

# The pending schema of Combinatorial Testing

XINTAO NIU, State Key Laboratory for Novel Software Technology, Nanjing University, China

CHANGHAI NIE, State Key Laboratory for Novel Software Technology, Nanjing University, China

JEFF Y. LEI, Department of Computer Science and Engineering, The University of Texas at Arlington, USA

XIAOYIN WANG, Department of Computer Science, The University of Texas at San Antonio, USA

FEI-CHING KUO, Swinburne University of Technology, Australia

Combinatorial testing (CT) aims to detect the failures which are triggered by the interactions of various factors that can influence the behaviour of the system, such as input parameters, and configuration options. Many studies in CT focus on designing an elaborate test suite (called covering array) to reveal such failures. Although covering array can assist testers to systemically check each possible factor interaction, however, it provides weak support to locate the failure-inducing interactions. Recently some elementary researches are proposed to handle the failure-inducing interaction identification problem, but some issues, such as unable to identify overlapping failure-inducing interactions, and generating too many additional test cases, can negatively influence the applicability of these approaches. In this paper, we propose a novel failure-inducing identification approach which aims to handle those issues. The key of our approach is to search for a proper factor interaction at each iteration to check whether it is failure-inducing or not until all the interactions in a failing test cases are checked. Moreover, we conduct empirical studies on both widely-used real-life highly-configurable software systems and synthetic softwares. Results showed that our approach obtained a higher quality at the failure-inducing interaction identification, while just needed a smaller number of additional test cases.

CCS Concepts: • **Software defect analysis** → **Software testing and debugging**;

Additional Key Words and Phrases: Pending Schema, Software Testing, Combinatorial Testing, Failure-inducing interactions

## ACM Reference Format:

Xintao Niu, Changhai Nie, Jeff Y. Lei, Xiaoyin Wang, and Fei-Ching Kuo. 2010. The pending schema of Combinatorial Testing. *ACM Trans. Web* 9, 4, Article 39 (March 2010), 13 pages. <https://doi.org/0000001.0000001>

## 1 INTRODUCTION

The behavior of modern software is affected by many factors, such as input parameters, configuration options, and specific events. To test such software system is challenging, as in theory we should test all the possible interaction of these factors to ensure the correctness of the System Under Test (SUT)[15]. When the number of factors is large, the interactions to be checked increase

---

Authors' addresses: Xintao Niu, State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Qixia District, Nanjing, Jiangsu, 210023, China, niuxintao@gmail.com; Changhai Nie, State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Qixia District, Nanjing, Jiangsu, 210023, China, changhainie@nju.edu.cn; Jeff Y. Lei, Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, Texas, USA, ylei@cse.uta.edu; Xiaoyin Wang, Department of Computer Science, The University of Texas at San Antonio, San Antonio, Texas, USA, Xiaoyin.Wang@utsa.edu; Fei-Ching Kuo, Swinburne University of Technology, Melbourne, Australia, dkuo@swin.edu.au.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2009 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1559-1131/2010/3-ART39 \$15.00

<https://doi.org/0000001.0000001>

Table 1. MS word example

id	<i>Highlight</i>	<i>Status bar</i>	<i>Bookmarks</i>	<i>Smart tags</i>	<b>Outcome</b>
1	On	On	On	On	PASS
2	Off	Off	On	On	PASS
3	Off	On	Off	Off	Fail
4	On	Off	Off	On	PASS
5	On	Off	On	Off	PASS

exponentially. Hence to apply exhaustive testing is not feasible, and even if it is possible, it is resource-inefficient to check all the interactions. Combinatorial testing (CT) is a promising solution to handle the combinatorial explosion problem [5, 6]. Instead of testing all the possible interactions in a system, it focuses on checking those interactions with number of involved factors no more than a prior number. Many studies in CT focus on designing a elaborate test suite (called covering array) to reveal such failures. Although covering array is effective and efficient as a test suite, it provides weak support to distinguish the failure-inducing interactions from all the remaining interactions.

Consider the following example [1], Table 1 presents a pair-wise covering array for testing an MS-Word application in which we want to examine various pair-wise interactions of options for ‘Highlight’, ‘Status Bar’, ‘Bookmarks’ and ‘Smart tags’. Assume the third test case failed. We can get five pair-wise suspicious interactions that may be responsible for this failure. They are respectively (Highlight: Off, Status Bar: On), (Highlight: Off, Bookmarks: Off), (Highlight: Off, Smart tags: Off), (Status Bar: On, Bookmarks: Off), (Status Bar: On, Smart tags: Off), and (Bookmarks: Off, Smart tags: Off). Without additional information, it is difficult to figure out the specific interactions in this suspicious set that caused the failure. In fact, considering that the interactions consist of other number of factors could also be failure-inducing interactions, e.g., (Highlight: Off) and (Highlight: Off, Status Bar: On, Smart tags: Off), the problem becomes more complicated. Generally, to definitely determine the failure-inducing interactions in a failing test case of  $n$  factors, we need to check all the  $2^n - 1$  interactions in this test case, which is not possible when  $n$  is a large number.

To address this problem, prior work [11] specifically studied the properties of the failure-inducing interactions in SUT, based on which additional test cases were generated to identify them. Other approaches to identify the failure-inducing interactions in SUT include building a tree model [17], adaptively generating additional test cases according to the outcome of the last test case [20], ranking suspicious interactions based on some rules [4], and using graphic-based deduction [8], among others. These approaches can be partitioned into two categories [2] according to how the additional test cases are generated: *adaptive*—additional test cases are chosen based on the outcomes of the executed tests [4, 7, 11–14, 16, 20] or *nonadaptive*—additional test cases are chosen independently and can be executed in parallel [2, 8, 9, 17, 19].

All these approaches, however, are essentially approximate solutions to failure-inducing interactions identification (Theoretically, a definite solution is of exponential computational complexity). Hence, many issues may affect their effectiveness when they are applied in practice. Generally, *non-adaptive* approaches can usually accurately identify the failure-inducing interactions, even when there are multiple ones in the SUT. Their effectiveness are based on some mathematical objects [2, 8, 9]. The shortcoming of these non-adaptive approaches is that they are very ad-hoc, that is, they have many limitations, such as the number of failure-inducing interactions must be given as well as the number of the maximal factors involved in a failure-inducing interaction. Moreover, these approaches usually consume many test cases [20]. *Adaptive* approaches are much more flexible, and they mainly focus on one failing test case. Commonly they also consume much

fewer test cases than *non-adaptive* approaches. However, some problems, such as unable to handle multiple failure-inducing interactions (especially they have overlapping factors), and cannot handle the newly introduced failure-interactions in the additional generated test cases, negatively affect their performance (in terms of both precision and recall).

