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.

Incompleteness is dangous for its potneail harmful behaviour.

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), 19 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

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.

1559-1131/2010/3-ART39 $15.00
https://doi.org/0000001.0000001

Table 1. MS word example

id	*Highlight*	*Status bar*	*Bookmarks*	*Smart tags*	**Outcome**
1	On	On	On	On	PASS
2	Off	Off	On	On	PASS
3	On	Off	Off	On	PASS
4	On	Off	On	Off	PASS
5	Off	On	Off	Off	Fail

Under Test (SUT)[16]. When the number of factors is large, the interactions to be checked increase exponentially, which makes exhaustive testing not feasible. Combinatorial testing (CT) is a promising solution to handle the combinatorial explosion problem [6, 7]. 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 [2, 9].

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 last 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 [12] specifically studied the properties of the Minimal Failure-causing Schemas (MFS) 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 [18], adaptively generating additional test cases according to the outcome of the last test case [22], ranking suspicious interactions based on some rules [5], and using graphic-based deduction [9], 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 [5, 8, 12–15, 17, 22]or *nonadaptive*–additional test cases are chosen independently and can be executed in parallel [2, 9, 10, 18, 21].

Although many efforts have been devoted to identify the failure-causing schemas from failing test cases, we argue that many of them are still incomplete in terms of the existence of schemas which cannot be determined to be failure-inducing or not. Specifically, when the failure causing schema is obtained, are we surely to answer the following question: is there exist other schemas that . Are we surely identifies all the schemas is determined? or in other word, are there still exist some schemas that we cannot determine it to be faulty or not? To answer this question is important, because this would developer confidence of the SUT under testing. Otherwise, An uncover debugging is potential harmful event and would put in danger. However, to our best knowledge, no such study has been proposed conducted, especially from a theoretical view.

One simple solution is exhaustive list all the schemas. Even is complete, to exhaustive is time consuming. Obviously, to figure it out we need to 2^n, where n is the number is .

In this paper, we tried to answer this question with the notion of pending schemas. We use to evaluate whether is incompete. As a side-effect, we can also evaluate the incompleteness of covering array.

Our approach is based on the notions of *faulty schemas* and *healthy schemas* [1], 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 undeterminable (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 $O(\tau^{|FSS^{\perp}|+|HSS^{\top}|})$, where τ is the number of parameter values in the failure-inducing interactions, and $|FSS^{\perp}|$ and $|HSS^{\top}|$ are two relatively small numbers and is to correlative with number of parameters n in one test case. 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.

We conduct a series studies on some real software systems with number ranged from small to large. There are two studies we conducted, first, we first studies the incompleteness of different fault localization approaches in CT and covering arrays. Second, we analyse the efficiency of three formulas.

Contributions of this paper:

- We show that the traditional covering arrays and the minimal failure-causing schema model are still incomplete in terms of the determination of schemas to be faulty or healthy.
- We introduce the notion of the pending schema to evaluate the incompleteness of these models in combinatorial testing.
- We propose several propositions to formulate the set of pending schemas and give three equivalent formulas to obtain the pending schemas, based on which we reduce the complexity of obtaining pending schemas from $O(2^n)$ to $O(\tau^{|FSS^{\perp}|+|HSS^{\top}|})$, where $|FSS^{\perp}|$ and $|HSS^{\top}|$ are two relatively small numbers and is not correlative to the number of n.
- We conducted a series of experiments to evaluate the incompleteness of traditional covering arrays and MFS identification approaches. Besides, we also evaluate the efficiency of the three formulas on obtaining pending schemas.

The remainder of this paper is organized as follows: Section 2 describes the motivation for this work. Section 3 introduces some preliminary definitions and propositions. Section 4 proposes several important propositions to formally identify the characteristics of the pending schemas. Section 5 evaluates the incompleteness of MFS identification approaches and compares the effectiveness of different approaches for obtaining pending schemas. Section 6 discusses the findings of our research works. Section 7 summarizes the related works. Section 8 concludes this paper and discusses the future works.

2 MOTIVATING EXAMPLES

In this section, we will show that is still incompleteness. Specifically, first shows that with a normal τ-way covering array, it is not completeness in terms of making all the schemas clear for each failing test case.

[1] schema is identical to interaction in this paper, and these two terms may be used interchangeably

Then we give examples shows that even with fault localization, this is also not completeness (with three most important fault localization techniques.)

give examples show based on two wide-used fault localization approaches in CT.

2.1 The incompleteness of covering array

in terms of making the failing test case clear. In this figure, which denoted what. what denoted what.

