The pending schema of Combinatorial Testing

XINTAO NIU, State Key Laboratory for Novel Software Technology, Nanjing University, China
HUAYAO WU, State Key Laboratory for Novel Software Technology, Nanjing University, China
CHANGHAI NIE, State Key Laboratory for Novel Software Technology, Nanjing University, China
YU 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, i.e., the Minimal Failure-causing Schemas (MFS). Recently some elementary researches are proposed to handle the MFS identification problem. However, we argue that many of them are still incomplete in terms of the existence of schemas that cannot be determined to be faulty or not yet. These cannot-be-determined schemas, i.e., the pending schemas, would be hidden dangers to the software under testing. Hence, it is important to obtain these pending schemas for these incomplete MFS identification approaches. In this paper, we proposed several propositions to formulate the set of pending schemas and give three equivalent formulas to obtain them, based on which we reduce the complexity of obtaining pending schemas from $O(2^n)$ to $O(\tau^{|FSS^{\perp}|+|HSS^{\top}|})$, where n is the number of factors in the software, while $|FSS^{\perp}|$ and $|HSS^{\top}|$ are two relatively small numbers and independent on the number of n. We conduct a series empirical studies on some real software systems with various number of parameters and values. Our results shows that the incompleteness is very common in the covering arrays and MFS identification approaches. We also observed that the third proposed formula is the most efficient when compared others in most cases.

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

Additional Key Words and Phrases: Pending Schema, Minimal Failure-causing Schema, Combinatorial Testing, Software Testing

ACM Reference Format:
Xintao Niu, Huayao Wu, Changhai Nie, Yu Lei, Xiaoyin Wang, and Fei-Ching Kuo. 2010. The pending schema of Combinatorial Testing. *ACM Trans. Web* 9, 4, Article 39 (March 2010), 38 pages. https://doi.org/0000001.0000001

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; Huayao Wu, State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Qixia District, Nanjing, Jiangsu, 210023, China, hywu@outlook.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; Yu 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

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)[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, i.e., Minimal Failure-causing schemas(MFS), from all the remaining interactions (schemas) [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 schemas 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 schemas in this suspicious set that caused the failure. In fact, considering that the schemas consist of other number of factors could also be MFS, e.g., (Highlight: Off) and (Highlight: Off, Status Bar: On, Smart tags: Off), the problem becomes more complicated. Generally, to definitely determine the MFS 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 MFS in SUT, based on which additional test cases were generated to identify them. Other approaches to identify the MFS 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 schemas 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 that cannot be determined to be faulty or not yet. Particularly, after identifying the MFS from one failing test case, we wonder that does the schemas other than the identified MFS are guaranteed to be irrelevant to the failure in this failing test case? A related question is that, after identifying the MFS, is there exists any schema in this failing test case that is still cannot be determined to be faulty or not? To answer these two questions is important, because these cannot-be-determined schemas would be hidden dangers to the SUT. Moreover, we need the measures to evaluate the adequacy of the covering arrays and MFS identification approaches in CT, which is a important key to form the confidence of the developer of the SUT. However, to our best knowledge, no such study has been proposed, especially from a theoretical view.

One simple solution is to exhaustively list all the schemas in one failing test case, and then check them to be faulty or not one by one. However, as we have mentioned before, the complexity of this procedure is 2^n, where n is the number of factors in this test case. Hence, this solution is far from feasible when n is very large.

For all of these, a metric should be proposed to assist in evaluating the completeness of MFS identification approaches, and it should be more efficient than a simple exhaustive testing. In this paper, we proposed the notion of **pending schemas**, which indicates the schemas that cannot be determined to be faulty or not. By calculating the number of pending schemas in one failing test case, we can easily assess the extent to which the MFS identification approaches are incomplete. In fact, by the use of pending schemas, we can also evaluate the incompleteness of traditional covering arrays.

Furthermore, we theoretically analyzed the relationships among schemas by proposing nine novel propositions. Based on them, we gave three equivalent formulas, but with different complexities, to obtain the pending schemas. Among these formulas, Formula 3 helps to reduce the complexity of obtaining pending schemas from $O(2^n)$ to $O(\tau^{|FSS^{\perp}|+|HSS^{\top}|})$, where τ is the number of parameter values in the MFS, and $|FSS^{\perp}|$ and $|HSS^{\top}|$ are two relatively small numbers and independent on the number of parameters n in one test case. Formula 3 is much more efficient at obtaining pending schemas when compared to the exhaustive methods which consecutively checks schemas in one failing test case, especially when n is large.

We conducted a series empirical studies on some real software systems with various number of parameters and values. We first evaluated the incompleteness of traditional covering arrays and different fault localization approaches in CT. We also compared the efficiency of three formulas in terms of obtaining pending schemas. Our results mainly shows that the incompleteness in very common in the covering arrays and MFS identification approaches. We also observed that Formula 3 is the most efficient formula among others in most cases.

Contributions of this paper:

- We showed 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 introduced the notion of the pending schema to evaluate the incompleteness of these models in combinatorial testing.
- We proposed several propositions to formulate the set of pending schemas and gave three equivalent formulas to obtain the pending schemas, based on which we reduced 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 independent on 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 evaluated 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 define the determinable schemas and pending schemas. Section 5 formally identify the characteristics of the pending schemas and give a efficient formula to obtain it. Section 6 evaluates the incompleteness of MFS identification approaches and compares the effectiveness of different approaches for obtaining pending schemas. Section 7 discusses the findings of our research works. Section 8 summarizes the related works. Section 9 concludes this paper and discusses the future works.

2 MOTIVATION

In this section, we will use several examples to show the incompleteness of traditional covering arrays and well-known MFS identification approaches, respectively. These examples are derived from the MS-Word example listed in the introduction. For simplification, we use integer 0 to represent the state *On* and 1 to represent the state *Off* for each option. For example, the second test case listed in Table 1 can be denoted as (1, 1, 0, 0). Also, we use the intuitive notation (.., v_{n_i} -, v_{n_k}, ..., -) to represent the schemas for the system, where v_{n_i} indicate the value that is assigned to the corresponding factor and '-' indicates that the corresponding factor is not in this schema. For example, (1, 1, -, -) represents the schema (Highlight: Off, Status Bar: Off) in this example. Note that we will introduce a more formal denotation of test case and schema in the following section.

Also, to understand the following examples, we first give two rules. The first rule is that all the schemas in a passing test case are non-faulty, i.e., will not cause failure. The second rule is that any schema contain a MFS is a faulty schema, i.e., will also cause the failure. These two rules are wildly used in the MFS identification approaches [5, 12, 13, 22]. We will discuss the justifications of these two rules later, as well as some issues if these two rules are not hold.