In this paper, we propose a novel adaptive failure-inducing interaction identification approach which aims to alleviate these issues. Our approach is based on the notions of *faulty schemas* and *healthy schemas*<sup>1</sup>, among which the former will result in a failure if any test case contains it while the latter will not. Furthermore, two important set of schemas are maintained in our approach, i.e., CMXS (candidate maximal pending schemas) and CMNS (candidate minimal pending schemas). With these two set of schemas, all the schemas in the failing test case can be determined to be either healthy, faulty, or still un-determinable (i.e., pending). Different from consecutively selecting each schema one by one in the failing test case and checking whether it is determinable or not, our approach can directly obtain all the pending schemas by CMXS and CMNS. Based on several important propositions proposed in this paper, the complexity of this procedure is only  $O(\tau^{|FSS^+|} \times \tau^{|HSS^+|})$ , where  $\tau$  is the number of parameter values in the failure-inducing interactions, and  $|FSS^+|$  and  $|HSS^+|$  are two relatively small numbers. This complexity is small when compared to consecutively checking schemas in the test case, of which the complexity is  $O(2^n)$ , where  $n$  is the total number of parameter values in the test case.

After obtaining these un-determined schemas, our approach needs to generate additional test cases to check them to be healthy or faulty. When an un-determined schema is checked, we will update these two structures, i.e., CMXS and CMNS. This process is repeated until all the schemas are determined to be healthy or faulty, and the failure-inducing schemas will be selected from those faulty schemas. By doing so, we will not omit any schema that can be candidate failure-inducing interaction in the failing test case. As a result, our approach can handle the cases such as a failing test case containing overlapping failure-inducing interactions and interactions consisting of any different number of factors.

To evaluate the performance of our approach, we firstly conduct several experiments to compare our approach with all the existing failure-inducing schemas identification approaches. These experiments consider various factors that may influence the performance of these approaches, e.g., the number of factors in the SUT, the number of MFS in the SUT, the degrees of these MFS. As far as we are aware, our work is the first to comprehensively compare all those adaptive MFS identification approaches. Moreover, we conduct the experiment based on 9 industrial software with real-life faults. Our results suggest that our approach obtains a better failure-inducing schema identification results when compared with other works, while requiring a small number of additional test cases.

### Contributions of this paper:

- We proposed several important propositions to support failure-inducing schemas identification, with which the complexity of obtaining un-determined schemas is significantly reduced.
- We proposed a novel adaptive approach which can identify the failure-inducing schemas effectively and efficiently.
- Our approach takes into account several issues that may negatively affect the performance of existing approaches of failure-inducing schemas identification.
- We conducted a series of experiments to comprehensively compare all those adaptive approaches.

The remainder of this paper is organized as follows: Section 3 introduces some preliminary definitions and propositions. Section 4 describes our approaches for identifying failure-inducing

<sup>1</sup>schema is identical to interaction in this paper, and these two terms may be used interchangeably

schemas. Section ?? gives the comparisons in theoretical metrics. Section ?? describes the experiment based on real-life subjects. Section ?? summarizes the related works. Section ?? concludes this paper and discusses the future works.

## 2 MOTIVATING EXAMPLE

Why minimal failure-causing schema is not effective? This example shows that . Because there still existing pending schemas.

## 3 PRELIMINARY

This section presents some definitions and propositions to give a formal model for CT.

Assume that the Software Under Test (SUT) is influenced by a set of parameters  $P$ , which contains  $n$  parameters, and each parameter  $p_i \in P$  can take the values from the finite set  $V_i$  ( $i = 1, 2, \dots, n$ ).

*Definition 3.1.* A test case of the SUT is a tuple of  $n$  values, one for each parameter of the SUT. It is denoted as  $(v_1, v_2, \dots, v_n)$ , where  $v_1 \in V_1, v_2 \in V_2 \dots v_n \in V_n$ .

In practice, these parameters in the test case can represent many factors, such as input variables, run-time options, building options or various combination of them. We need to execute the SUT with these test cases to ensure the correctness of the behaviour of the SUT.

We consider any abnormally executing test case as a *fault*. It can be a thrown exception, compilation error, assertion failure or constraint violation. When faults are triggered by some test cases, it is desired to figure out the cause of these faults.

*Definition 3.2.* For the SUT, the  $\tau$ -set  $\{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), \dots, (p_{x_\tau}, v_{x_\tau})\}$ , where  $0 \leq x_i \leq n$ ,  $p_{x_i} \in P$ , and  $v_{x_i} \in V_{x_i}$ , is called a  $\tau$ -degree *schema* ( $0 < \tau \leq n$ ), when a set of  $\tau$  values assigned to  $\tau$  distinct parameters.

For example, the interactions (Highlight: Off, Status Bar: On, Smart tags: Off) appearing in Section 1 is a 3-degree schema, where three parameters are assigned to corresponding values. In effect a test case itself is a  $n$ -degree *schema*, which can be described as  $\{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ .

Note that the schema is a formal description of the interaction between parameter values we discussed before.

*Definition 3.3.* Let  $c_1$  be a  $l$ -degree schema,  $c_2$  be an  $m$ -degree schema in SUT and  $l < m$ . If  $\forall e \in c_1, e \in c_2$ , then  $c_1$  is the *sub-schema* of  $c_2$ , and  $c_2$  the *super-schema* of  $c_1$ , which can be denoted as  $c_1 < c_2$ .

For example, the 2-degree schema  $\{(\text{Highlight}, \text{Off}), (\text{Status Bar}, \text{On})\}$  is a sub-schema of the 3-degree schema  $\{(\text{Highlight}, \text{Off}), (\text{Status Bar}, \text{On}), (\text{Smart tags}, \text{Off})\}$ .

*Definition 3.4.* If for any test cases that contain a schema, say  $c$ , it will trigger a failure, then we call this schema  $c$  the *faulty schema*. Additionally, if none of sub-schema of  $c$  is a *faulty schema*, we then call the schema  $c$  the *minimal failure-causing schema (MFS)* [11].

Note that MFS is identical to the failure-inducing interaction discussed previously. In this paper, the terms *failure-inducing interactions* and *MFS* are used interchangeably. Figuring the MFS helps to identify the root cause of a failure and thus facilitate the debugging process.

*Definition 3.5.* A schema, say,  $c$ , is called a *healthy schema* when we find at least one passing test case that contains this schema. In addition, if none of super-schema of  $c$  is the *healthy schema*, we then call the schema  $c$  the *maximal healthy schema (MHS)*.

These two type of schemas, i.e., MFS and MHS, are the keys to our approach, as they are essentially representations of the healthy schemas and faulty schemas in a test case. As shown later, other schemas can be determined to be healthy or faulty by these two type of schemas. As a result, we just need to record these two types of schemas (normally a small amount) instead of recording all the schemas in a test case (up to  $2^n$ ) when identifying MFS.