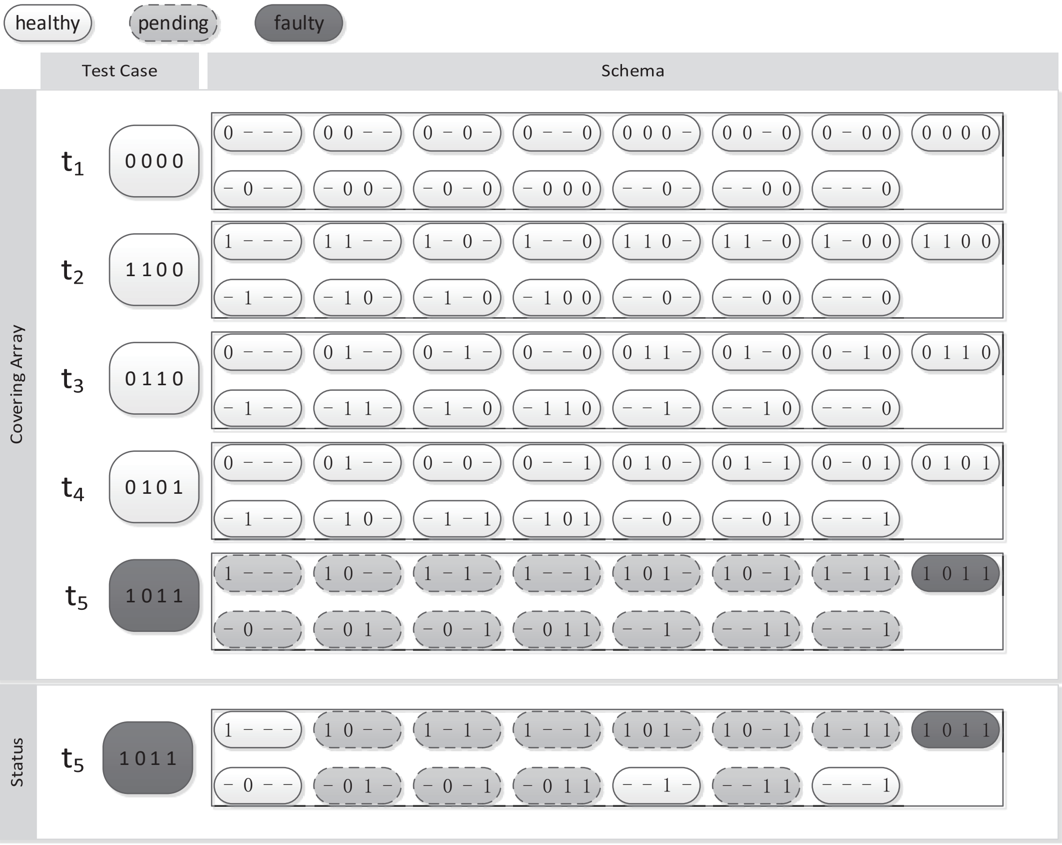

Fig. 1. The incompleteness of Covering array

2.2 The incompleteness of error locating array

Since that , many approaches give a alleviation of how to compute the remaining schemas, which is called the fault localization. Error locating array is one of them. However, this time, we also give .

2.3 The incompleteness of OFOT and SOFOT

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

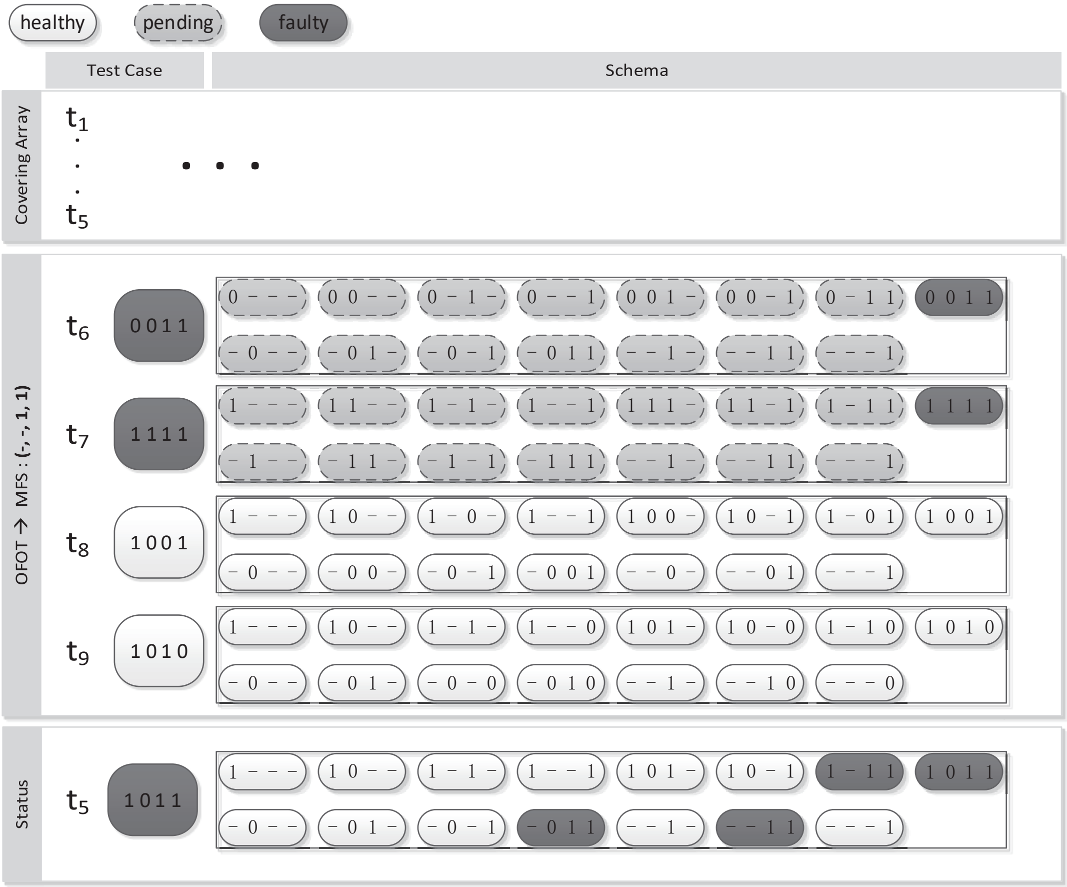

Fig. 2. OFOT with single MFS

2.4 The incompleteness of FIC and FIC_BS

Since we cannot obtain a completeness of . It is important to evaluate how incompleteness of covering array, fault localizaiton.

2.5 A complete process

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,..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 $(v_1, v_2,...,v_n)$, where $v_1 \in V_1$, $v_2 \in V_2$ ... $v_n \in V_n$.

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.