2.1 The incompleteness of covering array

We first consider the traditional covering arrays. To understand the incompleteness of the covering array, we need to check each schema in the test case of the covering array. As we said before, we use the same MS-Word example in the first section. Figure 1 lists the detail of this example. The test cases t_1 to t_5 constitute the covering array shown in Table 1. The complete set of the schemas of each test case is attached at the right side of the corresponding test case. For example, for test case t_1, i.e., (0, 0, 0, 0), all the possible schemas (0, -, -, -), (0, 0, -, -), ..., are listed at the right side of t_1. There are $2^4 - 1 = 15$ schemas in total for each test case in this example.

In this figure, the test case with dark color represents a failing test case, while the test case with white color is a passing test case. The schema with white color is non-faulty, i.e., will not cause a failure, while the schema with dark color is faulty, i.e., any test case contain this schema would fail after testing. At last, the schema with light dark color and dashed outline is the pending schema, indicating that we cannot still determine whether it is faulty or non-faulty.

In this figure, we can first observe that all the schemas in the passing test case is non-faulty. This result is according to the first rule we mentioned before. The second observation is that the schema with the maximal number of factors (4 factors) in a failing test case is a faulty schema. In fact, this schema is the failing test case itself, i.e., (1, 1, 0, 1). This is because the failing test case must contain at least one MFS (otherwise, it will not fail). Hence, the schema which is the test case itself must also contain at least one MFS. As a result, it must be faulty schema according to the second rule we mentioned before. The last observation is that the other schemas in this failing test case t_5 are not guaranteed to be faulty schemas. In fact, if we assume this test case only contains one MFS (1, 1, 0, 1), then all the other schemas can be non-faulty schemas. Hence, these schemas cannot be determined to be faulty or not if we focus on this failing test case alone. As a result, we label these schemas as pending schemas initially.

Combining the three observations, we can further remove some pending schemas in t_5 by selecting the schemas that have already been appeared in the passing test cases. These schemas are (1, -, -, -), (-, -, 0, -), (-, 1, -, -), and (-, -, -, 0), which are labeled as non-faulty schemas. At last, the determination results of these schemas of t_5 can be shown in the "*Status*" row. Note that except for

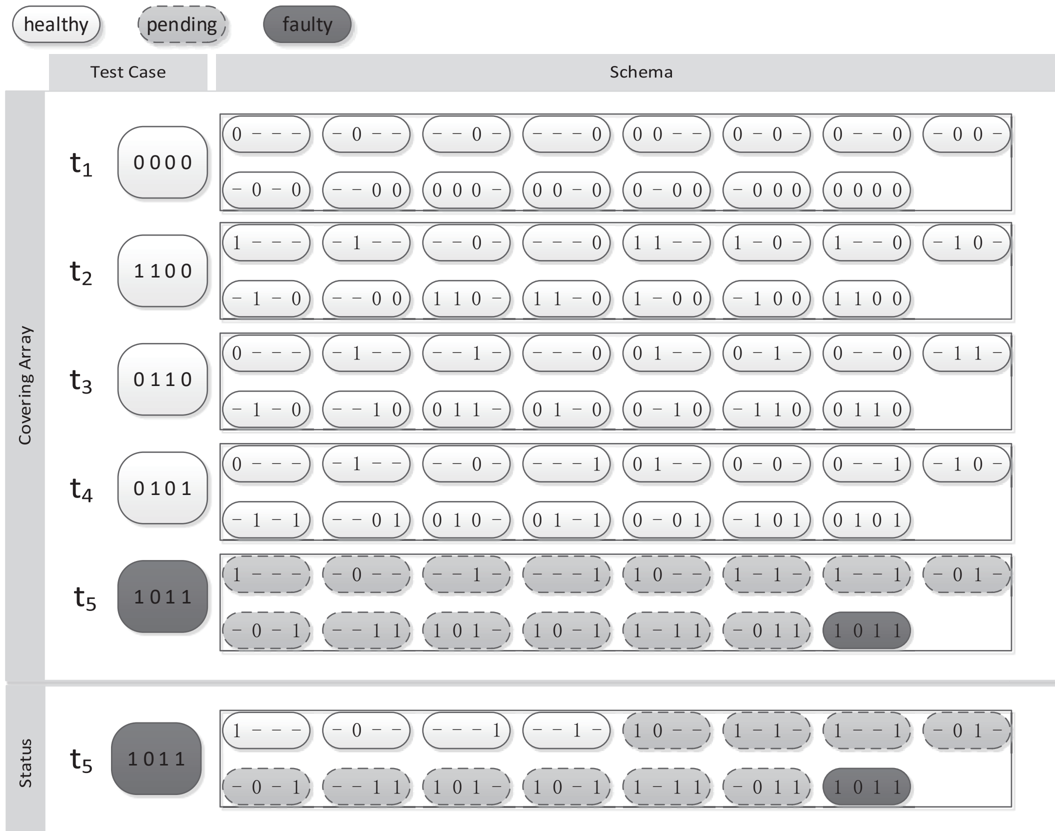

Fig. 1. The incompleteness of Covering array

these schemas that have been determined to be faulty and non-faulty, there still exist some pending schemas we cannot further removed by the original two rules. For example, (1, 0, -, -) didn't appear in any passing test case, and it did not contain any identified MFS. Hence, in this example, a single covering array is incomplete because of the existence of these pending schemas.

2.2 The incompleteness of OFOT

Since covering array alone cannot remove all the pending schema in the failing test cases, we need more information to satisfy this target. According to the second rule, i.e., the schema that contain the MFS is faulty schema, one method to reduce the number of pending schemas is to filter out those schemas which contain the MFS. However, without knowing the specific MFS in prior, we can only guarantee that the failing test case itself is faulty schema (it must contain the MFS). In fact, with the covering array alone, this is what we can only do to utilize the second rule.

Hence, to further reduce the set of pending schemas, we need to identify the MFS in the failing test case. One-Factor-One-Time (OFOT) [12] is one of the most wildly used MFS identification approach. It identifies the MFS by modifying the original failing test case to see whether the modification would break the MFS in it. More specifically, at each time, it modifies one factor of the original failing test case and keeps the remaining factors to be as the same as the original failing test case. By doing this, it generates one new test case at each time. It then tests the newly generated test case. If this newly generated test case passes, it indicates the modified factor break the MFS in the original failing test case, and therefore, the original factor in the failing test case is one factor in the MFS. Otherwise, the original factor in the failing test case is not the factor in the MFS if the newly generated test case fails.

Next, let us use OFOT to identify the MFS and help to narrow down the set of pending schemas of t_5 in the original covering array. First, we assume that there is one MFS (-, -, 1, 1) in failing test case t_5. Then OFOT will work as follows: it generates four additional test cases t_6 to t_9 as shown in Figure 2, respectively, each of which has one factor to be mutated from t_5. Since t_8 and t_9 passed after testing, the original two factors (-, -, 1, -) and (-, -, -, 1) in t_5 are two factors in the MFS. The fails of t_6 and t_7 shows that there is no other factors in this MFS. Hence, OFOT identified the schema (-, -, 1, 1) as the MFS, which is identical to the schema (-, -, 1, 1) that we set as MFS in prior.