*Definition 3.6.* A schema is called a *pending* schema, if it is not yet determined to be *healthy* schema or *faulty* schema.

The *pending* schema is actually the *un-determined* schema discussed in Section 1. In effect, to identify the MFS in a failing test case is to figure out all the *pending* schemas, and then try to classify each of them into *healthy* or *faulty* schema. After that, the MFS can be selected from those *faulty* schemas by definition.

To facilitate our discussion, we introduce the following assumptions that will be used throughout this paper:

ASSUMPTION 1. *The execution result of a test case is deterministic.*

This assumption is a common assumption of CT[4, 12, 20]. It indicates that the outcome of executing a test case is reproducible and will not be affected by some random events. Some approaches have already proposed measures to handle this problem, e.g., studies in [3, 17] use multiple covering arrays to avoid this problem.

ASSUMPTION 2. *If a test case contains a MFS, it must fail as expected.*

This assumption shows that we can always observe the failure caused by the MFS. In practice, some issues may prevent this observation. For example, the coincidental correctness problem [10] may happen through testing, when the faulty-code is executed but the failure doesn't propagate to the output. Masking effect [18] may also make the failure-observation difficult, as other failure may triggered and stop the program to go on discovering the remaining failures.

We will later discuss the impacts on MFS identification from these two assumptions, as well as how to alleviate them. Based on these definitions and assumptions, we can get several propositions as following. These propositions are the foundation of our approach, and their proofs are omitted due to their simplicity.

PROPOSITION 3.7. *Given schemas  $s_1, s_2$ , and  $s_3$ , if  $s_1 < s_2, s_2 < s_3$ , then  $s_1 < s_3$ .*

PROPOSITION 3.8. *Given a faulty schema  $s_1$ , then  $\forall s_2, s_1 < s_2, s_2$  is a faulty schema.*

PROPOSITION 3.9. *Given a healthy schema  $s_1$ , then  $\forall s_2, s_2 < s_1, s_2$  is a healthy schema.*

PROPOSITION 3.10. *Given two pending schemas  $s_1, s_2$ , and  $s_1 < s_2$ . Then  $\forall s_3, s_1 < s_3 < s_2, s_3$  is a pending schema.*

## 4 THE APPROACH

In this section, we will describe our approach to identify the failure-inducing schemas in the SUT. To give a better description, we will give several propositions which provide theoretical supports for our approach. The proofs of these propositions are given in the Appendix.

### 4.1 Obtaining pending schema

To identify the MFS in a failing test case, we need to figure out all the pending schemas and then classify them into faulty or healthy schemas. For this, we firstly need to be able to find one pending schema and check it. Then, we repeat this procedure until no more pending schema can be obtained.

To be general, we formalize the problem of obtaining one pending schema as the following problem:

*Given a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , a set of faulty schemas  $FSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i < t$  or  $c_i = t$  (That is,  $c_i \leq t$ ), a set of healthy schemas  $HSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i \leq t$ , and  $FSS \cap HSS = \emptyset$ . The goal is to find one pending schema.*

Note that at the beginning of MFS identification, if there is no additional information,  $FSS$  will be initialized to have one element, i.e.,  $t$  itself, and  $HSS$  is an empty set.

We will settle this problem step by step. According to the Propositions 3.8 and 3.9, it is easy to find that, the pending schemas set  $PSS$  should be the following set

$$PSS = \{c \mid c < t \ \&\& \ \forall c_f \in FSS, c_f \not\leq c \ \&\& \ \forall c_h \in HSS, c \not\leq c_h\}. \quad (1)$$

To obtain one pending schema, we just need to select one schema which satisfies  $c \in PSS$ . However, to directly utilize Formula 1 is not practical to obtain one pending schema, because in the worst case it needs to check every schema in a test case  $t$ , of which the complexity is  $O(2^n)$ . Hence, we need to find another formula which is equivalent to Formula 1, but with much lower complexity.

For this purpose, we defined the following two sets, i.e. CMXS and CMNS.

**Definition 4.1.** For a  $k$ -degree faulty schema  $c = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), \dots, (p_{x_k}, v_{x_k})\}$ , a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and  $c \leq t$ . We denote the candidate maximal pending schema set as  $CMXS(c, t) = \{t \setminus (p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in c\}$ .

Note that CMXS is the set of schemas that remove one distinct factor value in  $c$ , such that all these schemas will not be the super-schema of  $c$ . For example assume the failing test case  $\{(p_1, v_1), (p_2, v_2), (p_3, v_3), (p_4, v_4)\}$ , and a faulty schema  $\{(p_1, v_1), (p_3, v_3)\}$ . Then the CMXS set is  $\{\{(p_2, v_2), (p_3, v_3), (p_4, v_4)\}, \{(p_1, v_1), (p_2, v_2), (p_4, v_4)\}\}$ . Obviously, the complexity of obtaining CMXS of one faulty schema is  $O(\tau)$ , where  $\tau$  is the number of parameter values in this faulty schema, i.e., the degree of this schema.

With respect to the CMXS set of a single faulty schema, we can get the following proposition:

**PROPOSITION 4.2.** *Given a faulty schema  $c_1$ , a failing test case  $t$ , where  $c_1 \leq t$ , we have  $\{c \mid c < t \ \&\& \ c_1 \not\leq c\} = \{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1\}$ .*

This proposition means that the pending schema must be the sub-schema or equal to the schemas in CMXS, otherwise, it is a faulty schema.

In the equation of Proposition 4.2, the schemas of the left side, i.e.,  $\{c \mid c \leq t \ \&\& \ c_1 \not\leq c\}$ , are the sub-schemas of test case  $t$ , but not the super-schemas of faulty schema  $c_1$  nor equal to  $c_1$ . Note that the pending schemas can only appear in this set, because any schema that is not belong to this set is faulty schema according to Proposition 3.8. The right side set in this equation, i.e.,  $\{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1\}$ , are schemas which are sub-schemas of or equal to at least one schema in  $CMXS(c_1, t)$ . Proposition 4.2 indicates that these two schema sets are equivalent. As an example, considering a failing test case  $t = \{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ , and a faulty schema  $c_f = \{(p_3, 1)\}$ . Table 2 shows the schema set  $\{c \mid c < t \ \&\& \ c_f \not\leq c\}$ ,  $CMXS(c_f, t)$  and  $\{c \mid \exists c'_1 \in CMXS(c_f, t), c \leq c'_1\}$ .

We can extend this conclusion to a set of faulty schemas. For this, we need the following notation: For two faulty schemas  $c_1, c_2$ , and a failing test case  $t$  ( $c_1 \leq t, c_2 \leq t$ ), let  $CMXS(c_1, t) \wedge CMXS(c_2, t) = \{c \mid c = c'_1 \cap c'_2, \text{ where } c'_1 \in CMXS(c_1, t), \text{ and } c'_2 \in CMXS(c_2, t)\}$ .