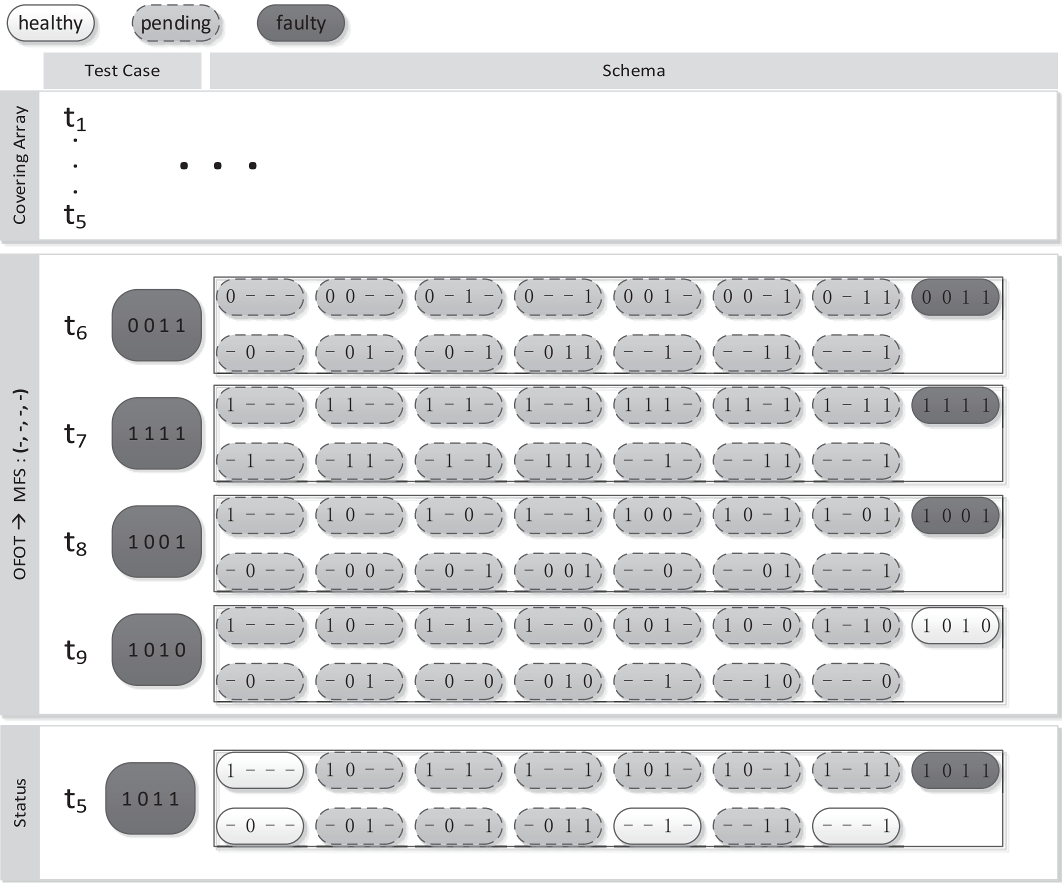

Fig. 3. OFOT with multi MFS

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 τ-set $\{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), ..., (p_{x_\tau}, v_{x_\tau})\}$, where $0 \leq x_i \leq n$, $p_{x_i} \in P$, and $v_{x_i} \in V_{x_i}$, is called a τ-degree *schema* ($0 < \tau \leq n$), when a set of τ values assigned to τ 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 $\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$.

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_2, which can be denoted as $c_1 \prec c_2$.

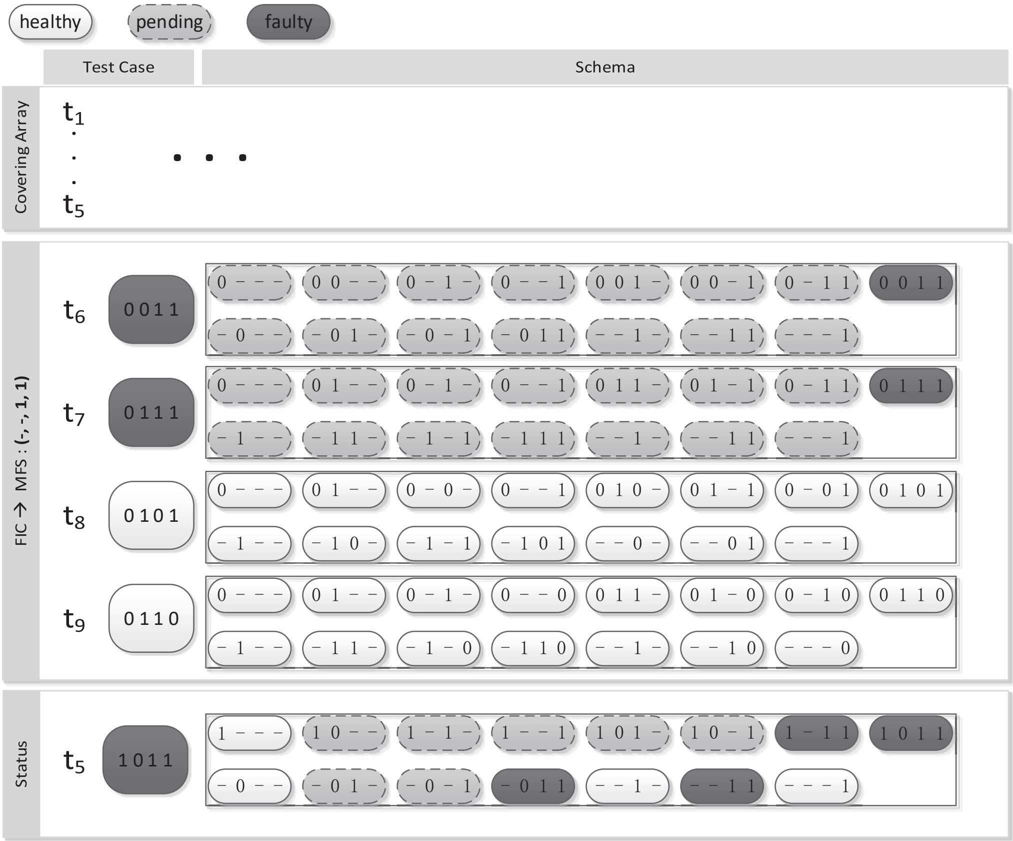

Fig. 4. fic with multiple MFS

For example, the 2-degree schema {(Highlight, Off), (Status Bar, On)} is a sub-schema of the 3-degree schema {(Highlight, Off), (Status Bar, On), (Smart tags, 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)* [12].

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.