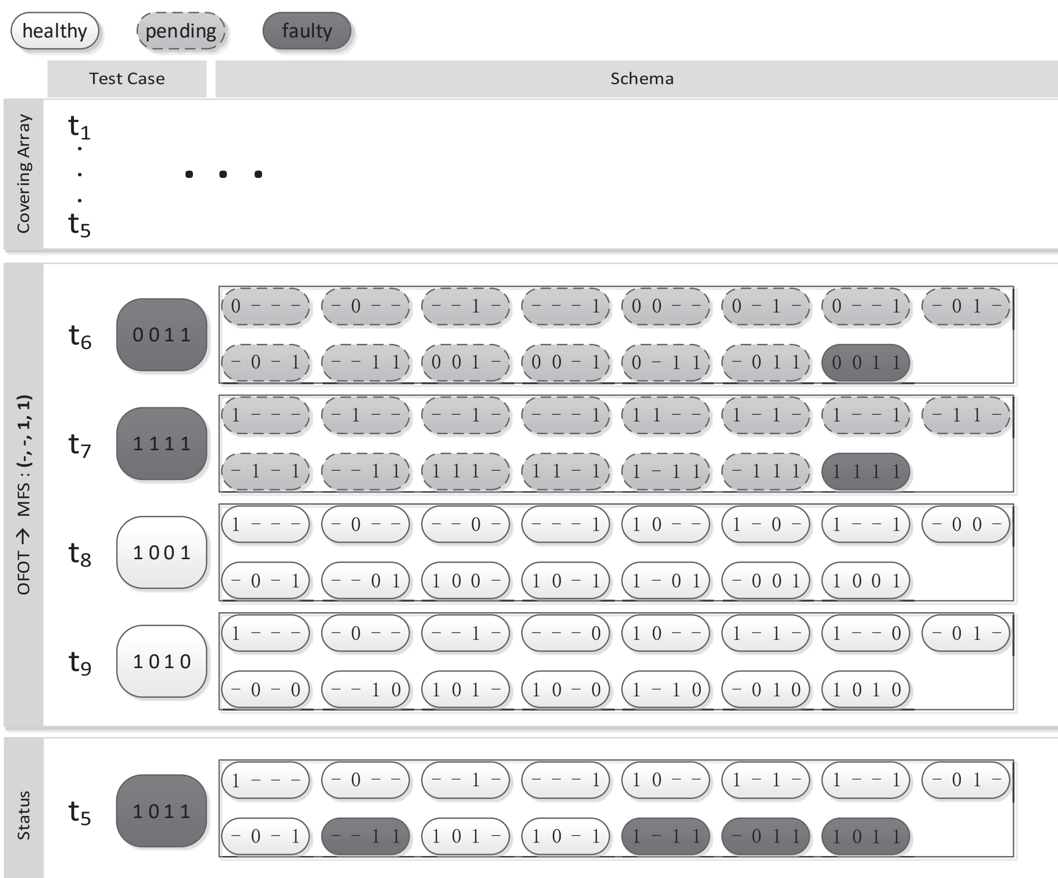

Fig. 2. OFOT with single MFS

To analyse the pending schemas, we first list all the schemas in each additional test case in Figure 2. The same as we observed from the covering array example, for the passing test case t_6 and t_7, all the schemas contained in it are non-faulty. For the failing test cases t_8 and t_9, we initially set the schemas which are failing test cases themselves as faulty schemas. Other schemas in these two failing test cases are all set to be pending schemas initially.

With these additional information, let us re-consider the status of the schemas in the original failing test case t_5. Firstly, as the identified MFS is (-, -, 1, 1), we can remove all the pending schemas of t_5 which contain this schema. These schemas are (-, -, 1, 1), (1, -, 1, 1), (-, 0, 1, 1), and (1, 0, 1, 1), respectively, and are labeled with dark color in the "*Status*" row for t_5 in Figure 2. Next, we remove all the pending schemas of t_5 which appeared in these two passing test case t_6 and t_7. As shown in the "*Status*" row of Figure 2, all the remaining pending schemas are removed and labeled with white color. Hence, we can learn that in this single MFS circumstance, OFOT works perfectly to remove all the pending schemas. However, when the failing test case contains multiple MFS, it does not go that well.

Now let us assume there are two MFS in the failing test case t_5, which are (1, 0, -, -) and (-, -, 1, 1), respectively. At this time, OFOT still generates the same four additional test cases, i.e., t_6 to t_9, as shown in Figure 3. But different from the example of single MFS, all the additional test cases failed at this time. This is because, the strategy of OFOT, i.e., mutating one factor at one time, cannot break all the MFS at the same time. As a result, it cannot identify any of the MFS.