For example, let  $t = \{(p_1, v_1), (p_2, v_2), (p_3, v_3)\}$ ,  $c_1 = \{(p_1, v_1), (p_2, v_2)\}$ ,  $c_2 = \{(p_2, v_2), (p_3, v_3)\}$ . Then we have  $CMXS(c_1, t) = \{\{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2), (p_3, v_3)\}\}$ ,  $CMXS(c_2, t) = \{\{(p_1, v_1), (p_2, v_2)\},$



Table 2. An example of Proposition 4.2

Test case $t$	$\{c \mid c < t \text{ \&\& } c_f \not\leq c\}$	$CMXS(c_f, t)$	$\{c \mid \exists c'_1 \in CMXS(c_f, t), c \leq c'_1\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$
<b>Faulty schema <math>c_f</math></b>	$\{(p_1, 1), (p_2, 1)\}$		$\{(p_1, 1), (p_2, 1)\}$
$\{(p_3, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_4, 1)\}$
	$\{(p_2, 1), (p_4, 1)\}$		$\{(p_2, 1), (p_4, 1)\}$
	$\{(p_1, 1)\}$		$\{(p_1, 1)\}$
	$\{(p_2, 1)\}$		$\{(p_2, 1)\}$
	$\{(p_4, 1)\}$		$\{(p_4, 1)\}$

$\{(p_1, v_1), (p_3, v_3)\}$ , and  $CMXS(c_1, t) \wedge CMXS(c_2, t) = \{\{(p_1, v_1)\}, \{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2)\}, \{(p_3, v_3)\}\}$ . It is easy to know the complexity of obtaining CMXS of two faulty schemas is  $O(\tau^2)$ , where  $\tau$  is the number of parameter values in the faulty schema. Based on this, we denote  $CMXS(FSS, t)$  for a set of faulty schemas.

**Definition 4.3.** Given a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and a set of faulty schemas  $FSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i \leq t$ , we denote the candidate maximal pending schema of this set as  $CMXS(FSS, t) = \bigwedge_{c_i \in FSS} CMXS(c_i, t)$ .

Note to compute the CMXS of a set of faulty schema, we just need to sequentially compute the CMXS of two faulty schemas until the last schema in this set is computed. Hence, the complexity of obtaining CMXS of a set of faulty schema is  $O(\tau^{|FSS|})$ , where  $|FSS|$  is the number of faulty schemas in the schema set, and  $\tau$  is the degree of the schema. According to Proposition 4.2, we have:

**PROPOSITION 4.4.** Given a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and a set of faulty schemas  $FSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i \leq t$ , we have  $\{c \mid c \leq t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c\} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1\}$ .

Proposition 4.4 extends Proposition 4.2 from a single faulty schema to a set of faulty schemas.

As an example, considering a failing test case  $t = \{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ , and a set of faulty schemas  $FSS = \{\{(p_3, 1)\}, \{(p_1, 1), (p_2, 1)\}\}$ . Table 3 shows the schema set  $\{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c\}$ ,  $CMXS(FSS, t)$  and  $\{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1\}$ .

Table 3. An example of Proposition 4.4

Test case $t$	$\{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c\}$	$CMXS(FSS, t)$	$\{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$
<b>Faulty schema set FSS</b>	$\{(p_2, 1), (p_4, 1)\}$	$\{(p_2, 1), (p_4, 1)\}$	$\{(p_2, 1), (p_4, 1)\}$
$\{(p_3, 1)\}$	$\{(p_1, 1)\}$		$\{(p_1, 1)\}$
$\{(p_1, 1), (p_2, 1)\}$	$\{(p_2, 1)\}$		$\{(p_2, 1)\}$
	$\{(p_4, 1)\}$		$\{(p_4, 1)\}$

Next, we give the definition of CMNS.

**Definition 4.5.** For a  $k$ -degree healthy schema  $c = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), \dots, (p_{x_k}, v_{x_k})\}$ , a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and  $c < t$ . We denote the candidate minimal pending schema set as  $CMNS(c, t) = \{\{(p_{x_i}, v_{x_i})\} \mid (p_{x_i}, v_{x_i}) \in t \setminus c\}$ .

Note that CMNS is the set of schemas that are assigned to one distinct factor value that is not in  $c$ , such that all these schemas will not be the sub-schema of  $c$ . For example assume the failing test

case  $\{(p_1, v_1), (p_2, v_2), (p_3, v_3), (p_4, v_4)\}$ , and a healthy schema  $\{(p_1, v_1), (p_3, v_3)\}$ . Then the CMNS set is  $\{\{(p_2, v_2)\}, \{(p_4, v_4)\}\}$ . With respect to the CMNS set of a single healthy schema, we can get the following proposition:

**PROPOSITION 4.6.** *Given a healthy schema  $c_1$ , a failing test case  $t$ , where  $c_1 < t$ , we have  $\{c \mid c < t \ \&\& \ c \not\leq c_1\} = \{c \mid c < t \ \&\& \ \exists c'_1 \in CMNS(c_1, t), c'_1 \leq c\}$ .*

This proposition means that the pending schema must be the super-schema or equal to the schemas in CMNS, otherwise, it is a healthy schema.

In the equation of Proposition 4.6, the schemas of the left side, i.e.,  $\{c \mid c \leq t \ \&\& \ c \not\leq c_1\}$ , are the sub-schemas of test case  $t$ , but not the sub-schemas of healthy schema  $c_1$  nor equal to  $c_1$ . It is obvious that the pending schemas can also only appear in this set, because they cannot be the sub-schema of any healthy schema nor equal to them. The right side set in this equation, i.e.,  $\{c \mid c \leq t \ \&\& \ \exists c'_1 \in CMNS(c_1, t), c'_1 \leq c\}$ , are sub-schemas of test case  $t$ , and also are the super-schemas of or equal to at least one schema in  $CMXS(c_1, t)$ . Proposition 4.6 indicates that these two schema sets are equivalent. As an example, considering a failing test case  $t = \{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ , and a healthy schema  $c_h = \{(p_1, 1), (p_2, 1), (p_3, 1)\}$ . Table 4 shows the schema set  $\{c \mid c < t \ \&\& \ c_h \not\leq c\}$ ,  $CMNS(c_h, t)$  and  $\{c \mid c < t \ \&\& \ \exists c'_1 \in CMXS(c_h, t), c'_1 \leq c\}$ .