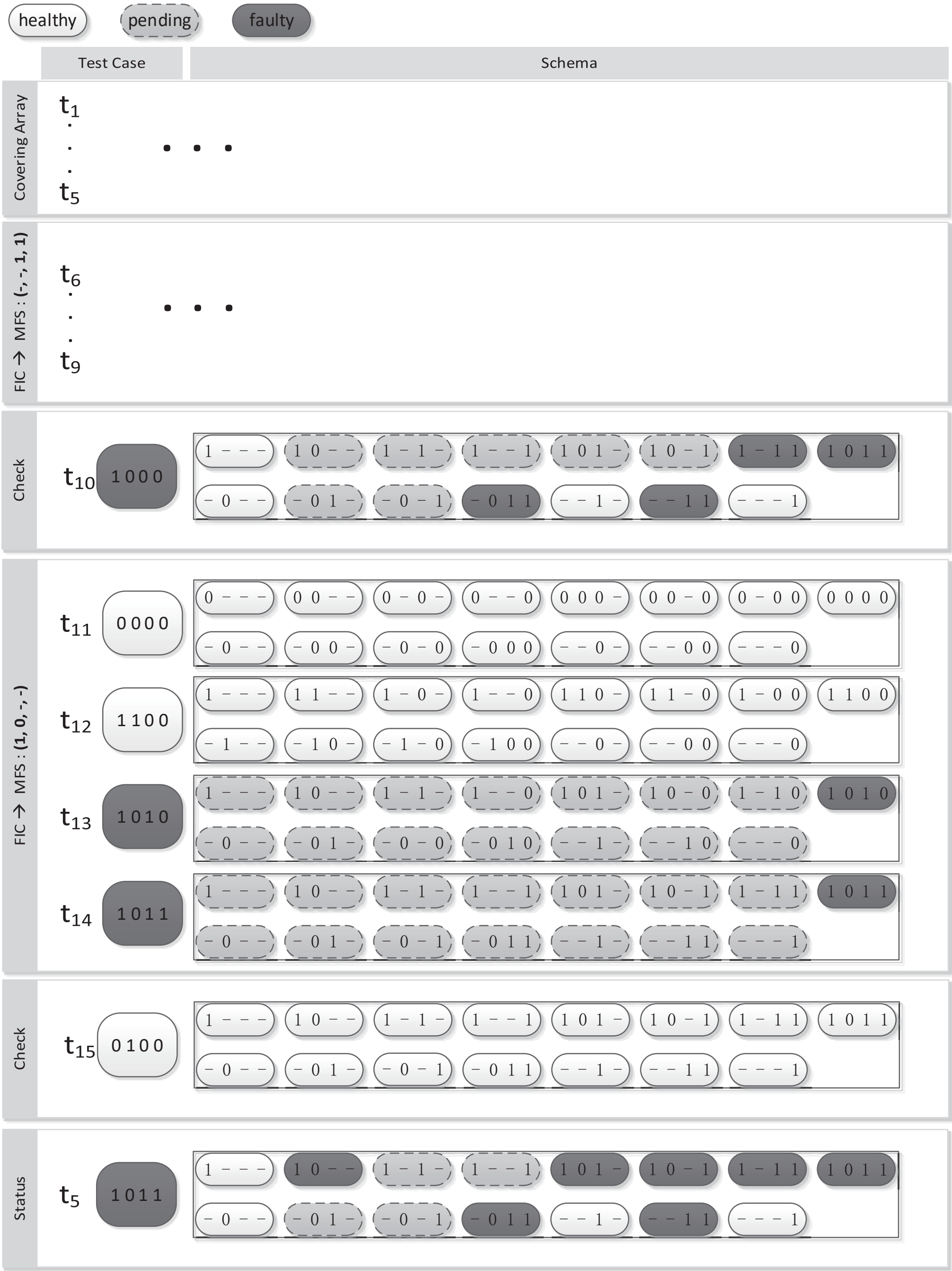

Fig. 5. finovlp with multiple MFS

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

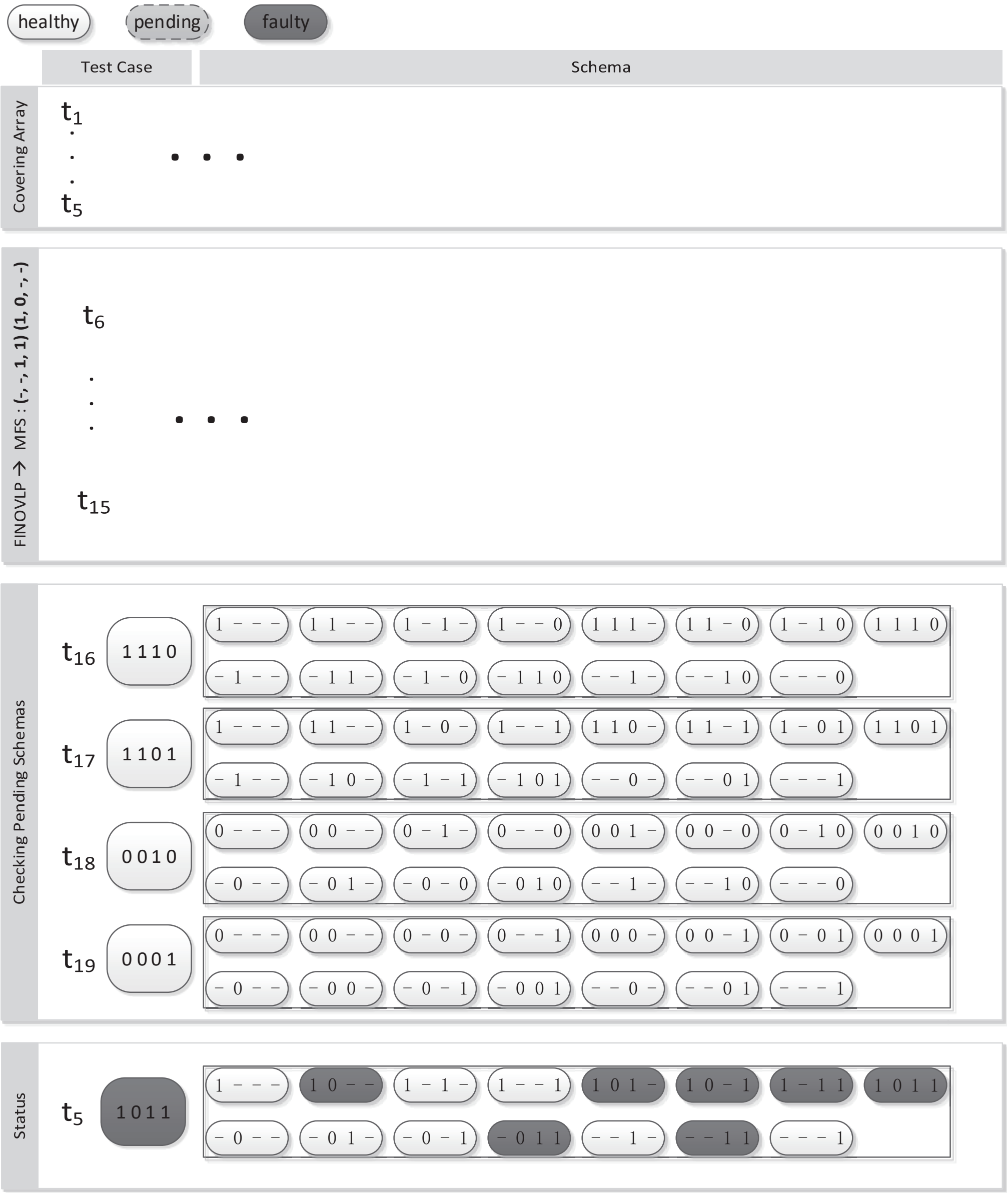

Fig. 6. Non pending schema

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[5, 13, 22]. 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, 18] 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 [11] may happen through testing, when the faulty-code is executed but the failure doesn't propagate to the output. Masking effect [19] 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 $s_1 \prec s_2$, $s_2 \prec s_3$, then $s_1 \prec s_3$.*

Proposition 3.8. *Given a faulty schema s_1, then $\forall s_2, s_1 \prec s_2$, s_2 is a faulty schema.*

Proposition 3.9. *Given a healthy schema s_1, then $\forall s_2, s_2 \prec s_1$, s_2 is a healthy schema.*

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

You should give normal sentences description and examples of them (if possible, give some simple proofs)

4 PENDING SCHEMA

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 What is pending schema

Through the motivating examples, we can learn that the pending schema is not checked to be healthy and faulty. That is,

However, to emulate all the faulty schemas and all the healthy schemas is very large. By the propostios 3 and 4,

More formally, we can learn that it should not be any super-schemas of existing faulty schemas, and any sub-shemas of existing healthy schemas.

