Poster: Enhancing Partition Testing through Output Variation

Huai Liu
RMIT University
huai.liu@rmit.edu.au

Pak-Lok Poon
The Hong Kong Polytechnic University
pak.lok.poon@polyu.edu.hk

Tsong Yueh Chen
Swinburne University of Technology
tychen@swin.edu.au

***Abstract*—A major test case generation approach is to divide the input domain into disjoint partitions, from which test cases can be selected. However, we observe that in some traditional approaches to partition testing, the same partition may be associated with different output scenarios. Such an observation implies that the partitioning of the input domain may not be precise enough for effective software fault detection. To solve this problem, partition testing should be fine-tuned to additionally use the information of output scenarios in test case generation, such that these test cases are more fine-grained not only with respect to the input partitions but also from the perspective of output scenarios.**

***Index Terms*—Partition testing, choice relation framework, output scenario.**

I. Introduction

Partition testing is a popular approach to test case generation. It first divides the set of all possible program inputs (namely input domain) into disjoint partitions, and then selects at least one input from each partition to construct a set of test cases. Many software practitioners consider that a single input will be sufficient to represent a partition, if the partition is homogeneous [5]. Typical partition testing methods include the **CHO**i**C**e re**LAT**ion fram**E**work (CHOC'LATE) [1][2], classification-tree method [4], and combinatorial testing [3].

However, we observe that, in some cases, the same input partition is associated with different output scenarios, indicating that some partitions are not sufficiently homogeneous. This problem jeopardizes the very benefit of using input partitions for generating test cases. To alleviate the above problem, we propose that the variations of output scenarios should also be considered when test cases are generated. In this paper, we make use of a typical partition testing method, namely CHOC'LATE, to illustrate how the use of various output scenarios enhances partition testing.

II. CHOC'LATE: A Patition Testing Method

The purpose of CHOC'LATE [1] [2] is to help testers generate test cases from specifications. It works as follows:

1. Identify categories and choices. Testers first identify input parameters and environment conditions as *categories* whose values or states affect the software execution behavior. Each category is further partitioned into *choices*, which refer to the category's different cases.
2. Determine the relation between each pair of choices, and capture all these relations in a *choice relation table*.
3. Generate complete test frames from the choice relation table. Any combination of choices is referred to as a *test frame*. A test frame *TF* can be further defined to be *complete*, if a test case is generated by selecting a concrete value from each choice in *TF*.
4. For each complete test frame, construct a test case by selecting a concrete value for each choice in the frame.

III. Motivating Example

As mentioned above, many partition testing methods, including CHOC'LATE, focus on how to partition the input domain so that test cases associated with the same partition are similar in terms of the execution behaviors aiming at achieving highly homogeneous partitions. However, we argue that input-domain partitioning may not fully satisfy this objective, as illustrated in the following example.

Example (Resource Allocation). Suppose that there are m projects, each of which generates a revenue of r_i with a manpower requirement of p_i (i = 1, 2, …, m), and n departments, each of which has e_j (j = 1, 2, …, n) employees. A program *Res* attempts to assign projects to departments such that (*a*) each project is either assigned to one department or discarded, (*b*) the total manpower required from each department does not exceed its e_j, and (*c*) the total revenue of all the assigned projects is maximized. The input for *Res* includes three sets of integers: two m-tuples $R = (r_1, r_2, \ldots, r_m)$ and $P = (p_1, p_2, \ldots, p_m)$, and one n-tuple $E = (e_1, e_2, \ldots, e_n)$. $\forall i$, $r_i > 0$, $p_i > 0$, and $\forall j$, $e_j > 0$. The output of *Res* is one m-tuple $S = (s_1, s_2, \ldots, s_m)$. $s_i = j$ (where i = 1, 2, …, m and j = 1, 2, …, n) represents that the ith project should be assigned to the jth department, while $s_i = 0$ means that the ith project is discarded.

The categories and choices for *Res* are shown in Table I. For *Res*, there are six categories in total, each of which is associated with three choices. However, it does not mean that there will be $3^6 = 729$ possible complete test frames, because some combinations of choices are invalid according to the specification. A total of 234 complete test frames can be constructed using algorithms provided by CHOC'LATE.

Let us look at a complete test frame {1b, 2a, 3a, 4a, 6b}. Both of the following test cases can be generated from it:

- TC#1: R = (129, 129), P = (55, 55), E = (182).
- TC#2: R = (61, 61), P = (97, 97), E = (114).

Because (55 + 55 < 182), the output of TC#1 is S = (1, 1), that is, both projects are assigned to the only department. The output of TC#2 is S = (1, 0) or S = (0, 1), that is, only one project is assigned while the other is discarded (because (97 + 97 > 114)). In other words, TC#1 and TC#2 trigger

* This project is supported by an ARC grant.

different output scenarios, even though they come from the same complete test frame (that is, the same input partition).