Table 4. An example of Proposition 4.6

Test case $t$	$\{c \mid c < t \ \&\& \ c \not\leq c_h\}$	$CMNS(c_h, t)$	$\{c \mid c < t \ \&\& \ \exists c'_1 \in CMNS(c_h, t), c'_1 \leq c\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$	$\{(p_4, 1)\}$	$\{(p_4, 1)\}$	$\{((p_4, 1))\}$
<b>Healthy schema <math>c_h</math></b>	$\{(p_1, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_4, 1)\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1)\}$	$\{(p_2, 1), (p_4, 1)\}$		$\{(p_2, 1), (p_4, 1)\}$
	$\{(p_3, 1), (p_4, 1)\}$		$\{(p_3, 1), (p_4, 1)\}$
	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$
	$\{(p_1, 1), (p_3, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_3, 1), (p_4, 1)\}$
	$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$		$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$

Similarly, for two healthy schemas  $c_1, c_2$ , and a failing test case  $t$  ( $c_1 \leq t, c_2 \leq t$ ), let  $CMNS(c_1, t) \vee CMNS(c_2, t) = \{c \mid c = c'_1 \cup c'_2, \text{ where } c'_1 \in CMXS(c_1, t), \text{ and } c'_2 \in CMXS(c_2, t)\}$ .

For example, let  $t = \{(p_1, v_1), (p_2, v_2), (p_3, v_3)\}$ ,  $c_1 = \{(p_1, v_1), (p_2, v_2)\}$ ,  $c_2 = \{(p_2, v_2), (p_3, v_3)\}$ . Then we have  $CMNS(c_1, t) = \{\{(p_3, v_3)\}\}$ ,  $CMNS(c_2, t) = \{\{(p_1, v_1)\}\}$ , and  $CMNS(c_1, t) \vee CMNS(c_2, t) = \{\{(p_1, v_1), (p_3, v_3)\}\}$ . Based on this, we denote  $CMNS(HSS, t)$  for a set of faulty schemas.

**Definition 4.7.** Given a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and a set of healthy schemas  $HSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i \leq t$ , we denote the candidate minimal pending schema of this set as  $CMNS(HSS, t) = \bigvee_{c_i \in HSS} CMNS(c_i, t)$ .

Similar to  $CMXS(FSS, t)$ , the complexity to obtain  $CMNS(HSS, t)$  is  $O(\tau^{|HSS|})$ , where  $|HSS|$  is the number of healthy schemas in the schema set, and  $\tau$  is the degree of the schema. With respect to  $CMNS(HSS, t)$ , we have:

**PROPOSITION 4.8.** *Given a failing test case  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and a set of healthy schemas  $HSS = \{c_1, c_2, \dots, c_i, \dots\}$ , where  $c_i \leq t$ , we have  $\{c \mid c < t \ \&\& \ \forall c_i \in HSS, c \not\leq c_i\} = \{c \mid c < t \ \&\& \ \exists c'_1 \in CMNS(HSS, t), c'_1 \leq c\}$ .*

Similar to Proposition 4.2 and 4.4, Proposition 4.8 extends Proposition 4.6 from a single healthy schema to a set of healthy schemas.



As an example, considering a failing test case  $t = \{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ , and a set of schema schemas  $HSS = \{ \{(p_1, 1), (p_2, 1), (p_3, 1)\}, \{(p_3, 1), (p_4, 1)\} \}$ . Table 5 shows the schema set  $\{c \mid c < t \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\}$ ,  $CMNS(HSS, t)$  and  $\{c \mid c < t \text{ \&\& } \exists c'_1 \in CMNS(HSS, t), c'_1 \leq c\}$ .

Table 5. An example of Proposition 4.8

Test case $t$	$\{c \mid c < t \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\}$	$CMNS(HSS, t)$	$\{c \mid c < t \text{ \&\& } \exists c'_1 \in CMNS(HSS, t), c'_1 \leq c\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_4, 1)\}$
<b>Healthy schema set <math>HSS</math></b>	$\{(p_2, 1), (p_4, 1)\}$	$\{(p_2, 1), (p_4, 1)\}$	$\{(p_2, 1), (p_4, 1)\}$
$\{(p_1, 1), (p_2, 1), (p_3, 1)\}$	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$
$\{(p_3, 1), (p_4, 1)\}$	$\{(p_1, 1), (p_3, 1), (p_4, 1)\}$		$\{(p_1, 1), (p_3, 1), (p_4, 1)\}$
	$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$		$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$

Based on Proposition 4.4 and 4.8, we can easily learn that  $\{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c\} \cap \{c \mid c < t \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1\} \cap \{c \mid c < t \text{ \&\& } \exists c'_1 \in CMNS(HSS, t), c'_1 \leq c\}$ . Considering  $\forall c \in \{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1\}, c < t$ , we can transform the aforementioned formula into the following equation.

$$\{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c \leq c'_1 \text{ \&\& } \exists c'_1 \in CMNS(HSS, t), c'_1 \leq c\} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c'_2 \in CMNS(FSS, t), c'_2 \leq c \leq c'_1\}.$$

Note that the leftmost side of this equation is identical to Formula 1, hence, we can learn that

$$PSS = \{c \mid \exists c'_1 \in CMXS(FSS, t), c'_2 \in CMNS(HSS, t), c'_2 \leq c \leq c'_1\}. \quad (2)$$

According to Formula 2, the complexity of obtaining one pending schema is  $O(\tau^{|FSS|} \times \tau^{|HSS|})$ . This is because to obtain one pending schema, we only need to search the schemas in  $CMXS(FSS, t)$  and  $CMNS(HSS, t)$ , of which the complexity are  $O(\tau^{|FSS|})$  and  $O(\tau^{|HSS|})$ , respectively. Then we need to check each pair of schemas in these two sets, to find whether exists  $c_1 \in CMXS(FSS, t)$ ,  $c_2 \in CMNS(HSS, t)$ , such that  $c_2 \leq c_1$ . If so, then both  $c_2$  and  $c_1$  satisfy Formula 2. Furthermore,  $\forall c_3, c_2 \leq c_3 \leq c_1, c_3$  also satisfy Formula 2. Hence, the complexity of obtaining one pending schema is  $O(\tau^{|FSS|} \times \tau^{|HSS|})$ .

