On Partitioning the Domain for Test Case Reusability

Yunwei Dong
Northwestern Polytechnic University
Xi'an, Shaanxi, CHINA
yunweidong@nwpu.edu.cn

M. F. Lau* and Si-yu Lin
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria, AUSTRALIA, 3122
{elau,slin}@swin.edu.au

Abstract

During the life time of software or even within the development stage, it is inevitable that the software needs to be modified. Test cases used before for the previous version could be reused in order to reduce the cost of regression testing. This paper proposes a regression testing methodology, called Partitioning of Domain Testing (PDT), to analyze the input domains of the previous version and the modified version so that test cases can be maximally reused. As a result, software test engineers can spend less effort in generating test cases to test the modified software. This methodology is different from traditional domain-based testing strategies in the sense that the partitioning is achieved by two supplementary perspectives, namely the specifications of the program and its testing criterion. First, the input domain is partitioned according to the specification and testing criterion of the program under test. Then, further refining methods are introduced to obtain 100% reusable test cases. We also illustrate the idea of PDT using a case study with 3 different testing criteria.

Keyword: Domain partitioning, specification-based testing, test case reusability

1. Introduction

For having a longer software lifetime and a better performance, software is always modified partly in maintenance phase or optimized in development phase. Many skills can be adopted. For example, we can replicate branch codes in conditional as well as unconditional branches so as to reduce the cost of comparing branch constructs to be executed [6, 7]. We can also use compilation tool to collect profile data to reorder the sequences of conditional branches of program so that the program executes faster [10]. In real-time system, in order to avoid unnecessary computations that occurred at different locations in a program, those original computations will be computed once and the subsequent computations could be replaced by variables or registers [5]. In order to have smaller code size of embedded programs, many conditions could be merged together by eliminating logically redundant conditions or avoiding further computations in conditional statements. For shortening the frequently executed paths or critical paths, a combined decision can be divided into more simpler decisions, or sometimes a redundant decision is inserted into the code so as to filter out the key conditions [4]. Furthermore, in order to adapt programs with good performance to new platforms, it may be necessary to recode these programs using other programming tools.

Regression testing is expensive. If test cases of the original software can be maximally reused in testing the modified version, the cost of regression testing can then be further reduced. Input domain is one of the essential attributes of program specifications, and it depends on the functional specification of the program but not on a particular implementation of the program. Test cases can be generated according to testing criterion-based partition, which divides the input-domain into "equivalence classes" based on testing strategy and software specification.

The focus of this paper is on test case reusability. We propose the partitioning of domain testing (PDT) for achieving coverage-based testing criteria on one hand, while maximizing the reusability of existing test cases. The input domain is partitioned into specification-based partition and a control-flow based partition based on the control-flow based testing criterion. Test cases are then selected to satisfy the control-flow based testing criteria to reveal the program coverage.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 discusses the definitions of specification-based and various control-flow based partitions. Section 4 presents our idea and results using a case study on various versions of a program. Section 5 concludes the paper.

*Corresponding author.

1550-6002/08 $25.00 © 2008 IEEE
DOI 10.1109/QSIC.2008.51

2. Related Work

Many studies focus on reusing test cases of original program for testing the modified program. Most of them focus on comparing the program structure (for example, the control-flow as well as the dataflow) of the modified version with the original program. They evaluate control-flow and dataflow coverage in testing to show the effectiveness of test cases [3].

In regression testing, test cases from the original version will be classified into obsolete test cases and non-obsolete test cases, and only those non-obsolete test cases can be reused to test the modified version. Non-obsolete test cases need to be classified into modification-revealing test cases, fault-revealing test cases and modification-traversing test cases to evaluate their reusability. In [8], a regression testing framework has been presented. It also introduced many techniques to classify test cases according to the control-flow between the old and new programs. The framework shows that regression testing is carried out with three assumptions which may restrict regression testing or have less operability, and it is a difficult and expensive task.

Input domain can be used to evaluate test cases reusability in regression testing. Domain based testing method was introduced [9]. It relies on software object modelling on three levels: scripting, individual manipulation of object and objective attribute value section. Though it facilitates reusing test cases in regression testing, the actual selection of reusable test cases is a complicated task.