4.2 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 =$\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, a set of faulty schemas $FSS = \{c_1, c_2, ...c_i, ...\}$, where $c_i \prec t$ or $c_i = t$ (That is, $c_i \preceq t$), a set of healthy schemas $HSS = \{c_1, c_2, ...c_i, ...\}$, where $c_i \preceq t$, and $FSS \bigcap HSS = \emptyset$. 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

$$PSS = \{c | c \prec t \ \&\& \ \forall c_f \in FSS, c_f \npreceq c \&\& \ \forall c_h \in HSS, c \npreceq c_h\}. \tag{1}$$

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 = $\{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), ..., (p_{x_k}, v_{x_k})\}$, a failing test case t = $\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and $c \preceq t$. We denote the candidate maximal pending schema set as CMXS(c,t)= $\{t \backslash (p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in c\}$.

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(τ), where τ 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* $c_1 \preceq t$*, we have* $\{c \mid c \prec t \ \&\& \ c_1 \npreceq c\} = \{c \mid \exists c_1' \in CMXS(c_1, t), c \preceq c_1'\}$.

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., $\{c \mid c \preceq t \ \&\& \ c_1 \npreceq c\}$, 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., $\{c \mid \exists c_1' \in CMXS(c_1, t), c \preceq c_1'\}$, 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 $\{c \mid c \prec t \ \&\& \ c_f \npreceq c\}$, $CMXS(c_f, t)$ and $\{c \mid \exists c_1' \in CMXS(c_f, t), c \preceq c_1'\}$.

Table 2. An example of Proposition 4.2

Test case t	$\{c \mid c \prec t \ \&\& \ c_f \npreceq c\}$	$CMXS(c_f, t)$	$\{c \mid \exists c_1' \in CMXS(c_f, t), c \preceq c_1'\}$
{(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)}
Faulty schema c_f	{(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)}

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 ($c_1 \preceq t, c_2 \preceq t$), let $CMXS(c_1, t) \bigwedge CMXS(c_2, t) = \{c \mid c = c_1' \cap c_2',\ where\ c_1' \in CMXS(c_1, t),\ and\ c_2' \in CMXS(c_2, t)\}$.

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 $CMXS\ (c_1, t) = \{\ \{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2), (p_3, v_3)\}\ \}$, $CMXS(c_2,\ t) = \{\ \{(p_1, v_1), (p_2, v_2)\}, \{(p_1, v_1), (p_3, v_3)\}\ \}$, and $CMXS(c_1,\ t) \bigwedge CMXS(c_2, t)\ =\ \{\{(p_1, v_1)\}, \{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2)\}, \{(p_3, v_3)\}\}$. It is easy to know the complexity of obtaining CMXS of two faulty schemas is $O(\tau^2)$, where τ 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 = $\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of faulty schemas FSS = $\{c_1, c_2, \ldots c_i, \ldots\}$, where $c_i \preceq t$, 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 τ is the degree of the schema. According to Proposition 4.2, we have:

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

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 $\{c \mid c \prec t\ \&\&\ \forall c_i \in FSS, c_i \npreceq c\}$, $CMXS(FSS, t)$ and $\{c \mid \exists c_1' \in CMXS(FSS, t), c \preceq c_1'\}$.

Table 3. An example of Proposition 4.4

Test case t	$\{\mathbf{c} \mid \mathbf{c} \prec \mathbf{t}\ \&\&\ \forall \mathbf{c_i} \in FSS, \mathbf{c_i} \npreceq \mathbf{c}\}$	**CMXS(FSS, t)**	$\{\mathbf{c} \mid \exists \mathbf{c}_1' \in$ **CMXS(FSS, t)** $, \mathbf{c} \preceq \mathbf{c}_1'\}$
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$
Faulty schema set FSS	$\{(p_2, 1),(p_4, 1)\}$	$\{(p_2, 1),(p_4, 1)\}$	$\{(p_2, 1),(p_4, 1)\}$
$\{(p_3, 1)\}$	$\{(p_1, 1)\}$		$\{(p_1, 1)\}$
$\{(p_1, 1),(p_2, 1)\}$	$\{(p_2, 1)\}$		$\{(p_2, 1)\}$
	$\{(p_4, 1)\}$		$\{(p_4, 1)\}$

Next, we give the definition of *CMNS.*

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

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 $c_1 \prec t$, we have $\{c \mid c \prec t\ \&\&\ c \npreceq c_1\} = \{c \mid c \prec t\ \&\&\ \exists c_1' \in CMNS(c_1, t), c_1' \preceq c\}$.*

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., $\{c \mid c \preceq t \ \&\& \ c \npreceq c_1\}$, 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.,$\{c \mid c \preceq t \ \&\& \ \exists c_1' \in CMNS(c_1, t),\ c_1' \preceq c\}$, 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 $\{c \mid c \prec t \ \&\& \ c_h \npreceq c\}$, $CMNS(c_h, t)$ and $c \mid c \prec t \ \&\& \ \exists c_1' \in CMXS(c_h, t),\ c_1' \preceq c\}$.

Table 4. An example of Proposition 4.6

Test case t	**$\{c \mid c \prec t \ \&\& \ c \npreceq c_h\}$**	**$CMNS(c_h, t)$**	**$\{c \mid c \prec t \ \&\& \ \exists c_1' \in CMNS(c_h, t),\ c_1' \preceq c\}$**
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_4, 1)\}$	$\{(p_4, 1)\}$	$\{((p_4, 1)\}$
Heathy schema c_h	$\{(p_1, 1),(p_4, 1)\}$		$\{(p_1, 1),(p_4, 1)\}$
$\{(p_1, 1),(p_2, 1),(p_3, 1)\}$	$\{(p_2, 1),(p_4, 1)\}$		$\{(p_2, 1),(p_4, 1)\}$
	$\{p_3, 1),(p_4, 1)\}$		$\{p_3, 1),(p_4, 1)\}$
	$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$		$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$
	$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$		$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$
	$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$		$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$

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

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 $CMNS(c_1,\ t) \bigvee CMNS(c_2, t) = \{\{(p_1, v_1), (p_3, v_3)\}\}$. Based on this, we denote $CMNS(HSS, t)$ for a set of faulty schemas.