In fact, we can further reduce the complexity of obtaining pending schemas. When given a set of schemas  $S$ , let the minimal schemas as  $S^\perp = \{c \mid c \in S, \nexists c' \in S, c' < c\}$  and the maximal schemas as  $S^\top = \{c \mid c \in S, \nexists c' \in S, c < c'\}$ . Based on this, we can have the following proposition:

**PROPOSITION 4.9.** *Given a failing test case  $t$ , a set of faulty schemas  $FSS$ , and a set of healthy schemas  $HSS$ , we have  $\{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(HSS^\top, t), c'_2 \leq c \leq c'_1\} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c'_1 \in CMNS(HSS, t), c'_2 \leq c \leq c'_1\}$ .*

As an example, consider a failing test case  $t = \{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ , the faulty schema set  $FSS = \{ \{(p_1, 1), (p_2, 1), (p_3, 1)\}, \{(p_1, 1), (p_2, 1)\} \}$ , and the healthy schema set  $HSS = \{ \{(p_2, 1), (p_3, 1), (p_4, 1)\}, \{(p_2, 1), (p_3, 1)\} \}$ . It is easy to learn that the minimal faulty schema set  $FSS^\perp = \{ \{(p_1, 1), (p_2, 1)\} \}$ , and the maximal healthy schema set  $HSS^\top = \{ \{(p_2, 1), (p_3, 1), (p_4, 1)\} \}$ .

Fig 1 lists all the faulty schemas, healthy schemas, and pending schemas of test case  $t$ . At the second part, it lists the  $CMXS(FSS, t)$ ,  $CMNS(HSS, t)$ , and the schema set  $\{c \mid \exists c'_1 \in CMXS(FSS, t), c'_1 \in CMNS(HSS, t), c'_2 \leq c \leq c'_1\}$ . At last it shows  $CMXS(FSS^\perp, t)$ ,  $CMNS(HSS^\top, t)$  and  $\{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(HSS^\top, t), c'_2 \leq c \leq c'_1\}$ .

We can learn that these two schema set mentioned in Proposition 4.9 are identical in Fig 1, but the schema set based on  $CMXS(FSS^\perp, t)$ ,  $CMNS(HSS^\top, t)$  used much fewer schemas (3 in total) to

obtain the result than the schema set based on  $CMXS(FSS, t)$ ,  $CMNS(HSS, t)$  (7 in total). Another observation from this figure is that these two schema set are both identical to the pending schemas. It suggests that using CMXS and CMNS can effectively and efficiently to get the pending schemas, when compared to list all the schemas in a test case and find the pending schema one by one according to Formula 1.

Failing Test Case $t$ : $\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ Faulty Schema Set (FSS): $\{(p_1, 1), (p_2, 1), (p_3, 1)\}, \{(p_1, 1), (p_2, 1)\}$ Healthy Schema Set (HSS): $\{(p_2, 1), (p_3, 1), (p_4, 1)\}, \{(p_2, 1), (p_3, 1)\}$ Minimal Faulty Schema Set ( $FSS^\perp$ ): $\{(p_1, 1), (p_2, 1)\}$ Maximal Healthy Schema Set ( $HSS^\top$ ): $\{(p_2, 1), (p_3, 1), (p_4, 1)\}$		
<b>Faulty Schemas</b>  $\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_2, 1), (p_3, 1)\}$ $\{(p_1, 1), (p_2, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_2, 1)\}$	<b>Healthy Schemas</b>  $\{(p_2, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_2, 1), (p_3, 1)\}$ $\{(p_2, 1), (p_4, 1)\}$ $\{(p_3, 1), (p_4, 1)\}$ $\{(p_2, 1)\}$ $\{(p_3, 1)\}$ $\{(p_4, 1)\}$	<b>Pending Schemas</b>  $\{(p_1, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_3, 1)\}$ $\{(p_1, 1)\}$
<b>CMXS(FSS, t)</b>  $\{(p_2, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_3, 1), (p_4, 1)\}$ $\{(p_2, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_4, 1)\}$	<b>CMNS(HSS, t)</b>  $\{(p_1, 1), (p_4, 1)\}$ $\{(p_1, 1)\}$	$\{c \mid \exists c_1 \in CMXS(FSS, t), c_2 \leq c \leq c_1\}$ $\{(p_1, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_3, 1)\}$ $\{(p_1, 1)\}$
<b>CMXS(<math>FSS^\perp</math>, t)</b>  $\{(p_2, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_3, 1), (p_4, 1)\}$	<b>CMNS(<math>HSS^\top</math>, t)</b>  $\{(p_1, 1)\}$	$\{c \mid \exists c_1 \in CMXS(FSS, t), c_2 \leq c \leq c_1\}$ $\{(p_1, 1), (p_3, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_4, 1)\}$ $\{(p_1, 1), (p_3, 1)\}$ $\{(p_1, 1)\}$

Fig. 1. The example of Proposition 4.9

At last, according to Proposition 4.9, we have the third formula to compute pending schemas as follow:

$$PSS = \{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(HSS^\top, t), c'_2 \leq c \leq c'_1\}. \quad (3)$$

According to Formula 3, the complexity of obtaining one pending schema is  $O(\tau^{|FSS^\perp|} \times \tau^{|HSS^\top|})$ , where  $|FSS^\perp|$  and  $|HSS^\top|$  are two relatively small numbers during MFS identification.

## 4.2 Empirical studies

### 4.3 The existence of pending schemas

### 4.4 The effectiveness of the approach

#### A PROOF

We will give the proofs of several important propositions.

Proposition 4.2.

PROOF. Let  $A = \{c \mid c < t \text{ \&\& } c_1 \not\leq c\}$ , and  $B = \{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1\}$ .

Let  $c_1 = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), \dots, (p_{x_k}, v_{x_k})\}$ ,  $t = \{(p_1, v_1), (p_2, v_2), \dots, (p_n, v_n)\}$ , and  $CMXS(c_1, t) = \{t \setminus (p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in c_1\}$ .

First we will show  $A \subseteq B$ .

With respect to set A,  $\forall c' \in A$ , it has  $c' < t$  and  $c_1 \not\leq c'$ . That is,  $\forall e \in c', e \in t$ , and  $\exists e' \in c_1, e' \notin c'$ . As  $c_1 \leq t$ ,  $e' \in t$ . Hence, we have  $\forall e \in c', e \in t \setminus e'$ , i.e.,  $c' \leq t \setminus e'$ .

Since  $t \setminus e' \in CMXS(c_1, t)$ ,  $c' \in \{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1\} = B$ . Hence,  $A \subseteq B$ .

Second we will show  $B \subseteq A$ .

With respect to set B,  $\forall c' \in B$ , it has  $\exists c'_1 \in CMXS(c_1, t), c' \leq c'_1$ . Since  $c'_1 \in CMXS(c_1, t)$ ,  $\exists e' \in c_1, c'_1 = t \setminus e'$ . Consequently,  $c' \leq t \setminus e'$ . Hence,  $c_1 \not\leq c'$ . Also,  $c' \leq t \setminus e' < t$ . Consequently,  $c \in \{c \mid c < t \text{ \&\& } c_1 \not\leq c\} = A$ , which indicates that  $B \subseteq A$ .

As we have shown  $B \subseteq A$ , and  $A \subseteq B$ , so  $A = B$ .

□

Proposition 4.4.