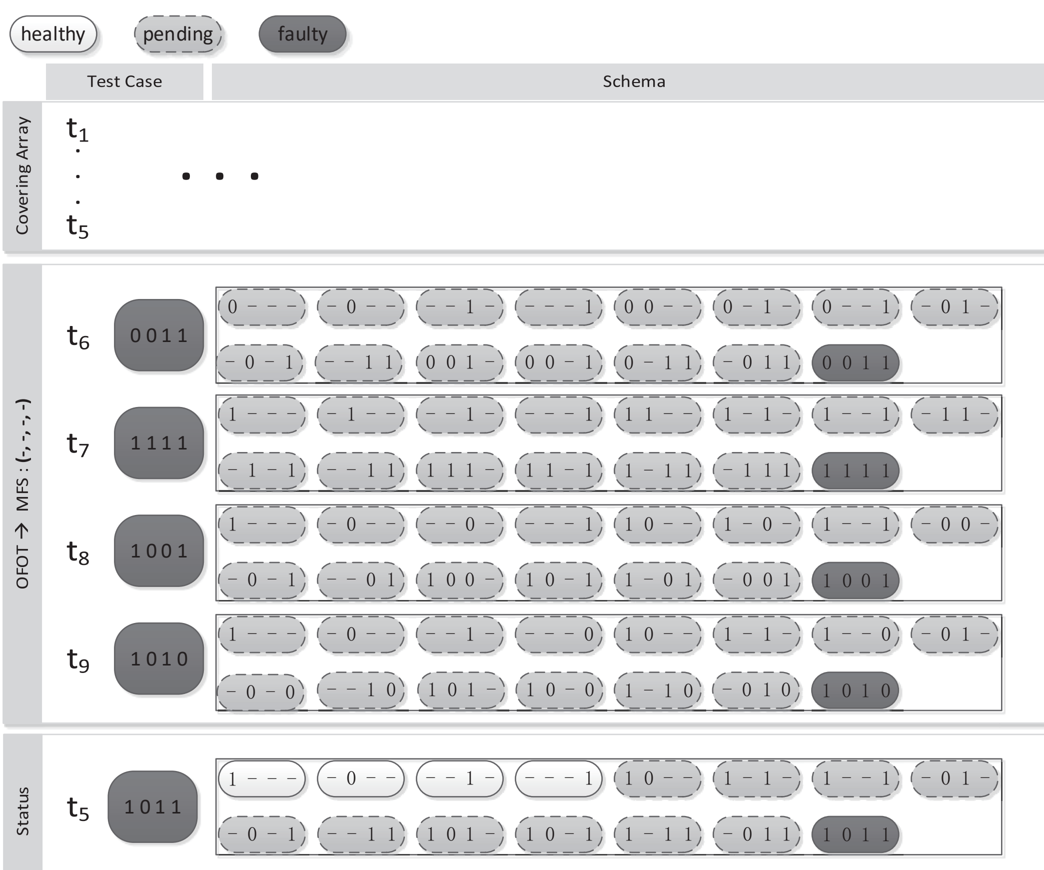

Fig. 3. OFOT with multi MFS

There are two negative influences of this result. First, as we cannot identify the MFS by OFOT, the pending schemas in t_5 that contain the MFS also cannot be determined. Second, as all the test cases failed after testing, we cannot remove any pending schemas in t_5 that appear in the passing test case. As a result, the status of the pending schemas of t_5 will evolve to the "*Status*" row of Figure 2. We can observe that the status of the schemas of t_5 is the same as the previous example with only using covering array alone. Hence, in the condition that one failing test case contains multiple MFS, the MFS identification approach OFOT is still incomplete.

2.3 The incompleteness of FIC

From the example of OFOT, we can learn the main cause of incompleteness of OFOT is that the failing test case contains multiple MFS. For this, FIC [22] (short for Faulty Interaction Characterization) augmented OFOT to handle the multiple MFS problem. FIC also mutates one factor at a time to generate one additional test case. The only difference is that it will not always rollback to the original value of one factor it has mutated when it goes on mutating other factors (only when a passing test case appears, it will rollback to the original value). This operation will break multiple MFS in one test case and finally there remains only one MFS to identify. We still use the same example of multiple MFS used in OFOT to illustrate how FIC works and to see whether FIC satisfies the completeness criteria. The result is shown in Figure 4.

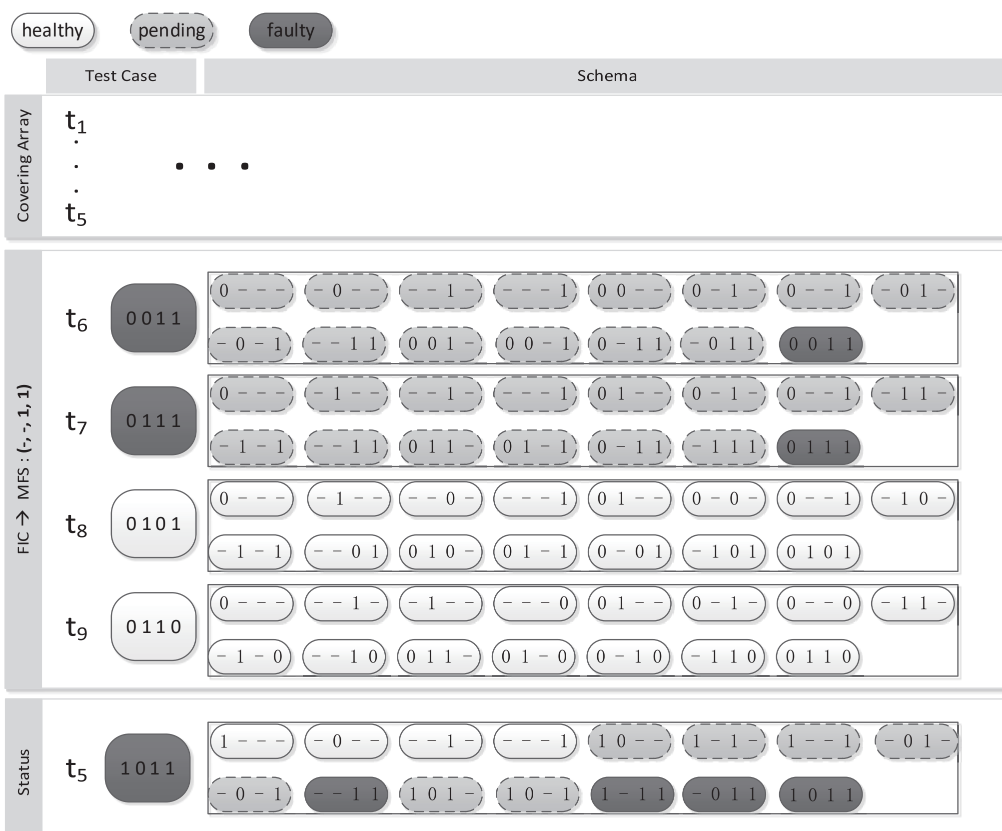

Fig. 4. FIC with multiple MFS

In Figure 4, FIC also generated four additional test cases t_6 to t_9. The first case t_6 is the same as the t_6 generated by OFOT, but the second additional test case t_7 generated by FIC is different from

t_7 generated by OFOT. This is because t_7 generated by FIC keeps the first value of t_6 (which is a failing test case) instead of the original first value of t_5. Note that FIC also keeps this value in the following generated test cases t_8 and t_9. By doing this, FIC forbids the appearance of the MFS (1, 0, -, -) later. The same as the first value of t_6, t_8 and t_9 keep the second value of t_7 because t_7 is also a failing test case. The failings of t_6 and t_7 indicated that there still exists other MFS in the original failing test case t_5, while the first and second factor is not in the remaining MFS.

With respect to the passing test case t_8, test case t_9 did not keep the third value of t_8 but rollback to the value of the original failing test case t_5. This is because, since t_8 passed after testing, there is no MFS in this test case. FIC should rollback this value to keep this factor of the MFS and check if there exists other factor of this MFS or not. After all, the passings of t_8 and t_9 indicated that (-, -, 1, -) and (-, -, -, 1) are two the factors in the remaining MFS. Hence, the MFS identified by FIC is (-, -, 1, 1).

Next, let us check status of the original failing test case t_5. The same as OFOT, we first remove the pending schemas that contain the MFS (-, -, 1, 1), and then remove the pending schemas that appear in the additional passing test case. The result is listed in the "*Status*" row of Figure 4. We can observe that there still exists 6 pending schemas, i.e., (1, 0, -, -), (1, -, 1, -), (1, -, -, 1), (1, 0, 1, -), (1, 0, -, 1), (-, 0, 1, -), and (-, 0, -, 1), respectively.