There are many criteria to evaluate the reusability of test cases, such as completeness, adequacy, or coverage of test cases on source codes [1]. In this paper, we focus on the control-flow coverage criteria such as branch coverage (BC), condition/decision coverage (C/DC) and modified condition/decision coverage (MC/DC). Interested reader may refer to [2] for details of these coverage criteria. Even though we illustrate our ideas using these three coverage criteria, the methodology can also be applied to other coverage criteria. When compared with previous regression testing approach, our approach is easier and less expensive to evaluate the reusability of test cases among programs with similar specifications.

3. Partitioning of Domain Testing

The input domain of program is defined by its specification, and it can be partitioned into sub-domains. Variables, conditions and decisions of a program may affect the partitioning of its input domain.

Definition 3.1 Let I be the input domain of a program, $S = \{s_1, \ldots, s_n\}$ be the set of input conditions of a program, and P be a partition of input domain into m different subdomains $P_1, P_2, \ldots, P_m$. P is a *specification-based partition (SP)* if it satisfies the following conditions

1. Elements in P are mutually exclusive (that is, for any two different P_i and P_j, $P_i \cap P_j = \emptyset$);
2. $I = \cup_{i=1}^{m} P_i$;
3. For all $P_i \in P$, there exists $s \in S$ such that $s(x) = s(y)$ for all $x, y \in P_i$;
4. For all $s \in S$, there exists $P_i \in P$ such that $s(x) = s(y)$ for all $x, y \in P_i$; and
5. For all $x \in P_i$ and all $y \in P_j (i \neq j)$, there exists $s \in S$ such that $s(x) \neq s(y)$

The input conditions are derived from the specification of the program. For example, if a program calculates the square root of the product of two positive integers x and y, one input condition may be "$x \geq 0$ and $y < 0$." Condition 3 in Definition 3.1 ensures that, for every subdomain in P, there is an input condition in which all inputs in that subdomain agree. Condition 4 ensures that each input condition can be satisfied by a certain subdomain. Condition 5 ensures that there is always an input condition to distinguish one subdomain from another.

When test cases are generated from input domain of a program to satisfy a control-flow based testing criterion such as the C/DC criterion, we need to consider the decisions and conditions in the program. Due to page limitation, we formally define the specification-cum-C/DC (S-C/DC) partition. The definitions of specification-cum-branch (S-B) and specification-cum-MC/DC (S-MC/DC) partitions are similar.

Definition 3.2 Let I be the input domain of a program and $P = \{P_1, P_2, \ldots, P_m\}$ be a specification-based partition of the input domain I. Suppose further that $C = \{c_1, \ldots, c_q\}$ and $D = \{d_1, \ldots, d_r\}$ are the sets of all conditions and decisions in the program, respectively. A partition $P_C = \{P_{C,1}, \ldots, P_{C,k}\}$ of the input domain I is a *specification-cum-C/DC-based (S-C/DC) partition* if it satisfies the following conditions

1. Elements in P_C are mutually exclusive;
2. $I = \cup_{i=1}^{k} P_{C,i}$;[1]
3. For all $P_{C,i} \in P_C$, there exists $d \in D$ such that $d(x) = d(y)$ for every $x, y \in P_{C,i}$;
4. For every $d \in D$, there is $P_{C,i} \in P_C$ such that $d(x) = d(y)$ for every $x, y \in P_{C,i}$;
5. For all $x \in P_{C,i}$ and all $y \in P_{C,j}$ $(j \neq i)$, there is a $d \in D$ such that $d(x) \neq d(y)$;
6. For every $P_{C,i}$, if there is a $x \in P_{C,i} \cap P_j$ for some j, $P_j \subseteq P_{C,i}$.
7. (Decision coverage) For every decision $d \in D$, there exist $P_{C,i}$ and $P_{C,j}$ $(i \neq j)$, if possible, such that $d(x) \neq d(y)$ for every $x \in P_{C,i}$ and $y \in P_{C,j}$ (that is, all possible outcomes of every decision are covered)

[1] Please note that Since P is a SP, $\cup_{i=1}^{m} P_i = I = \cup_{i=1}^{k} P_{C,i}$.