Definition 4.7. Given a failing test case t = $\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of healthy schemas HSS = $\{c_1, c_2, ...c_i, ...\}$, where $c_i \preceq t$, 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 τ is the degree of the schema. With respect to CMNS(HSS,t), we have:

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

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 $\{c \mid c \prec t \ \&\& \ \forall c_i \in HSS, c \npreceq c_i\}$, $CMNS(HSS, t)$ and $\{c \mid c \prec t \ \&\& \ \exists c_1' \in CMNS(HSS, t),\ c_1' \preceq c\}$.

Based on Proposition 4.4 and 4.8, we can easily learn that $\{c \mid c \prec t \ \&\& \ \forall c_i \in FSS, c_i \npreceq c\} \bigcap \{c \mid c \prec t \ \&\& \ \forall c_i \in HSS, c \npreceq c_i\} = \{c \mid \exists c_1' \in CMXS(FSS, t),\ c \preceq c_1'\} \bigcap \{c \mid c \prec t \ \&\& \ \exists c_1' \in CMNS(HSS, t),\ c_1' \preceq c\}$. Considering $\forall c \in \{c \mid \exists c_1' \in CMXS(FSS, t),\ c \preceq c_1'\}, c \prec t$, we can transform the aforementioned formula into the following equation.

$\{c \mid c \prec t \ \&\& \ \forall c_i \in FSS, c_i \npreceq c \&\& \ \forall c_i \in HSS, c \npreceq c_i\} = \{c \mid \exists c_1' \in CMXS(FSS, t),\ c \preceq c_1' \ \&\& \ \exists c_1' \in CMNS(HSS, t),\ c_1' \preceq c\} = \{c \mid \exists c_1' \in CMXS(FSS, t), c_2' \in CMXS(FSS, t), c_2' \preceq c \preceq c_1'\ \}$.

Table 5. An example of Proposition 4.8

Test case t	$\mathbf{\{c \mid c \prec t\ \&\&\ \forall c_i \in HSS, c \npreceq c_i\}}$	**CMNS(HSS, t)**	$\mathbf{\{c \mid c \prec t\ \&\&\ \exists c'_1 \in CMNS(HSS, t), c'_1 \preceq c\}}$
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$
Heathy schema set HSS	$\{(p_2, 1),(p_4, 1)\}$	$\{(p_2, 1),(p_4, 1)\}$	$\{(p_2, 1),(p_4, 1)\}$
$\{(p_1, 1),(p_2, 1),(p_3, 1)\}$	$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$		$\{(p_1, 1),p_2, 1),(p_4, 1)\}$
$\{(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$		$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$
	$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$		$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$

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

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

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 $c_2 \preceq c_1$. If so, then both c_2 and c_1 satisfy Formula 2. Furthermore, $\forall c_3, c_2 \preceq c_3 \preceq c_1$, 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 $S^\perp = \{c | c \in S, \nexists c' \in S, c' \prec c\}$ and the maximal schemas as $S^\top = \{c | c \in S, \nexists c' \in S, c \prec c'\}$. 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* $\{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(\ HSS^\top, t), c'_2 \preceq c \preceq c'_1\ \} = \{c \mid \exists c'_1 \in CMXS(FSS, t), c'_1 \in CMNS(HSS, t), c'_2 \preceq c \preceq c'_1\ \}$.

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 7 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 $\{c \mid \exists c'_1 \in CMXS(FSS, t)$, $c'_1 \in CMNS(HSS, t)$, $c'_2 \preceq c \preceq c'_1\ \}$. At last it shows $CMXS(FSS^\perp, t)$, $CMNS(HSS^\top, t)$ and $\{c \mid \exists c'_1 \in CMXS(FSS^\perp, t), c'_2 \in CMNS(\ HSS^\top, t), c'_2 \preceq c \preceq c'_1\ \}$.

We can learn that these two schema set mentioned in Proposition 4.9 are identical in Fig 7, 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.

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

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

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.

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 ($HSS^{\top}$): $\{(p_2, 1),(p_3, 1),(p_4, 1)\}$

Faulty Schemas	Healthy Schemas	Pending Schemas
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_2, 1),(p_3, 1)\}$ $\{(p_1, 1),(p_2, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_2, 1)\}$	$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_2, 1),(p_3, 1)\}$ $\{(p_2, 1),(p_4, 1)\}$ $\{(p_3, 1),(p_4, 1)\}$ $\{(p_2, 1)\}$ $\{(p_3, 1)\}$ $\{(p_4, 1)\}$	$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_3, 1)\}$ $\{(p_1, 1)\}$
CMXS(FSS, t)	CMNS(HSS, t)	$\{c \mid \exists c_1 \in CMXS(FSS,t),\ c_2 \in CMNS(HSS,t),\ c_2 \preccurlyeq c \preccurlyeq c_1\}$
$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_3, 1),(p_4, 1)\}$ $\{(p_2, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$ $\{(p_1, 1)\}$	$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_3, 1)\}$ $\{(p_1, 1)\}$
CMXS($FSS^{\perp}$, t)	CMNS($HSS^{\top}$, t)	$\{c \mid \exists c_1 \in CMXS(FSS^{\perp},t),\ c_2 \in CMNS(HSS^{\top},t),\ c_2 \preccurlyeq c \preccurlyeq c_1\}$
$\{(p_2, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1)\}$	$\{(p_1, 1),(p_3, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_4, 1)\}$ $\{(p_1, 1),(p_3, 1)\}$ $\{(p_1, 1)\}$

Fig. 7. The example of Proposition 4.9

4.3 Getting back to the motivation example

When applying this analysis, we can get back to the motivation example. We do not need list all the schemas, but we can figure out which one is the pending schema (Covering array, OFOT, OFOT with single MFS, FIC, FIC with multiple)

5 EMPIRICAL STUDIES

5.1 The pending schemas for covering arrays

subjects (30 existing wildly used)
generation approaches (ipog, aetg, augmented simulating?)