One reason for this incompleteness that FIC missed anther MFS in the failing test case t_5, which is (1, 0, -, -). The iterative version of FIC [22], i.e., FINOLP is designed to handle this problem. Specifically, after identifying one MFS in the original failing test case, FINOLP first generates one more test case by mutating the factors in the original failing test case which appear in the identified MFS. If the generated test case fails after testing, which indicates that there still exists other MFS, it will use FIC to identify the MFS in this generated test case. This process repeats until the test case which is generated by mutating all the factors in the identified MFS passes. Figure 5 shows the detail when applying FINOLP on this example.

In Figure 5, test cases t_1 to t_9 are the same as those of the example in Figure 4. After identifying the MFS (-, -, 1, 1), FINOLP first generated the test case t_{10} to check whether there exists other MFS in the original failing test case t_5 by mutating the values that appear in this MFS. Since t_{10} failed after testing, it repeated FIC approach on t_{10} to identified the remaining MFS. Therefore, it generated four additional test cases t_{11} to t_{14}. The passings of t_{11} and t_{12} indicated another MFS was (1, 0, -, -). FINOLP continued to check whether there exists any other MFS by generating the test case t_{15}. The passing of t_{15} showed that there did not exist any other MFS in the original failing test case t_5. Above all, FINOLP accurately identified all the MFS we have set in piror.

Next, the same as before, we use the two rules to check the status of the pending schemas in the original failing test case t_5. The result is shown in the "*Status*" row of Figure 5. We can observe that there still exists four schemas, which are (1, -, 1, -), (1, -, -, 1), (-, 0, 1, -), and (-, 0, -, 1), respectively. It is easy to find that these four schemas neither contain any MFS nor appears in any passing test cases. Hence, although FIC and FINOLP can handle the multiple MFS problem, it is still incomplete for the existence of these pending schemas.

2.4 Additional efforts to remove the pending schemas

Since Covering array, OFOT, FIC, and FINOLP cannot remove all the pending schemas, more efforts are needed to accomplish this goal. Note that to clear all the pending schemas is important, because some of them can be potential faulty schemas and may be harmful for this system under testing. For this, we decided to check these remaining pending schemas by generating more test cases that contained them and executing these test cases. If these test cases passes, we can directly use the first rule we mentioned before to determine these schemas to be non-faulty schemas. Otherwise, we need to adopt other methods to determine these schemas to be faulty or not.

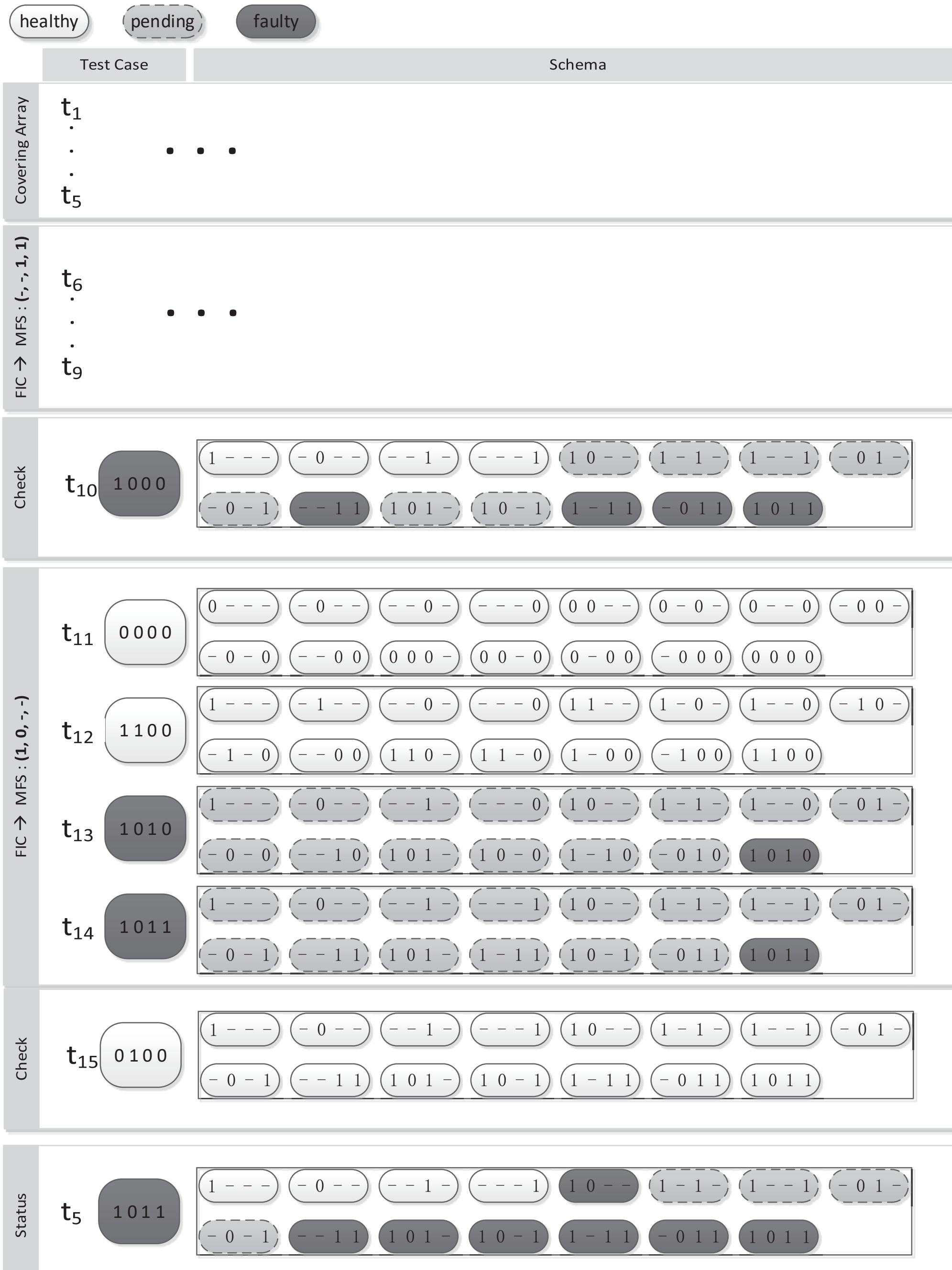

Fig. 5. FINOVLP with multiple MFS

As for this example, we generated four test cases to contain these pending schemas one by one (Note that in this example, we cannot generate one test case contain more than one pending schema