8. (Condition coverage) For every decision $d \in D$ and every condition c in d, there exist $P_{C,i}$ and $P_{C,j}$ ($i \neq j$), if possible, such that $c(x) \neq c(y)$ for every $x \in P_{C,i}$ and $y \in P_{C,j}$ (that is, all possible outcomes of every condition in a decision are covered)

In other words, a specification-cum-C/DC-based (S-C/DC) partition is a partition of the input domain based on the specification-based partition that it can further satisfy the C/DC criterion. Conditions 1–5 in Definition 3.2 are similar to those in Definition 3.1. Condition 6 in Definition 3.2 ensures that the S-C/DC partition can be formed by regrouping the subdomains in SP (P) without subdividing them into smaller parts. Conditions 7 and 8 in Definition 3.2 ensure that any test set, generated by selecting one element from every $P_{C,i}$, satisfies the C/DC criterion.

4. Case Study

We now illustrate our methodology and results via a case study. An optical application called `FocalLength` is adopted as the subject program for the case study. It calculates the focal length of a single lens based on 4 input variables, namely, type of the lens (`type`), the radii of curvatures of the two surfaces (`r1` and `r2`) and direction of the two surfaces (`dir`). The original source code of `FocalLength` is referred to as `Version 0` (or, simply `V0`). We modified `V0` to create three similar source codes having different control-flow structures without changing their specifications. These three newly created programs are referred to as `V1`, `V2` and `V3`.

The original program (`V0`) has 10 decisions, 12 conditions and 11 feasible paths. Program `V1` is created by combining several decisions in `V0` to first check the validity of input values before making further decisions and removing the obsolete decisions afterwards. However, the order in which the variables are being checked remains the same as `V0`. It has 10 decisions, 15 conditions and 11 feasible paths. Program `V2` is also created by combining several decisions in `V0` to first check the input values before making further decisions. Furthermore, the order in which the variables are being checked has been rearranged. After such rearrangement, obsolete decisions are removed. It has 8 decisions, 11 conditions and 9 feasible paths. Program `V3` is similar to `V2` except that the checking of the input values in some decisions are different from those in `V2`. It has 8 decisions, 11 conditions and 9 feasible paths.

4.1. Specification-based Partitioning

We first create the specification-based partition based on the input conditions in the specification of `FocalLength`. Based on the specification, we have

1. `type`: It is represented by an integer, indicating whether it is a convex (represented by a '0') or concave ('1') lens. Hence, it has four different possible input categories, namely 0, 1, (-∞,0) and (1, +∞), which are identified as categories 1, 2, 3 and 4 of `type`, respectively.
2. `r1` and `r2`: They are represented by positive real numbers to indicate the radii of curvatures of the two sides of the lens. Each has two different possible input categories, namely, `r1 (r2)` ≤ 0 and `r1 (r2)` > 0, which are identified as categories 1 and 2 of `r1 (r2)`.
3. `dir`: It is represented by an integer, indicating the direction where the surfaces of the lens are facing. They may be facing the same ('0') or opposing ('1') directions. Hence, it has four different possible input categories, namely 0, 1, (-∞,0) and (1, +∞). which are identified as categories 1, 2, 3 and 4 of `dir`, respectively.
4. R12: There are two additional possible input categories related to `r1` and `r2` because the formula used to calculate the focal length depends on whether `r1` $\geq$ `r2` or `r1` $<$ `r2` when `r1`, `r2` > 0. They are identified as categories 1 and 2 of R12 accordingly.

We then create the specification-based partition P of the input domain of `FocalLength` as a collection of $P_{i,j,k,l,m}$ where i (= 1, 2, 3, 4), j (= 1, 2), k (= 1, 2), l (= 1, 2, 3, 4) and m (= 1, 2) are the input categories of `type`, `r1`, `r2`, `dir`, and R12, respectively.

4.2. S-C/DC Partitioning

Since C/DC criteria relates to the program source, we illustrate how to create the S-C/DC partition of `FocalLength` using `V0` as an example.

