterating more than 2 times the control flow may reach the node $k+4$. Thus the new IP/Oj-chains ($j \leq 2$) are:

**Table 15: New IP/Oj-chains in Program $H$ ($j \leq 2$)**

<table>
<thead>
<tr>
<th>ID</th>
<th>NEW IP/Oj-CHAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$[d_1^n, p_{(3,4)}^n]$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$[d_1^n, p_{(3,k+4)}^n]$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$[d_1^n, p_{(k+5,k+6)}^n]$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$[d_1^u, p_{(k+5,k+7)}^u]$</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$[d_2^{x_1}, p_{(3,4)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(3,k+4)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(k+5,k+6)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(k+5,k+7)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_9$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(3,k+4)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_{10}$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(k+5,k+6)}^{x_1}]$</td>
</tr>
<tr>
<td>$c_{11}$</td>
<td>$[d_2^{x_1}, (u_4^{x_1}d_4^{x_2}, u_5^{x_1}d_5^{x_3}, \ldots, u_{k+2}^{x_1}d_{k+2}^{x_k}, u_k^{x_1}d_{k+3}^{x_1}, p_{(k+5,k+7)}^{x_1}]$</td>
</tr>
</tbody>
</table>

It should be noted that the definitions of all other $x_j$ for $j \neq 1$ are not inputs of the program. Therefore, there are no IP/O-chains starting with one of the first definitions of these $x_j$. (new IP/O$_n$-chains) requires that $c_3$, $c_4$, $c_7$, $c_8$, $c_{10}$, $c_{11}$ to be covered at least once. If $n = 1, 2, \ldots, 1 + 2k(k + 1)$ are the input of the program and $S$ is a specification such that $n = 1 + \frac{3k(k+1)}{2}$ will cause an error, then the subdomains of $c_3$, $c_4$, $c_7$, $c_8$, $c_{10}$, $c_{11}$ are:

- $D(c_3) = \{1 + \frac{3k(k+1)}{2}, 1 + 2k(k + 1)\}$
- $D(c_4) = \{n \mid n \neq 1 + \frac{3k(k+1)}{2}, 1 + 2k(k + 1)\}$
- $D(c_7) = \{1 + \frac{3k(k+1)}{2}\}$
- $D(c_8) = \{n \mid n \neq 1 + \frac{3k(k+1)}{2}\}$
- $D(c_{10}) = \{1 + 2k(k + 1)\}$
- $D(c_{11}) = \{n \mid n \neq 1 + 2k(k + 1)\}$
The failure causing rate of $D(c_7)$ is 1. Therefore $M(N-IP/O_2, I, S) = 1$.

But, for any $l \leq k$ the $l$-dr interaction does not have subdomain $D(c_7)$. Since one and two iterations of the loop are non-executable and the maximal iteration (four) of the loop does not have subdomain $D(c_7)$, if we choose one iteration and four iterations of the loop to generate the test paths for these $l$-dr interactions whose first definition of a variable or last use of a variable in the loop, then they can not have subdomain $D(c_7)$. So, $M(k$-dr, $I, S) < M(N-IP/O_2, I, S)$. Hence $M(k$-dr, $I, S) < M(N-IP/O_2, I, S)$.  

From this example we can also see that the application of $(required\ k$-tuples$^+)^8$ produces too many duplications of subdomains to write them out. It also generates a considerable number of non-executable test paths.

**Lemma 4.3.2** For any given integer $n$, there exists a program $P$ and a specification $S$ such that $(new\ IP/O_n$-chains$)^8$ does not properly cover $(required\ 2$-tuples$^+)^8$.

**Proof:** For a given integer $k$, let's consider program $I$ and its flowgraph given in Fig. 12.

![Flowgraph](image)

**Fig. 12 Program $I$ and Its Flowgraph**

For this program, $(required\ 2$-tuples$^+)^8$ yields the same results as $(all$-uses$)^8$. Let $x = 1, 2$ be the inputs of the program $I$. The $du$-pairs and their subdomains are:
\[(d_1^x, u_2^x) \{1, 2\}, \quad (d_2^x, u_4^x) \{1\}, \quad (d_2^x, u_5^x) \{2\}, \]
\[(d_2^x, p_{(3,4)}^x) \{1\}, \quad (d_2^x, p_{(3,5)}^x) \{2\}.\]

So, the multi-set of subdomains of \((\text{required } 2\text{-tuples}^+)^s\) is

\[\text{Sub}_{2\text{-dr}^+} = \{ \{1,2\}, \{1\}^2, \{2\}^2 \},\]

where \(B^r\) denotes the set \(B\) occurs \(r\) times in the multi-set. But, \((\text{new } IP/O_n\text{-chains})^s\) requires that \([d_1^x, u_2^x d_2^x, u_4^x]\) and \([d_4^x, u_2^x d_2^x, u_5^x]\) to be covered at least once during testing. The multi-set of subdomains of \((\text{new } IP/O_n\text{-chains})^s\) is

\[\text{Sub}_{N\text{-IP}/O_n} = \{ \{1\}, \{2\} \}.\]

So, \((\text{new } IP/O_n\text{-chains})^s\) does not properly cover \((\text{required } 2\text{-tuples}^+)^s\) for program \(I\).}

Note that

1. In the example above, if we repeat \{1\} and \{2\} two times respectively then the deduced \((\text{new } IP/O_n\text{-chains})^s\) properly covers \((\text{required } 2\text{-tuples}^+)^s\) for program \(I\).

2. Because of Theorem 4.2, it is impossible to give other kinds of examples so that \((\text{new } IP/O_n\text{-chains})^s\) does not properly cover \((\text{required } 2\text{-tuples}^+)^s\). In other words, \((\text{new } IP/O_n\text{-chains})^s\) does not properly cover \((\text{required } 2\text{-tuples}^+)^s\) only because that:

   a) some subdomains do not duplicate enough,

   b) some subdomains which are refined by the others are missing.