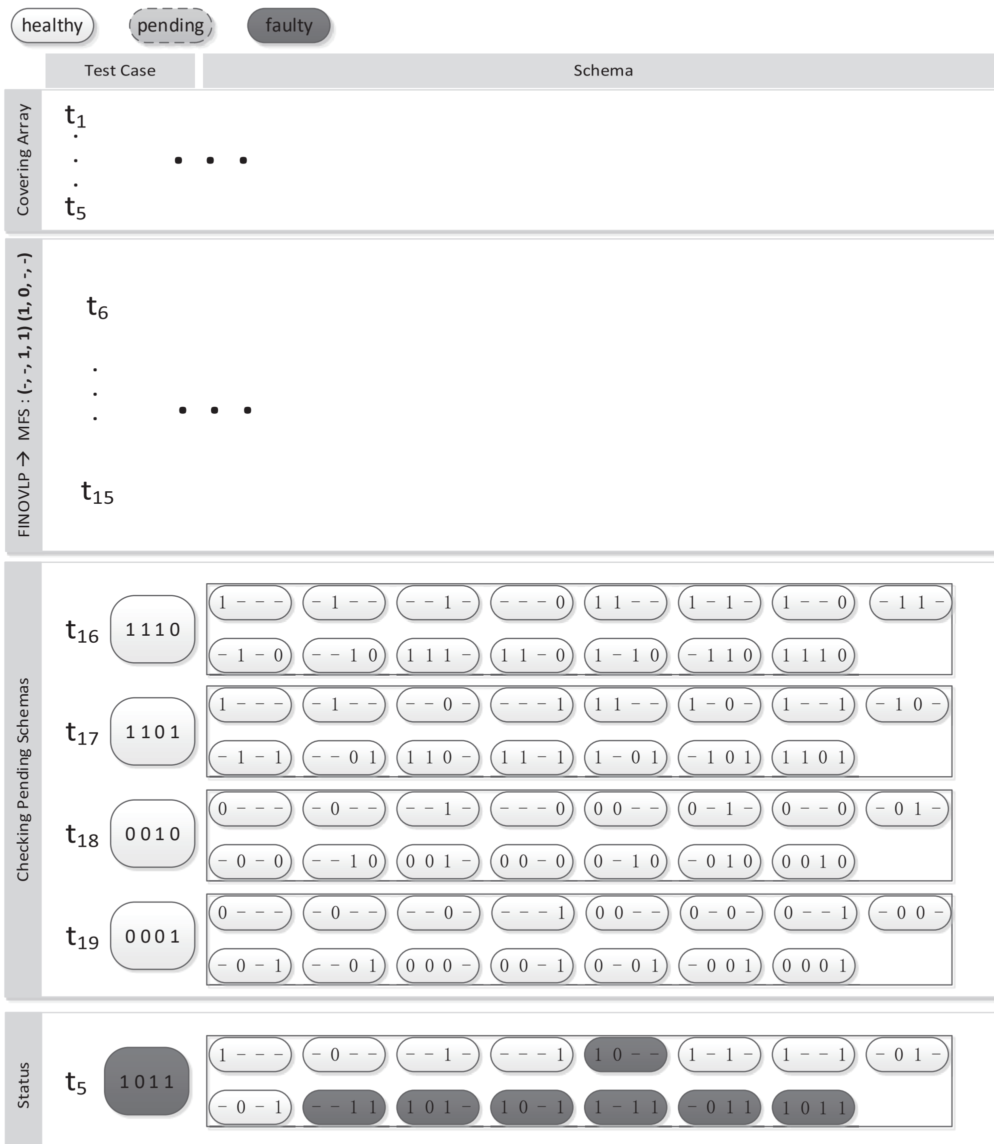

Fig. 6. Additional efforts to remove pending schemas

without including any MFS). The result is shown in Figure 6. In this figure, all the additional test cases, i.e., t_{16} to t_{19}, passed after testing. Hence, all the remaining schemas are non-faulty schemas. The final status of the pending schemas of t_5 is shown in the "*Status*" row of Figure 6. We can observe that all the schemas in t_5 are determined to be non-faulty or faulty. Hence, in this condition, we can guarantee that the test of t_5 is complete.

Note that if any of the test case (t_{16} to t_{19}) failed after testing, we cannot determine whether the corresponding pending schema in that test case is faulty or not. In this case, we need to generate more test cases to determine the status of this pending schema (If this schema is a faulty schema, the cost will increase exponentially according to the formal definition of faulty schema which will be given later).

2.5 A summary

There are two main observations from this section. First, traditional covering array and MFS identification approaches are still incomplete in terms of the existence of pending schemas which cannot be determined to faulty or non-faulty. Second, to remove all the pending schemas is time-consuming. In fact, just listing all the schemas and checking them one by one is inefficient. In this example, we need to check $2^4 - 1 = 5$ schemas for each test case. However, with the increase of the number of factors in one test case, the cost for checking the pending schemas increase exponentially.

For all of these, we need to theoretically analyze the properties of the pending schemas in the failing test cases and to give a more efficient method to obtain them.

3 BACKGROUND

This section presents some definitions and propositions to give a formal model for CT. Without loss of generality, 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).

3.1 Test cases and schemas

In this subsection, we will formally define the test case and schema. We will also give the properties to reveal the relationships between them. In these propositions, Proposition are originated from our previous work .

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.

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), ..., $(p_n, v_n)\}$. Also, for any schema, say, s, its degree can be denoted as $|s|$ because a schema is essentially a set of parameter values.

Note that this definition of schema is a formal description of schemas we discussed in the section of *Motivation*. For example, the schema (1, -, 0, -) is exactly the schema $\{ (p_1, 1), (p_3, 0) \}$ here. We use this formal definition because it benefits the description of the following theoretical analysis, including these propositions and proofs.

Definition 3.3. Let s_1, s_2 be two schemas in SUT. If $\forall (p_{x_1}, v_{x_1}) \in s_1$, $(p_{x_1}, v_{x_1}) \in s_2$, then s_1 is the $sub-schema$ of s_2, and s_2 the $super-schema$ of s_1, which can be denoted as $s_1 \preceq s_2$. Further, if $|s_1| < |s_2|$, we call s_1 the $real-sub-schema$ of s_2, which is denoted as $s_1 \prec s_2$.

For example, the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$ is a sub-schema (also is a real-sub-schema) of the 3-degree schema $\{(p_1, 1), (p_2, 0), (p_4, 1)\}$.

According to the definition of schemas, it is easy to find the following three properties of schemas.

Proposition 3.4 (Reflexivity). *For any schema s, $s \preceq s$.*

Proof. Since $\forall (p_{x_i}, v_{x_i}) \in s$, $(p_{x_i}, v_{x_i}) \in s$. Hence, $s \preceq s$ according to the definition of 3.3. □

This proposition tells that any schema is the subschema of itself. For example the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$ is a sub-schema of itself $\{(p_1, 1), (p_2, 0)\}$.