The original version `V0` has 10 decisions and 12 conditions. Strictly speaking, there are altogther 24 different situations for satisfying C/DC. These decisions and conditions are interrelated. For example, if Decision 2 (say, `r1` $\leq$ `0 || r2` $\leq$ `0`) falls within the `TRUE` branch of Decision 1 (say, `type == CONVEX`), we can reduce all 6 possible situations of both Decisions 1 and 2 to 5. Hence, these 24 situations can be further reduced to 15.

We then denote such a partition as $P_C^{V0} = \{P_{C,1}^{V0}, \ldots, P_{C,15}^{V0}\}$ where $P_{C,i}^{V0}$ is the subdomain corresponding to the inputs that satisfy the i-th (out of 15) possible situation for satisfying C/DC of `V0`. Table 1 shows the relation between S-C/DC and specification-based partitions. For example, for `V0`, the subdomain $P_{C,7}^{V0}$ consists of specification-based subdomains P_{1223X} and P_{1224X} where X can be either '1' or '2'. Another example is that `V1` has 18 S-C/DC subdomains and its S-C/DC subdomain $P_{C,7}^{V1}$ consists of specification-based subdomains P_{1224X} where X can be either '1' or '2'.

Table 1: Relation between S-C/DC and specification-based partitions

	V0	V1	V2	V3
$P^{V}_{C,1}$	P_{3111X},[a] P_{3112X}, P_{3113X}, P_{3114X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{3221X}, P_{3222X}, P_{3223X}, P_{3224X}, P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}, P_{4221X}, P_{4222X}, P_{4223X}, P_{4224X}	P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}, P_{4221X}, P_{4222X}, P_{4223X}, P_{4224X}	P_{1111X}, P_{1112X}, P_{1113X}, P_{1114X}, P_{2111X}, P_{2112X}, P_{2113X}, P_{2114X}, P_{3111X}, P_{3112X}, P_{3113X}, P_{3114X}, P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}	P_{1111X}, P_{1112X}, P_{1113X}, P_{1114X}, P_{2111X}, P_{2112X}, P_{2113X}, P_{2114X}, P_{3111X}, P_{3112X}, P_{3113X}, P_{3114X}, P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}
$P^{V}_{C,2}$	P_{1121X}, P_{1122X}, P_{1123X}, P_{1124X}	P_{3111X}, P_{3112X}, P_{3113X}, P_{3114X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{3221X}, P_{3222X}, P_{3223X}, P_{3224X}	P_{1121X}, P_{1122X}, P_{1123X}, P_{1124X}, P_{2121X}, P_{2122X}, P_{2123X}, P_{2124X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}	P_{1121X}, P_{1122X}, P_{1123X}, P_{1124X}, P_{2121X}, P_{2122X}, P_{2123X}, P_{2124X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}
$P^{V}_{C,3}$	P_{1211X}, P_{1212X}, P_{1213X}, P_{1214X}	P_{1111X}, P_{1112X}, P_{1113X}, P_{1114X}	P_{1211X}, P_{1212X}, P_{1213X}, P_{1214X}, P_{2211X}, P_{2212X}, P_{2213X}, P_{2214X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}	P_{1211X}, P_{1212X}, P_{1213X}, P_{1214X}, P_{2211X}, P_{2212X}, P_{2213X}, P_{2214X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}
$P^{V}_{C,4}$	P_{12211}	P_{1121X}, P_{1122X}, P_{1123X}, P_{1124X}	P_{1223X}, P_{2223X}, P_{3223X}, P_{4223X}	P_{4221X}, P_{4222X}, P_{4223X}, P_{4224X}
$P^{V}_{C,5}$	P_{12212}	P_{1211X}, P_{1212X}, P_{1213X}, P_{1214X}	P_{1224X}, P_{2224X}, P_{3224X}, P_{4224X}	P_{3221X}, P_{3222X}, P_{3223X}, P_{3224X}
$P^{V}_{C,6}$	P_{1222X}	P_{1223X}	P_{4221X}, P_{4222X}	P_{1223X}, P_{2223X}
$P^{V}_{C,7}$	P_{1223X}, P_{1224X}	P_{1224X}	P_{3221X}, P_{3222X}	P_{1224X}, P_{2224X}
$P^{V}_{C,8}$	P_{2111X}, P_{2112X}, P_{2113X}, P_{2114X}	P_{12211}	P_{1222X}	P_{1222X}
$P^{V}_{C,9}$	P_{2121X}, P_{2122X}, P_{2123X}, P_{2124X}	P_{12212}	P_{2222X}	P_{2222X}
$P^{V}_{C,10}$	P_{2211X}, P_{2212X}, P_{2213X}, P_{2214X}	P_{1222X}	P_{12211}	P_{12211}
$P^{V}_{C,11}$	P_{22211}	P_{2111X}, P_{2112X}, P_{2113X}, P_{2114X}	P_{12212}	P_{12212}
$P^{V}_{C,12}$	P_{22212}	P_{2121X}, P_{2122X}, P_{2123X}, P_{2124X}	P_{22211}	P_{22211}
$P^{V}_{C,13}$	P_{2222X}	P_{2211X}, P_{2212X}, P_{2213X}, P_{2214X}	P_{22212}	P_{22212}
$P^{V}_{C,14}$	P_{2223X}, P_{2224X}	P_{2223X}	n/a	n/a
$P^{V}_{C,15}$	P_{1111X}, P_{1112X}, P_{1113X}, P_{1114X}	P_{2224X}	n/a	n/a
$P^{V}_{C,16}$	n/a	P_{22211}	n/a	n/a
$P^{V}_{C,17}$	n/a	P_{22212}	n/a	n/a
$P^{V}_{C,18}$	n/a	P_{2222X}	n/a	n/a