But, it's these improvements that make \((\text{new } IP/O_n\text{-chains})^s\) more efficient than the other criteria.
To prove Theorem 4.3, we also need following trivial propositions

**Proposition 4.3.1** For any given program $P$ and its specification $S$, $M(N-IP/O_{n}, P, S) \leq M(N-IP/O_{n+1}, P, S)$, where $n = 1, 2, \ldots$.

**Proof:** From the Definition 4.7, we know that the multi-set of subdomains of $(\text{new } IP/O_{n}\text{-chains})^g$ is a multi-subset of the multi-set of subdomains of $(\text{new } IP/O_{n+1}\text{-chains})^g$, where $n = 1, 2, \ldots$. Since multi-set properly covers its multi-subset, $(\text{new } IP/O_{n+1}\text{-chains})^g$ properly covers $(\text{new } IP/O_{n}\text{-chains})^g$. Thus, $M(N-IP/O_{n}, P, S) \leq M(N-IP/O_{n+1}, P, S)$, where $n = 1, 2, \ldots$.

By the similar argument as in the proof of Proposition 4.3.1 we have,

**Proposition 4.3.2** For any given program $P$ and its specification $S$, $M(k-dr^+, P, S) \leq M((k+1)-dr^+, P, S)$, where $k = 2, 3, 4, \ldots$.

**Proof of Theorem 4.3:** From Lemma 4.3.1 and Proposition 4.3.1 we know that there is no integer $n$ such that $(\text{required } k\text{-tuples}^+)^g$ criterion universally properly covers $(\text{new } IP/O_{n}\text{-chains})^g$ for any given integer $k$. The Lemma 4.3.2 and Proposition 4.3.2 imply that $(\text{new } IP/O_{n}\text{-chains})^g$ does not universally properly cover $(\text{required } k\text{-tuples}^+)^g$ criterion for any integer $k$ and $n$.

Although $(\text{new } IP/O_{n}\text{-chains})^g$ and $(\text{required } k\text{-tuples}^+)^g$ are incomparable under universally properly covers relation, for each individual program we still have the following:

**Theorem 4.4** For a given program $P$, there exists a number $n$ such that for each $IP/O_j$-chain $c$ if one duplicates the subdomain of $c m(c)$ times, where $j \leq n$ and $m(c)$ is the total
number of df-chains on c, then (new IP/O_n-chains)^n properly covers (required k-tuples^+)^n for program P.

**Proof:** Using the same argument as in the proof of Theorem 4.3 we can show that the subdomain D(c) is a union of subdomains of some IP/O-chains required in Definition 4.7 for every df-chain c. Since the subdomain of each l-dr interaction is actually the subdomain of a df-chain for any l ≤ k if the subdomain of the l-dr interaction is not empty, every subdomain in (required k-tuples^+)^n is a union of subdomains in (new IP/O_n-chains)^n, where n is the maximal height of df-chains in program P. Let m(c) be the total number of df-chains on IP/O_j-chain c, where j ≤ n. Then the deduced (new IP/O_n-chains)^n properly covers subdomain based (required k-tuples^+)^n.
Chapter 5

Conclusion and Future Research Directions

We have defined a new version of $IP/O_n$-chains coverage criterion, and proved that:

(i) Applicable new $IP/O_2$-chains coverage criterion strictly includes applicable all-uses criterion; (ii) For any given program $P$, there exists a number $n$ such that subdomain-based new $IP/O_n$-chains coverage criterion covers subdomain-based all-uses criterion; (iii) For any given program $P$, there exists a number $n$ such that for each $IP/O_j$-chain $c$, if one duplicates the subdomain of $c$ $l(c)$ times, where $j \leq n$ and $l(c)$ is the length of $c$, then subdomain-based new $IP/O_n$-chains coverage criterion is better than subdomain-based all-uses criterion under measure $M$; (iv) Subdomain-based new $IP/O_n$-chains coverage criterion and subdomain-based required $k$-tuples+ criterion are incomparable in “universally properly covers” relation; (v) For any given program $P$, there exists a number $n$ such that for each $IP/O_j$-chain $c$, if one duplicates the subdomain of $c$ $m(c)$ times, where $j \leq n$ and $m(c)$ is the total number of $df$-chains on $c$, then subdomain-based new $IP/O_n$-chains coverage criterion properly covers the subdomain-based required $k$-tuples+ criterion.

A new method of handling non-executable $df$-chains is used. The method is also applicable to the improvement of other criteria that suffer from non-executable paths such as the required $k$-tuples+ criterion, etc.
The main costs of applying the subdomain-based \( IP/O_n \)-chains coverage criterion to a program come from: (i) determine \( IP/O_n \)-chains; (ii) selecting \( IP/O_n \)-chains that need to be covered. While the most difficult part of (i) and (ii) is actually to determine whether or not an \( IP/O_n \)-chain is executable, i.e., whether or not an executable complete path that covers the \( IP/O_n \)-chain exists. This is, in general, undecidable. Like required \( k \)-tuples criterion, to apply the new \( IP/O_n \)-chains coverage criterion, we can perform the symbolic execution technique given in [9]. But, it will be very interesting to study new methods to determine executable complete paths.

Because we are concentrated on the fault detecting ability of \( IP/O_n \)-chains coverage criterion, the complete comparisons of the new \( IP/O_n \)-chains coverage, all-uses and required \( k \)-tuples+ criteria need further study. Nevertheless, we conjecture that the new \( IP/O_n \)-chains coverage, all-uses and required \( k \)-tuples+ criteria have the same complexity.
References