5.2 The existence of pending schemas for different MFS identification approaches

subjects (my , may also used from)
approaches (my)

5.3 The characteristics of pending schemas with various types of MFS (multiple, overlapped, single, low-high degrees)

the same as the captions

5.4 The effectiveness of the approach

three way: (three formulas in the propositions)

5.5 Threats to validity

6 DISCUSSION

7 RELATED WORKS

Shi and Nie [15] presented an approach for failure revealing and failure diagnosis in CT , which first tests the SUT with a covering array, then reduces the value schemas contained in the failing test case by eliminating those appearing in the passing test cases. If the failure-causing schema is found in the reduced schema set, failure diagnosis is completed with the identification of the specific input values which caused the failure; otherwise, a further test suite based on SOFOT is developed for each failing test case, and the schema set is then further reduced, until no more faults are found or the fault is located. Based on this work, Wang [17] proposed an AIFL approach which extended the SOFOT process by adaptively mutating factors in the original failing test cases in each iteration to characterize failure-inducing interactions.

Nie et al. [12] introduced the notion of Minimal Failure-causing Schema(MFS) and proposed the OFOT approach which is an extension of SOFOT that can isolate the MFS in the SUT. This approach mutates one value of that parameter at a time, hence generating a group of additional test cases each time to be executed. Compared with SOFOT, this approach strengthens the validation of the factor under analysis and can also detect the newly imported faulty interactions.

Delta debugging [20] is an adaptive divide-and-conquer approach to locate interaction failure. It is very efficient and has been applied to real software environment. Zhang et al. [22] also proposed a similar approach that can efficiently identify the failure-inducing interactions that have no overlapped part. Later, Li [8] improved the delta-debugging based approach by exploiting useful information in the executed covering array.

Colbourn and McClary [2] proposed a non-adaptive method. Their approach extends a covering array to the locating array to detect and locate interaction failures. Martínez [9, 10] proposed two adaptive algorithms. The first one requires safe value as the assumption and the second one removes this assumption when the number of values of each parameter is equal to 2. Their algorithms focus on identifying faulty tuples that have no more than 2 parameters.

Ghandehari et al. [5] defined the suspiciousness of tuple and suspiciousness of the environment of a tuple. Based on this, they ranked the possible tuples and generated the test configurations. They [4] further utilized the test cases generated from the inducing interaction to locate the fault.

Yilmaz [18] proposed a machine learning method to identify inducing interactions from a combinatorial testing set. They constructed a classification tree to analyze the covering arrays and detect potential faulty interactions. Beside this, Fouché [3] and Shakya [14] made some improvements in identifying failure-inducing interactions based on Yilmaz's work.

Our previous work [13] proposed an approach that utilizes the tuple relationship tree to isolate the failure-inducing interactions in a failing test case. One novelty of this approach is that it can identify the overlapped faulty interaction. This work also alleviates the problem of introducing new failure-inducing interactions in additional test cases.

8 CONCLUSION

A PROOF

We will give the proofs of several important propositions.

Proposition 4.2.

Proof. Let A = $\{c \mid c \prec t \; \&\& \; c_1 \npreceq c\}$, and B = $\{c \mid \exists c'_1 \in CMXS(c_1, t), c \preceq c'_1\}$.

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

First we will show A ⊆ B.

With respect to set A, $\forall c' \in A$, it has $c' \prec t$ and $c_1 \npreceq c'$. That is, $\forall e \in c', e \in t$, and $\exists e' \in c_1, e' \notin c'$. As $c_1 \preceq t$, $e' \in t$. Hence, we have $\forall e \in c', e \in t \backslash e'$, i.e., $c' \preceq t \backslash e'$.

Since $t \backslash e' \in CMXS(c_1, t)$, $c' \in \{c \mid \exists c'_1 \in CMXS(c_1, t), c \preceq c'_1\}$ = B. Hence, A ⊆ B.

Second we will show B ⊆ A.

With respect to set B, $\forall c' \in B$, it has $\exists c'_1 \in CMXS(c_1, t), c' \preceq c'_1$. Since $c'_1 \in CMXS(c_1, t)$, $\exists e' \in c_1, c'_1 = t \backslash e'$. Consequently, $c' \preceq t \backslash e'$. Hence, $c_1 \npreceq c'$. Also, $c' \preceq t \backslash e' \prec t$. Consequently, $c \in \{c \mid c \prec t \; \&\& \; c_1 \npreceq c\}$ = A, which indicates that B ⊆ A.

As we have shown B ⊆ A, and A ⊆ 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 ($c_1 \preceq t$, $c_2 \preceq t$), we have $\{c \mid c \prec t \; \&\& \; \forall c_i \in \{c_1, c_2\}, c_i \npreceq c\} = \{c \mid \exists c'_1 \in CMXS(c_1, t) \bigwedge CMXS(c_2, t), c \preceq c'_1\}$.

Let $A = \{c \mid c \prec t \; \&\& \; \forall c_i \in \{c_1, c_2\}, c_i \npreceq c\}$, $A_1 = \{c \mid c \prec t \; \&\& \; c_1 \npreceq c\}$, $A_2 = \{c \mid c \prec t \; \&\& \; c_2 \npreceq c\}$. It is easily to get $A = A_1 \bigcap A_2$.

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

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

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

□

Proposition 4.6.

Proof. Let A = $\{c \mid c \prec t \; \&\& \; c \npreceq c_1\}$. B = $\{c \mid c \prec t \; \&\& \; \exists c'_1 \in CMNS(c_1, t), \; c'_1 \preceq c\}$. $CMNS(c_1, t)$=$\{(p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in t \backslash c_1\}$.

First we will show A ⊆ B.

With respect to set A, $\forall c' \in A$, it has $c' \prec t$ and $c' \npreceq c_1$. That is, $\forall e \in c', e \in t$, and $\exists e' \in c', e' \notin c_1$. Hence, $\{e'\} \preceq c'$, $e' \in t \backslash c_1$, which indicates that $c' \in \{c \mid c \prec t \; \&\& \; \exists c'_1 \in CMNS(c_1, t), c'_1 \preceq c\}$ = B, so A ⊆ B.

Second we will show B ⊆ A.

With respect to set B, $\forall c' \in B$, it has $c' \prec t$ and $\exists c'_1 \in CMNS(c_1, t), c'_1 \preceq c'$. As $c'_1 \in CMNS(c_1, t)$, $\exists e' \in t \backslash c_1, c'_1 = \{e'\}$. Hence, $\{e'\} \preceq c'$. Hence, $c' \npreceq c_1$. Consequently, $c' \in \{c \mid c \prec t \; \&\& \; c \npreceq c_1\}$ = A, which indicates that B ⊆ A.