[a]Note: X can be either '1' or '2'.

Table 2: Reusability of test cases from V0 to V1 based on their S-C/DC partitions (%)

	$P^{V1}_{C,1}$	$P^{V1}_{C,2}$	$P^{V1}_{C,3}$	$P^{V1}_{C,4}$	$P^{V1}_{C,5}$	$P^{V1}_{C,6}$	$P^{V1}_{C,7}$	$P^{V1}_{C,8}$	$P^{V1}_{C,9}$	$P^{V1}_{C,10}$	$P^{V1}_{C,11}$	$P^{V1}_{C,12}$	$P^{V1}_{C,13}$	$P^{V1}_{C,14}$	$P^{V1}_{C,15}$	$P^{V1}_{C,16}$	$P^{V1}_{C,17}$	$P^{V1}_{C,18}$
$P^{V0}_{C,1}$	50	50																
$P^{V0}_{C,2}$				100														
$P^{V0}_{C,3}$					100													
$P^{V0}_{C,4}$								100										
$P^{V0}_{C,5}$									100									
$P^{V0}_{C,6}$										100								
$P^{V0}_{C,7}$						50	50											
$P^{V0}_{C,8}$											100							
$P^{V0}_{C,9}$												100						
$P^{V0}_{C,10}$													100					
$P^{V0}_{C,11}$																100		
$P^{V0}_{C,12}$																	100	
$P^{V0}_{C,13}$																		100
$P^{V0}_{C,14}$														50	50			
$P^{V0}_{C,15}$			100															
Reusability	50	50	100	100	100	50	50	100	100	100	100	100	100	50	50	100	100	100
Total	50	50	100	100	100	50	50	100	100	100	100	100	100	50	50	100	100	100

Table 3: Reusability of test case sets – S-B partition (%)

	V0		V1		V2		V3	
	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment
V0	n/a		100	100.0	16.7 – 100	15.5 – 100.0	25 – 100	25.0 – 100.0
V1	100	100.0	n/a		16.7 – 100	16.4 – 100.0	25 – 100	24.8 – 100.0
V2	33.3 – 100	31.1 – 100.0	33.3 – 100	31.1 – 100.0	n/a		66.7 – 100	65.6 – 100.0
V3	33.3 – 100	32.2 – 100.0	33.3 – 100	30.6 – 100.0	66.7 – 100	63.9 – 100.0	n/a	

Table 4: Reusability of test case sets – S-C/DC partition (%)