Proposition 3.5 (Antisymmetry). *For schemas s_1, s_2, if $s_1 \preceq s_2$ and $s_2 \preceq s_1$, then $s_1 = s_2$.*

Proof. As $s_1 \preceq s_2$, then $\forall (p_{x_i}, v_{x_i}) \in s_1, (p_{x_i}, v_{x_i}) \in s_2$. This indicates that the parameter value set s_1 is the subset of s_2, i.e., $s_1 \subseteq s_2$. Also as $s_2 \preceq s_1$, $s_2 \subseteq s_1$. Since, $s_1 \subseteq s_2$ and $s_2 \subseteq s_1$, we have the two parameter value sets s_1 and s_2 are equal, i.e., $s_1 = s_2$. □

This proposition shows that for two distinct schemas, one cannot be both super-schema and sub-schema of the other schema. For example, the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$ is a sub-schema of the 3-degree schema $\{(p_1, 1), (p_2, 0), (p_4, 1)\}$, but not a super-schema of this 3-degree schema. Only two equal schemas can be both the sub-schema and super-schema of each other. For example, the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$ is both the sub-schema and super-schema of itself.

Proposition 3.6 (Transitive). *Given schemas s_1, s_2, and s_3, if $s_1 \preceq s_2$, $s_2 \preceq s_3$, then $s_1 \preceq s_3$.*

Proof. As $s_1 \preceq s_2$, then $\forall (p_{x_i}, v_{x_i}) \in s_1, (p_{x_i}, v_{x_i}) \in s_2$. Also, $s_2 \preceq s_3$, then $\forall (p_{x_i}, v_{x_i}) \in s_2, (p_{x_i}, v_{x_i}) \in s_3$. Hence, $\forall (p_{x_i}, v_{x_i}) \in s_1, (p_{x_i}, v_{x_i}) \in s_3$. According to definition 3.3, $s_1 \preceq s_3$. □

This proposition shows the transitivity of the subsuming relationships of schemas. For example, the 1-degree schema $\{(p_1, 1)\}$ is a sub-schema of the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$, and this 2-degree schema is a sub-schema of the 3-degree schema $\{(p_1, 1), (p_2, 0), (p_4, 1)\}$. Therefore, the 1-degree schema $\{(p_1, 1)\}$ is also a sub-schema of the 3-degree schema $\{(p_1, 1), (p_2, 0), (p_4, 1)\}$.

These three propositions together show that the schema set $(s, \preceq)$ is a partial order set. Next, we will introduce the relationships between schemas and tests.

Definition 3.7. For a schema s, if there exists one test case t, such that each parameter value in the schema s is also in the test case t, i.e., $\forall (p_{x_i}, v_{x_i}) \in s, (p_{x_i}, v_{x_i}) \in t$, then we call the test case t *hits* or *contains* the schema s, and test case t is one *container* of schema s.

Note that test case is also a n-degree schema, hence when a test case t hits one schema s, we can denote this relationship simply as $s \preceq t, |t| = n$. As an example, test case $\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$ hits the 2-degree schema $\{(p_1, 1), (p_2, 0)\}$.

Definition 3.8. For a schema s, the set of all the possible containers of it, i.e., the set of all the possible test cases that contain this schema, is called the *whole set* of containers of this schema s, which is denoted as $\mathcal{T}(s)$.

Formally, $\mathcal{T}(s) = \{t | s \preceq t, |t| = n\}$. As an example, let us consider the SUT that has four parameters and each parameter has two values (All the following examples are based on the same SUT). Let schema s be $\{(p_1, 1), (p_2, 0)\}$. Table 2 shows $\mathcal{T}(s)$. It can be observed that each possible test case in the SUT that contains schema s is listed in Table 2.

With respect the the whole set of containers of one schema, we have the following property.

Proposition 3.9 (smaller schema has a larger $\mathcal{T}(s)$). *For schemas s_1 and s_2, if $s_1 \preceq s_2$, then $\mathcal{T}(s_2) \subseteq \mathcal{T}(s_1)$.*

Table 2. Whole set of containers of $\{(p_1, 1), (p_2, 0)\}$

Schema	
s	$\{(p_1, 1), (p_2, 0)\}$
$\mathcal{T}(s)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
t_3	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$
t_4	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$

Proof. $\forall t \in \mathcal{T}(s_2)$, $s_2 \preceq t$. Since $s_1 \preceq s_2$, we have $s_1 \preceq t$ according to Proposition 3.6. That is, t contains s_1. Therefore, $t \in \mathcal{T}(s_1)$. Hence, $\mathcal{T}(s_2) \subseteq \mathcal{T}(s_1)$. □

As an example, let schema s_1 be $\{(p_1, 1), (p_2, 0)\}$, and schema s_2 be $\{(p_1, 1), (p_2, 0), (p_3, 0)\}$, which is one of the super-schemas of s_1. Table 3 shows the whose set of containers of s_1 and s_2. We can observe that $\mathcal{T}(s_1)$ subsumes $\mathcal{T}(s_2)$.

Table 3. Comparison of the whole set of containers of two subsuming schemas

Schema	
s_1	$\{(p_1, 1), (p_2, 0)\}$
$\mathcal{T}(s_1)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
t_3	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$
t_4	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$
Schema	
s_2	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$
$\mathcal{T}(s_2)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

Definition 3.10. For a test case t, all the schemas that it can hit is called the *whole set* of hit schemas of t, which is denoted as $\mathcal{I}(t)$.

Formally,$\mathcal{I}(t) = \{s | s \preceq t\}$. As an example, let test case t be $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$. Table 4 shows $\mathcal{I}\ (t)$. We can find it is a set of $2^4 - 1 = 15$ schemas.

Definition 3.11. For a set of test case T, all the schemas that these test cases can hit is called the *whole set* of hit schemas of T, which is denoted as $\mathcal{I}(T)$.

$\mathcal{I}(T)$ extends $\mathcal{I}(t)$ to a set of test cases. Formally, $\mathcal{I}(T) = \bigcup_{t \in T} \mathcal{I}(t)$.

Definition 3.12. All the schemas that are **only** contained in test set T is called the *special schemas* of test case set T, which is denoted as $\mathcal{S}(T)$.

Formally, $\mathcal{S}(T) = \mathcal{I}(T) \backslash \mathcal{I}(\bar{T})$, where $\bar{T}$ is the complementary test set of T, i.e., $T \bigcup \bar{T} = T^*, T \bigcup \bar{T} = \emptyset$. Here, T^* indicates all the possible test cases in the SUT. Based on this formula, we can learn that the special schemas are essentially the whole set of hit schemas of test case T, except for those hit schemas of other test cases.