□

Proposition 4.8.

Proof. We just need to prove that for two healthy schemas c_1, c_2, and a failing test case t ($c_1 \prec t$, $c_2 \prec t$), we have $\{c \mid c \prec t \ \&\& \ \forall c_i \in \{c_1, c_2\}, c \npreceq c_i\} = \{c \mid c \prec t \ \&\& \ \exists c_1' \in CHFS(c_1, t) \bigvee CHFS(c_2, t), c_1' \preceq c\}$.

Let $A = \{c \mid c \prec t \ \&\& \ \forall c_i \in \{c_1, c_2\}, c \npreceq c_i\}$, $A_1 = \{c \mid c \prec t \ \&\& \ c \npreceq c_1\}$, $A_2 = \{c \mid c \prec t \ \&\& \ c \npreceq c_2\}$. It is easily to get $A = A_1 \bigcap A_2$.

Let $B = \{c \mid c \prec t \ \&\& \ \exists c_1' \in CHFS(c_1, t) \bigvee CHFS(c_2, t), c \preceq c_1'\}$. Here, $CMXS(c_1, t) \bigvee CMXS(c_2, t) = \{c \mid c = c_1' \cup c_2',\ where\ c_1' \in CMXS(c_1, t),\ and\ c_2' \in CMXS(c_2, t)\}$.

Let $B_1 = \{c \mid c \prec t \ \&\& \ \exists c_1' \in CHFS(c_1, t), c_1' \preceq c\}$, and $B_2 = \{c \mid c \prec t \ \&\& \ \exists c_2' \in CHFS(c_2, t), c_2' \preceq c\}$. $B_1 \bigcap B_2 = \{c \mid c \prec t \ \&\& \ \exists c_1' \in CHFS(c_1, t), c_1' \preceq c \ \&\& \ \exists c_2' \in CHFS(c_2, t), c_2' \preceq c\}$. Note that, $c_1' \preceq c \ \&\& \ c_2' \preceq c \equiv c_1' \cup c_2' \preceq c$. Hence, $B_1 \bigcap B_2 = \{c \mid c \preceq t \ \&\& \ \exists c_1', c_2',\ c_1' \in CHFS(c_1, t)$, and $c_2' \in CHFS(c_2, t), c_1' \cup c_2' \preceq c\}$ = B.

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

□

Proposition 4.9.

Proof. Let A = $\{c \mid \exists c_1' \in CMXS(FSS^{\perp}, t), c_2' \in CMNS\ (HSS^{\top}, t), c_2' \preceq c \preceq c_1'\}$.

Let B = $\{c \mid \exists c_1' \in CMXS(FSS, t), c_2' \in CMNS(\ HSS, t), c_2' \preceq c \preceq c_1'\ \}$.

Based on Proposition 4.4 and 4.8:

A = $\{c \mid c \prec t \ \&\& \ \forall c_i \in FSS^{\perp}, c_i \npreceq c \ \&\& \ \forall c_i \in HSS^{\top}, c \npreceq c_i\}$.

B = $\{c \mid c \prec t \ \&\& \ \forall c_i \in FSS, c_i \npreceq c \ \&\& \ \forall c_i \in HSS, c \npreceq c_i\}$.

First we will prove B ⊆ A.

As $FSS^{\perp} \subseteq FSS$, $HSS^{\top} \subseteq HSS$, $\{\ c \mid \forall c_i \in FSS, c_i \npreceq c\} \subseteq \{\ c \mid \forall c_i \in FSS^{\perp}, c_i \npreceq c\}$ and $\{\ c \mid \forall c_i \in HSS, c \npreceq c_i\} \subseteq \{\ c \mid \forall c_i \in HSS^{\top}, c \npreceq c_i\}$. Hence, $\{c \mid c \prec t \ \&\& \ \forall c_i \in FSS, c_i \npreceq c \ \&\& \ \forall c_i \in HSS, c \npreceq c_i\} \subseteq \{c \mid c \prec t \ \&\& \ \forall c_i \in FSS^{\perp}, c_i \npreceq c \ \&\& \ \forall c_i \in HSS^{\top}, c \npreceq c_i\}$. That is $B \subseteq A$.

Next we will prove A ⊆ B.

Note that $\forall c \prec t$, if $\forall c_i \in FSS^{\perp}, c_i \npreceq c$, it must have $\forall c_i' \in FSS, c_i' \npreceq c$. Because if not so, then $\exists c_i' \in FSS, c_i' \preceq c$. As $\exists c_i \in FSS^{\perp}, c_i' \in FSS, c_i \preceq c_i'$, hence, $c_i \preceq c_i' \preceq c$, which is contradiction.

Similarly, $\forall c \prec t$, if $\forall c_i \in HSS^{\top}, c \npreceq c_i$, it must have $\forall c_i' \in HSS, c \npreceq c_i'$. Because if not so, then $\exists c_i' \in HSS, c \preceq c_i'$. As $\exists c_i \in HSS^{\top}, c_i' \in HSS, c_i' \preceq c_i$, hence, $c \preceq c_i' \preceq c_i$, which is contradiction.

Combining them, we can get $\forall c' \in \{c \mid c \prec t \ \&\& \ \forall c_i \in FSS^{\perp}, c_i \npreceq c \ \&\& \ \forall c_i \in HSS^{\top}, c \npreceq c_i\}$, $c' \in \{c \mid c \prec t \ \&\& \ \forall c_i \in FSS, c_i \npreceq c \ \&\& \ \forall c_i \in HSS, c \npreceq c_i\}$. That is, A ⊆ B.

As we have shown B ⊆ A, and A ⊆ B, so A = B. □

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 isnâĂŹt. 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 Sh Ghandehari, Yu Lei, David Kung, Raghu Kacker, and Richard Kuhn. 2013. Fault localization based on failure-inducing combinations. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 168–177.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Liang Shi, Changhai Nie, and Baowen Xu. 2005. A software debugging method based on pairwise testing. In *Computational Science–ICCS 2005*. Springer, 1088–1091.

[16] 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.

[17] 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.

[18] 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.

[19] C. Yilmaz, E. Dumlu, M.B. Cohen, and A. Porter. 2014. Reducing Masking Effects in CombinatorialInteraction Testing: A Feedback DrivenAdaptive Approach. *Software Engineering, IEEE Transactions on* 40, 1 (Jan 2014), 43–66.

[20] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *Software Engineering, IEEE Transactions on* 28, 2 (2002), 183–200.

[21] 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.

[22] 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