	V0		V1		V2		V3	
	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment
V0	n/a		50 – 100	46.0 – 100.0	4.17 – 100	5.4 – 100.0	12.5 – 100	12.5 – 100.0
V1	100	84.0 – 100.0	n/a		12.5 – 100	11.0 – 100.0	25 – 100	22.3 – 100.0
V2	25 – 100	20.7 – 100.0	25 – 100	20.2 – 100.0	n/a		50 – 100	47.7 – 100.0
V3	25 – 100	20.6 – 100.0	25 – 100	19.2 – 100.0	50 – 100	49.2 – 100.0	n/a	

Table 5: Reusability of test case sets – S-MC/DC partition (%)

	V0		V1		V2		V3	
	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment	Analysis	Experiment
V0	n/a		50 – 100	47.2 – 100.0	4.17 – 100	5.5 – 100.0	12.5 – 100	11.0 – 100.0
V1	100	86.2 – 100.0	n/a		12.5 – 100	11.7 – 100.0	25 – 100	23.7 – 100.0
V2	25 – 100	21.5 – 100.0	25 – 100	20.6 – 100.0	n/a		50 – 100	50.5 – 100.0
V3	25 – 100	21.2 – 100.0	25 – 100	21.5 – 100.0	50 – 100	50.2 – 100.0	n/a	

4.3. Evaluating Test Case Reusability

The S-C/DC partition can be used to evaluate test case reusability. For example, when we evaluate the reusability of test cases collected from V0 to test V1 using the C/DC criteria, we need to consider the S-C/DC partition of both versions. Based on the overlapping of elements in these S-C/DC partitions, we can calculate the percentage of test cases from one subdomain that satisfy the other subdomain. Table 2 gives these percentages which indicate the reusability of test cases collected from elements of S-C/DC partition of V0 for testing V1 with C/DC critera. For example, from Table 1, test cases from $P^{V0}_{C,7}$ belong to two groups, namely P_{1223X} and P_{1224X}, and these test cases can satisfy either $P^{V1}_{C,6}$ (from P_{1223X}) or $P^{V1}_{C,7}$ (from P_{1224X}). Hence, the chances of these test cases satisfying $P^{V1}_{C,6}$ is 50%. Moreover, the percentage ranges from 50% – 100%. It should be noted that such an estimate is coarse. For more accurate results, we need to know the relative sizes of the related partitions but this is a very time consuming process. Tables 3–5 show the analytical results of the percentages for every pair of V0, V1, V2 and V3.

4.4. Experimental Result

In order to verify the analysis performed as discussed in Section 4.3, we perform an experiment to collect the test case reusability information. Our experiment uses four different versions of FocalLength, namely V0, V1, V2 and V3, and three coverage-based partitions, namely S-B, S-C/DC and S-MC/DC partitions. For a particular version (say, V0), we randomly generate 1000 test sets based on a particular coverage-based partition (say, S-B partition). We then run these 1000 test sets on the other three versions, recording the paths that are exercised by the test sets and check whether the test sets satisfy the same coverage criteria.

Tables 3–5 also show the experimental results for BC, C/DC and MC/DC critieria. For example, from Table 4, the percentage of test cases generated from the S-C/DC partition of V0 that can be reused for testing V1 ranges from 46.0% to 100.0%.

4.5. Reused by Refining a Partition

As the reusability of test cases in Tables 3–5 is not always 100.0%, we propose the following approach to further refine the partitions for enhancing the reusability. Let us illustrate our approach using the situation of generating S-C/DC partition from V0 to test V1. From Table 1, test cases from $P^{V0}_{C,7}$ (P_{1223X} and P_{1224X}) satisfy either $P^{V1}_{C,6}$ (P_{1223X}) or $P^{V1}_{C,7}$ (P_{1224X}). Hence, we can further refine $P^{V0}_{C,7}$ into two subdomains, $P^{V0}_{C,7-1}$ and $P^{V0}_{C,7-2}$, that contain P_{1223X} and P_{1224X}, respectively. As a result, if we select one test case from each of these two newly created