PROOF. We just need to prove that for two faulty schemas  $c_1, c_2$ , and a failing test case  $t$  ( $c_1 \leq t, c_2 \leq t$ ), we have  $\{c \mid c < t \text{ \&\& } \forall c_i \in \{c_1, c_2\}, c_i \not\leq c\} = \{c \mid \exists c'_1 \in CMXS(c_1, t) \wedge CMXS(c_2, t), c \leq c'_1\}$ .

Let  $A = \{c \mid c < t \text{ \&\& } \forall c_i \in \{c_1, c_2\}, c_i \not\leq c\}$ ,  $A_1 = \{c \mid c < t \text{ \&\& } c_1 \not\leq c\}$ ,  $A_2 = \{c \mid c < t \text{ \&\& } c_2 \not\leq c\}$ . It is easily to get  $A = A_1 \cap A_2$ .

Let  $B = \{c \mid \exists c'_1 \in CMXS(c_1, t) \wedge CMXS(c_2, t), c \leq c'_1\}$ . Here,  $CMXS(c_1, t) \wedge CMXS(c_2, t) = \{c \mid c = c'_1 \cap c'_2, \text{ where } c'_1 \in CMXS(c_1, t), \text{ and } c'_2 \in CMXS(c_2, t)\}$ .

Let  $B_1 = \{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1\}$ , and  $B_2 = \{c \mid \exists c'_2 \in CMXS(c_2, t), c \leq c'_2\}$ .  $B_1 \cap B_2 = \{c \mid \exists c'_1 \in CMXS(c_1, t), c \leq c'_1 \text{ \&\& } \exists c'_2 \in CMXS(c_2, t), c \leq c'_2\}$ . Note that,  $c \leq c'_1 \text{ \&\& } c \leq c'_2 \equiv c \leq c'_1 \cap c'_2$ . Hence,  $B_1 \cap B_2 = \{c \mid \exists c'_1, c'_2, c'_1 \in CMXS(c_1, t), \text{ and } c'_2 \in CMXS(c_2, t), c \leq c'_1 \cap c'_2\} = B$ .

Based on Proposition 4.2,  $A_1 = B_1, A_2 = B_2$ . Consequently,  $A = A_1 \cap A_2 = B_1 \cap B_2 = B$ .

□

Proposition 4.6.

PROOF. Let  $A = \{c \mid c < t \text{ \&\& } c \not\leq c_1\}$ .  $B = \{c \mid c < t \text{ \&\& } \exists c'_1 \in CMNS(c_1, t), c'_1 \leq c\}$ .  $CMNS(c_1, t) = \{(p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in t \setminus c_1\}$ .

First we will show  $A \subseteq B$ .

With respect to set A,  $\forall c' \in A$ , it has  $c' < t$  and  $c' \not\leq c_1$ . That is,  $\forall e \in c', e \in t$ , and  $\exists e' \in c_1, e' \notin c'$ . Hence,  $\{e'\} \not\leq c', e' \in t \setminus c_1$ , which indicates that  $c' \in \{c \mid c < t \text{ \&\& } \exists c'_1 \in CMNS(c_1, t), c'_1 \leq c\} = B$ , so  $A \subseteq B$ .

Second we will show  $B \subseteq A$ .

With respect to set  $B$ ,  $\forall c' \in B$ , it has  $c' < t$  and  $\exists c'_1 \in CMNS(c_1, t), c'_1 \leq c'$ . As  $c'_1 \in CMNS(c_1, t)$ ,  $\exists e' \in t \setminus c_1, c'_1 = \{e'\}$ . Hence,  $\{e'\} \leq c'$ . Hence,  $c' \not\leq c_1$ . Consequently,  $c' \in \{c \mid c < t \text{ \&\& } c \not\leq c_1\} = A$ , which indicates that  $B \subseteq A$ .  $\square$

**Proposition 4.8.**

**PROOF.** We just need to prove that for two healthy schemas  $c_1, c_2$ , and a failing test case  $t$  ( $c_1 < t, c_2 < t$ ), we have  $\{c \mid c < t \text{ \&\& } \forall c_i \in \{c_1, c_2\}, c \not\leq c_i\} = \{c \mid c < t \text{ \&\& } \exists c'_1 \in CHFS(c_1, t) \vee CHFS(c_2, t), c'_1 \leq c\}$ .

Let  $A = \{c \mid c < t \text{ \&\& } \forall c_i \in \{c_1, c_2\}, c \not\leq c_i\}$ ,  $A_1 = \{c \mid c < t \text{ \&\& } c \not\leq c_1\}$ ,  $A_2 = \{c \mid c < t \text{ \&\& } c \not\leq c_2\}$ . It is easily to get  $A = A_1 \cap A_2$ .

Let  $B = \{c \mid c < t \text{ \&\& } \exists c'_1 \in CHFS(c_1, t) \vee CHFS(c_2, t), c \leq c'_1\}$ . Here,  $CMXS(c_1, t) \vee CMXS(c_2, t) = \{c \mid c = c'_1 \cup c'_2, \text{ where } c'_1 \in CMXS(c_1, t), \text{ and } c'_2 \in CMXS(c_2, t)\}$ .

Let  $B_1 = \{c \mid c < t \text{ \&\& } \exists c'_1 \in CHFS(c_1, t), c'_1 \leq c\}$ , and  $B_2 = \{c \mid c < t \text{ \&\& } \exists c'_2 \in CHFS(c_2, t), c'_2 \leq c\}$ .  $B_1 \cap B_2 = \{c \mid c < t \text{ \&\& } \exists c'_1 \in CHFS(c_1, t), c'_1 \leq c \text{ \&\& } \exists c'_2 \in CHFS(c_2, t), c'_2 \leq c\}$ . Note that,  $c'_1 \leq c \text{ \&\& } c'_2 \leq c \equiv c'_1 \cup c'_2 \leq c$ . Hence,  $B_1 \cap B_2 = \{c \mid c \leq t \text{ \&\& } \exists c'_1, c'_2, c'_1 \in CHFS(c_1, t), \text{ and } c'_2 \in CHFS(c_2, t), c'_1 \cup c'_2 \leq c\} = B$ .

Based on Proposition 4.6,  $A_1 = B_1, A_2 = B_2$ . Consequently,  $A = A_1 \cap A_2 = B_1 \cap B_2 = B$ .  $\square$

**Proposition 4.9.**

**PROOF.** Let  $A = \{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(HSS^\top, t), c'_2 \leq c \leq c'_1\}$ .

Let  $B = \{c \mid \exists c'_1 \in CMXS(FSS, t), c'_2 \in CMNS(HSS, t), c'_2 \leq c \leq c'_1\}$ .

Based on Proposition 4.4 and 4.8:

$A = \{c \mid c < t \text{ \&\& } \forall c_i \in FSS^\perp, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS^\top, c \not\leq c_i\}$ .