TABLE I. CATEGORIES AND CHOICES FOR *RES* ($i_1 \neq i_2$ and $j_1 \neq j_2$)

Categories	Associated Choices
1. Number of projects (m)	1a. $m = 1$
	1b. $m = 2$
	1c. $m \geq 3$
2. Number of departments (n)	2a. $n = 1$
	2b. $n = 2$
	2c. $n \geq 3$
3. Revenue of project (r_i)	3a. $\forall$ pair of i_1 and i_2, $r_{i1}=r_{i2}$
	3b. $\forall$ pair of i_1 and i_2, $r_{i1}\neq r_{i2}$
	3c. $\exists$ pair of i_1 and i_2, $r_{i1}=r_{i2}$, and $\exists$ pair of i_1 and i_2, $r_{i1}\neq r_{i2}$
4. Manpower for project (p_i)	4a. $\forall$ pair of i_1 and i_2, $p_{i1}=p_{i2}$
	4b. $\forall$ pair of i_1 and i_2, $p_{i1}\neq p_{i2}$
	4c. $\exists$ pair of i_1 and i_2, $p_{i1}=p_{i2}$, and $\exists$ pair of i_1 and i_2, $p_{i1}\neq p_{i2}$
5. Number of employers in department (e_j)	5a. $\forall$ pair of j_1 and j_2, $e_{j1}=e_{j2}$
	5b. $\forall$ pair of j_1 and j_2, $e_{j1}\neq e_{j2}$
	5c. $\exists$ pair of j_1 and j_2, $e_{j1}=e_{j2}$, and $\exists$ pair of j_1 and j_2, $e_{j1}\neq e_{j2}$
6. Relation between p_i and e_j	6a. $\forall\ i$, $p_i > \max(e_1, e_2, \ldots, e_n)$
	6b. $\forall\ i$, $p_i \leq \min(e_1, e_2, \ldots, e_n)$
	6c. $\exists$ pair of i and j, $p_i > e_j$, and $\exists$ pair of i and j, $p_i \leq e_j$

The above observation clearly shows that, although partitioning the input domain by existing methodologies (for example, CHOC'LATE) ensures the homogeneity in terms of the selected input aspects, some test cases from the same partition may still be heterogeneous with respect to output scenarios. Intuitively speaking, to maximize testing effectiveness, each partition should be as homogeneous as possible, and one good way to do this is to have fine-grained partitions that are not only related to input parameters but also corresponding to output scenarios.

IV. ENHANCING CHOC'LATE BY OUTPUT VARIATION

We suggest that, on top of the "traditional" partitioning of the input domain, a partition testing method should also consider the variation in program outputs, with a view to fine-tuning the test case generation process. Here, we propose an enhanced method, namely **CHO**i**C**e re**LAT**ion fram**E**work with **DI**stinguishing out**P**ut scenarios (abbreviated as CHOC'LATE-DIP). It improves CHOC'LATE from the following perspectives.

1. In addition to categories and choices with respect to the input parameters and environment conditions (which, for clarity, are hereafter referred as *I-categories* and *I-choices*, respectively), identify different scenarios of program outputs and define categories and choices for these scenarios (referred to as *O-categories* and *O-choices*, respectively). The O-categories and O-choices for *Res* are listed in Table II (the I-categories and I-choices are already listed in Table I).
2. Construct an *extended* choice relation table. Besides the relation between each pair of I-choices, the table also captures the relation between each pair of O-choices as well as that between every I-choice and every O-choice.
3. Generate valid combinations of I-choices and O-choices as "complete test frames" from the extended choice relation table. Since these generated "complete test frames" contain both I-choices and O-choices, we call them *IO-based complete test frames* (abbreviated as CTF_{IO}). For clarity, those complete test frames containing I-choices only are called *I-based complete test frames* (abbreviated as CTF_I). For *Res*, a total of 607 CTF_{IO} can be generated. Table III shows the relevant statistics for *Res*, confirming that the situation of having a CTF_I associated with multiple CTF_{IO} is very common.

TABLE II. O-CATEGORIES AND O-CHOICES FOR *RES*

O-categories	O-choices
I. Number of selected projects	Ia. No project is assigned
	Ib. All projects are assigned
	Ic. Only some projects are assigned
II. Number of departments with projects assigned	IIa. No department is assigned any project
	IIb. All departments are assigned project(s)
	IIc. Only some departments are assigned project(s)

TABLE III. RELATIONSHIP BETWEEN CTF_I AND CTF_{IO} FOR *RES*

k	Number of	Percentage of
	CTF_I associated with k CTF_{IO}	
1	101	43.2%
2	13	5.5%
3	120	51.3%

4. Based on each CTF_{IO}, not only can a test case be generated, but its corresponding type of expected output can also be determined simultaneously. In CHOC'LATE, generating a test case from CTF_I and determining its corresponding type of expected output are two "separate" tasks. In CHOC'LATE-DIP, these two tasks are integrated through CTF_{IO}. This represents another merit of CHOC'LATE-DIP over CHOC'LATE.

V. DISCUSSION AND CONCLUSION

The traditional approach to partitioning the input domain may not be sufficiently strong to ensure similar execution behaviors for the same resultant partitions. In this paper, we propose that output scenarios should also be explicitly considered for any partition testing method to improve the homogeneity of the input partitions, which, in turn, is the key factor for high fault-detection effectiveness. Such improvement is not only restricted to the testing method (CHOC'LATE) under this study, but can be generally used for enhancing many other partition testing techniques.

Due to the page limit, we only used one real-life system for the illustration and case study. A larger scale empirical study is our next step to investigate how to improve various partition testing methods on different types of systems.

REFERENCES

[1] T.Y. Chen, P.-L. Poon, S.-F. Tang, and T.H. Tse. DESSERT: a divide-and-conquer methodology for identifying categories, choices, and choice relations for test case generation. *IEEE T Software Eng*, 38(4):794-809, 2012.

[2] T.Y. Chen, P.-L. Poon, and T.H. Tse. A choice relation framework for supporting category-partition test case generation. *IEEE T Software Eng*, 29(7):577-593, 2003.

[3] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 13(5):83-88, 1996.

[4] M. Grochtmann and K. Grimm. Classification tree for partition testing. *Software Test Verif Rel*, 3(2):63-82, 1993.

[5] D. Hamlet. Partition testing does not inspire confidence. *IEEE T Software Eng*, 16(12):1402-1411, 1990.