Table 6: Refining existing S-C/DC partition of `V0` to test `V1`

Before Refinement		After Refinement	
Existing Partition	Entries of specification-based partition		New Partition
$P^{V0}_{C,1}$	P_{3111X}, P_{3112X}, P_{3113X}, P_{3114X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{3221X}, P_{3222X}, P_{3223X}, P_{3224X},	P_{3111X}, P_{3112X}, P_{3113X}, P_{3114X}, P_{3121X}, P_{3122X}, P_{3123X}, P_{3124X}, P_{3211X}, P_{3212X}, P_{3213X}, P_{3214X}, P_{3221X}, P_{3222X}, P_{3223X}, P_{3224X}	$P^{V0}_{C,1-1}$
	P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}, P_{4221X}, P_{4222X}, P_{4223X}, P_{4224X}	P_{4111X}, P_{4112X}, P_{4113X}, P_{4114X}, P_{4121X}, P_{4122X}, P_{4123X}, P_{4124X}, P_{4211X}, P_{4212X}, P_{4213X}, P_{4214X}, P_{4221X}, P_{4222X}, P_{4223X}, P_{4224X}	$P^{V0}_{C,1-2}$
$P^{V0}_{C,7}$	P_{1223X}, P_{1224X}	P_{1223X}	$P^{V0}_{C,7-1}$
		P_{1224X}	$P^{V0}_{C,7-2}$
$P^{V0}_{C,14}$	P_{2223X}, P_{2224X}	P_{2223X}	$P^{V0}_{C,14-1}$
		P_{2224X}	$P^{V0}_{C,14-2}$

subdomains to test `V1`, both $P^{V1}_{C,6}$ and $P^{V1}_{C,7}$ can be satisfied. The refinement is based on the relations between S-C/DC partitions of `V0` and `V1`. Table 6 shows the details of the refinement. After the refinement, if we generate additional test cases from the newly created subdomains that are not covered by the original test set, the newly created test set satisfies the C/DC criterion of `V1`. Obviously, this will incur additional costs. However, our approach provides an analytical way to achieve maximum reusability of the already generated test cases.

5. Conclusion

Tester needs to evaluate what test cases can be reused and how these test cases are reused at the beginning of testing. The test case reusability is affected by the modified program and the coverage criteria used in testing. Partitioning of domain testing focuses on input domain to generate test cases and evaluate their reusability among programs with similar specifications on one hand, while maximizing the reuse of existing test cases. Both input domains of the original and modified programs are partitioned with a test coverage criterion. Reusability analysis is based on the overlapping elements of the partitions between the original and modified programs. To achieve 100% coverage and maximize test case reusability, testers may need to further refine existing criterion-based partitions to select more test cases. In this paper, we propose an approach to perform such task and illustrate our idea using a case study.

References

[1] A. Abdurazik, P. Ammann, W. Ding, J. Offutt. Evaluation of three specification-based test criteria. In *Proceedings of 6th IEEE International Conference on Engineering of Complex Computer Systems*, pages 55–68, 2000

[2] J.J. Chilenski and S.P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, 1994.

[3] M. Hutchins, H. Foster, T. Goradia, T. Ostrand. Experiments on the effectiveness of dataflow and controlflow-based test adequacy criteria. In *Proceedings of the 16th ICSE*, pages 191–200, 1994.

[4] J. Knoop, O. Rüthing, B. Steffen. Lazy code motion. *ACM SIGPLAN'92 Notices*, 27(7):224–234, 1992.

[5] W.C. Kreahing, D.B. Whalley, M.W. Bailley, X. Yuan, G.-R. Uh, R. van Engelen. Branch elimination by condition merging *Software Practice and Experience*, 35(1):51–74, 2005.

[6] F. Mueller and D.B. Whalley. Avoiding unconditional jumps by code replication. In *Proceedings of the ACM SIGPLAN*, pages 322–330, 1992.

[7] F. Mueller and D.B. Whalley. Avoiding conditional branches by code replication. In *Proceedings of the ACM SIGPLAN*, pages 56–66, 1995.

[8] G. Rothermel, M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):183–210, 1997.

[9] A. von Mayrhauser, R. Mraz, J. Walls, Domain based testing: increasing test case reuse. In *Proceedings of the IEEE International Conference on Computer Design*, pages 484–491, 1994.

[10] M. Yang, G.-R. Uh, D.B. Whalley. Efficient and effective branch reordering using profile data. *ACM Transaction on Programming Languages and System*, 24(6):667–696, 2002.