$B = \{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\}$ .

First we will prove  $B \subseteq A$ .

As  $FSS^\perp \subseteq FSS, HSS^\top \subseteq HSS, \{c \mid \forall c_i \in FSS, c_i \not\leq c\} \subseteq \{c \mid \forall c_i \in FSS^\perp, c_i \not\leq c\}$  and  $\{c \mid \forall c_i \in HSS, c \not\leq c_i\} \subseteq \{c \mid \forall c_i \in HSS^\top, c \not\leq c_i\}$ . Hence,  $\{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\} \subseteq \{c \mid c < t \text{ \&\& } \forall c_i \in FSS^\perp, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS^\top, c \not\leq c_i\}$ . That is  $B \subseteq A$ .

Next we will prove  $A \subseteq B$ .

Note that  $\forall c < t$ , if  $\forall c_i \in FSS^\perp, c_i \not\leq c$ , it must have  $\forall c'_i \in FSS, c'_i \not\leq c$ . Because if not so, then  $\exists c'_i \in FSS, c'_i \leq c$ . As  $\exists c_i \in FSS^\perp, c'_i \in FSS, c_i \leq c'_i$ , hence,  $c_i \leq c'_i \leq c$ , which is contradiction.

Similarly,  $\forall c < t$ , if  $\forall c_i \in HSS^\top, c \not\leq c_i$ , it must have  $\forall c'_i \in HSS, c \not\leq c'_i$ . Because if not so, then  $\exists c'_i \in HSS, c \leq c'_i$ . As  $\exists c_i \in HSS^\top, c'_i \in HSS, c'_i \leq c_i$ , hence,  $c \leq c'_i \leq c_i$ , which is contradiction.

Combining them, we can get  $\forall c' \in \{c \mid c < t \text{ \&\& } \forall c_i \in FSS^\perp, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS^\top, c \not\leq c_i\}, c' \in \{c \mid c < t \text{ \&\& } \forall c_i \in FSS, c_i \not\leq c \text{ \&\& } \forall c_i \in HSS, c \not\leq c_i\}$ . That is,  $A \subseteq B$ .

As we have shown  $B \subseteq A$ , and  $A \subseteq B$ , so  $A = B$ .  $\square$

## ACKNOWLEDGMENTS

The authors would like to thank Dr. Maura Turolla of Telecom Italia for providing specifications about the application scenario.

The work is supported by the National Natural Science Foundation of China under Grant No.: 61273304\_a and Young Scientists' Support Program (<http://www.nnsf.cn/youngscientists>).

## REFERENCES

- [1] James Bach and Patrick Schroeder. 2004. Pairwise testing: A best practice that isn't. In *Proceedings of 22nd Pacific Northwest Software Quality Conference*. Citeseer, 180–196.
- [2] Charles J Colbourn and Daniel W McClary. 2008. Locating and detecting arrays for interaction faults. *Journal of combinatorial optimization* 15, 1 (2008), 17–48.
- [3] Sandro Fouché, Myra B Cohen, and Adam Porter. 2009. Incremental covering array failure characterization in large configuration spaces. In *Proceedings of the eighteenth international symposium on Software testing and analysis*. ACM, 177–188.
- [4] Laleh Shikh Gholamhossein Ghandehari, Yu Lei, Tao Xie, Richard Kuhn, and Raghu Kacker. 2012. Identifying failure-inducing combinations in a combinatorial test set. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 370–379.
- [5] D Richard Kuhn and Michael J Reilly. 2002. An investigation of the applicability of design of experiments to software testing. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*. IEEE, 91–95.
- [6] D Richard Kuhn, Dolores R Wallace, and Jr AM Gallo. 2004. Software fault interactions and implications for software testing. *Software Engineering, IEEE Transactions on* 30, 6 (2004), 418–421.
- [7] Jie Li, Changhai Nie, and Yu Lei. 2012. Improved Delta Debugging Based on Combinatorial Testing. In *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 102–105.
- [8] Conrado Martínez, Lucia Moura, Daniel Panario, and Brett Stevens. 2008. Algorithms to locate errors using covering arrays. In *LATIN 2008: Theoretical Informatics*. Springer, 504–519.
- [9] Conrado Martínez, Lucia Moura, Daniel Panario, and Brett Stevens. 2009. Locating errors using ELAs, covering arrays, and adaptive testing algorithms. *SIAM Journal on Discrete Mathematics* 23, 4 (2009), 1776–1799.
- [10] Wes Masri and Rawad Abou Assi. 2014. Prevalence of Coincidental Correctness and Mitigation of Its Impact on Fault Localization. *ACM Trans. Softw. Eng. Methodol.* 23, 1, Article 8 (Feb. 2014), 28 pages.
- [11] Changhai Nie and Hareton Leung. 2011. The minimal failure-causing schema of combinatorial testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, 4 (2011), 15.
- [12] Xintao Niu, Changhai Nie, Yu Lei, and Alvin TS Chan. 2013. Identifying Failure-Inducing Combinations Using Tuple Relationship. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*. IEEE, 271–280.
- [13] Kiran Shakya, Tao Xie, Nuo Li, Yu Lei, Raghu Kacker, and Richard Kuhn. 2012. Isolating Failure-Inducing Combinations in Combinatorial Testing using Test Augmentation and Classification. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 620–623.
- [14] Liang Shi, Changhai Nie, and Baowen Xu. 2005. A software debugging method based on pairwise testing. In *Computational Science–ICCS 2005*. Springer, 1088–1091.
- [15] Charles Song, Adam Porter, and Jeffrey S Foster. 2012. iTree: Efficiently discovering high-coverage configurations using interaction trees. In *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 903–913.
- [16] Ziyuan Wang, Baowen Xu, Lin Chen, and Lei Xu. 2010. Adaptive interaction fault location based on combinatorial testing. In *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 495–502.
- [17] Cemal Yilmaz, Myra B Cohen, and Adam A Porter. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *Software Engineering, IEEE Transactions on* 32, 1 (2006), 20–34.
- [18] C. Yilmaz, E. Dumlu, M.B. Cohen, and A. Porter. 2014. Reducing Masking Effects in Combinatorial Interaction Testing: A Feedback Driven Adaptive Approach. *Software Engineering, IEEE Transactions on* 40, 1 (Jan 2014), 43–66.
- [19] Jian Zhang, Feifei Ma, and Zhiqiang Zhang. 2012. Faulty interaction identification via constraint solving and optimization. In *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer, 186–199.
- [20] Zhiqiang Zhang and Jian Zhang. 2011. Characterizing failure-causing parameter interactions by adaptive testing. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 331–341.

Received February 2007; revised March 2009; accepted June 2009