Table 4. An example of whole set of hit schemas of one test case

Test Case	
t	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
$\mathcal{I}(t)$	
s_1	$\{(p_1, 1)\}$
s_2	$\{(p_2, 0)\}$
s_3	$\{(p_3, 0)\}$
s_4	$\{(p_4, 0)\}$
s_5	$\{(p_1, 1),(p_2, 0)\}$
s_6	$\{(p_1, 1),(p_3, 0)\}$
s_7	$\{(p_1, 1),(p_4, 0)\}$
s_8	$\{(p_2, 0),(p_3, 0)\}$
s_9	$\{(p_2, 0),(p_4, 0)\}$
s_{10}	$\{(p_3, 0),(p_4, 0)\}$
s_{11}	$\{(p_1, 1),(p_2, 0),(p_3, 0)\}$
s_{12}	$\{(p_1, 1),(p_2, 0),(p_4, 0)\}$
s_{13}	$\{(p_1, 1),(p_3, 0),(p_4, 0)\}$
s_{14}	$\{(p_2, 0),(p_3, 0),(p_4, 0)\}$
s_{15}	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$

As an example, let test case t_1 be $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$ and t_2 be $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$. Let the test case set T be $\{ t_1 , t_2 \}$. Table 5 shows the schemas of $\mathcal{S}(T)$. We can observe that there are only three schemas s_1, s_2, and s_3 in this schema set. Other hit schemas of T that can appear in other test cases do not belong to $\mathcal{S}(T)$. For example, schema $\{(p_1, 1), (p_2, 0)\}$ can appear in the test case $\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$, which is not in T. Hence, $\{(p_1, 1), (p_2, 0)\}$ is only one hit schema of T, but not the special schema of T.

Table 5. An example of special schemas of test case set

Test Case set T	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{S}(T)$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$
s_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
s_3	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

With respect to the special schema set of test case T, we can obtain the following property.

Proposition 3.13 (Whole container of special schema set). *For any test set T of the SUT,* $\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s) = T$.

Proof. As $\mathcal{S}(T) = \mathcal{I}(T) \backslash \mathcal{I}(\bar{T})$, $\forall s \in \mathcal{S}(T)$, $s \in \mathcal{I}(T)$ *and* $s \notin \mathcal{I}(\bar{T})$. Then $\forall t \in \mathcal{T}(s)$, t *contains* s, indicating that $t \in T$. Hence, $\mathcal{T}(s) \subseteq T$. Then $\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s) \subseteq T$.

On the other hand, $\forall t \in T$, $\exists s' \in \mathcal{I}(t)$, such that $s' \notin \mathcal{I}(\bar{T})$ (at least it holds when $s' = t$). Hence, $s' \in \mathcal{S}(T)$. Obviously $t \in \mathcal{T}(s') \subseteq \bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s)$. Therefore, $T \subseteq \bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s)$.

Since $\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s) \subseteq T$ and $T \subseteq \bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s)$, we have $\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s) = T$. □

Proposition 3.13 revels the relationship between test case set and its special schemas. That is, for a specific test set, the whole set of containers of its special schemas are equal to this original test set. As an example, let test case t_1, t_2 still be $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$, $\{(p_1, 1)$, and $(p_2, 0), (p_3, 0), (p_4, 1)\}$, respectively. Let the test case set T be $\{ t_1 , t_2\}$. Table 6 specifically shows the whole set of containers for each special schema for test case set T, as well as their union. We can observe that their union, i.e., $\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s)$ is equal to the original test set T.

Table 6. An example of the whole set of containers of special schemas

Test Case set	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{S}(T)$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$
s_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
s_3	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{T}(s_1)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{T}(s_2)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
$\mathcal{T}(s_3)$	
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\bigcup_{s \in \mathcal{S}(T)} \mathcal{T}(s) = \mathcal{T}(s_1) \bigcup \mathcal{T}(s_2) \bigcup \mathcal{T}(s_3)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

Definition 3.14. A set of the **minimal** schemas that are only contained in test set T is called the *minimal special schemas* of test set T, which is denoted as $C(T)$.

Formally, $C(T) = \{s | s \in \mathcal{S}(T)\ and\ \nexists s' \prec s, s.t., s' \in \mathcal{S}(T)\}$. As an example, let test set T still be $\{ t_1, t_2 \}$, where t_1 is $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$ and t_2 is $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$. Table 7 shows the $C(T)$. We can observe that there is only one schema, i.e., s_1, in this schema set. The other two schemas in $\mathcal{S}(T)$, i.e., s_2 and s_3, are the super-schemas of this schema, therefore, they are eliminated from this set.

Table 7. An example of minimal special schemas of test case set

Test Case set T	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$C(T)$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$

According to the definition of $C(T)$, one obvious property is $C(T) \subseteq \mathcal{S}(T)$. The second property is that for any schema in $\mathcal{S}(T)$, it must be the super schema of one element of $C(T)$, i.e., $\forall s \in \mathcal{S}(T), \exists s' \in C(T), s.t., s' \preceq s$. Besides these two obvious properties, we also have the following important property about the minimal special schema set.

Proposition 3.15 (Whole container of minimal special schema set). *For any test set T of the SUT,* $\bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s) = T$.

Proof. As $\mathcal{C}(T) \subseteq \mathcal{S}(T)$, it is then obviously $\bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s) \subseteq \bigcup_{s\in\mathcal{S}(T)} \mathcal{T}(s)$. Hence, we just need to prove that $\bigcup_{s\in\mathcal{S}(T)} \mathcal{T}(s) \subseteq \bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s)$.

$\forall t \in \bigcup_{s\in\mathcal{S}(T)} \mathcal{T}(s)$, $\exists s \in \mathcal{S}(T)$, $s.t., t \in \mathcal{T}(s)$. According to the definition of $\mathcal{C}(T)$, $\exists s' \in \mathcal{C}(T), s.t., s' \preceq s$. Correspondingly $\mathcal{T}(s) \subseteq \mathcal{T}(s')$ by Proposition 3.9. Hence, $t \in \mathcal{T}(s') \subseteq \bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s)$.

Therefore, $\bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s) = \bigcup_{s\in\mathcal{S}(T)} \mathcal{T}(s) = T$. □

This proposition revels the relationship between the minimal special schemas and the corresponding test set. That is, for a specific test set, the whole set of containers of its minimal special schemas are equal to this original test set. As an example, we still consider the test case set we used before. Table 6 specifically shows the whole set of containers for each minimal special schema for test case set T, as well as their union (Note in this example, there is only one minimal special schema). We can observe that their union, i.e., $\bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s)$ is equal to the original test set T.

Table 8. An example of the whole set of containers of minimal special schemas

Test Case set	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{C}(T)$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$
$\mathcal{T}(s_1)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\bigcup_{s\in\mathcal{C}(T)} \mathcal{T}(s) = \mathcal{T}(s_1)$	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

Next, we consider the minimal special schemas for a subset of test case set T.

Proposition 3.16 (Whole set of containers is the subset of test set T). *For any test set T and schema s of the SUT, if $\mathcal{T}(s) \subseteq T$, then $s \in \mathcal{S}(T)$.*

Proof. Assume $s \notin \mathcal{S}(T)$, i.e., $s \notin \mathcal{I}(T)\backslash\mathcal{I}(\bar{T})$, then $s \in \mathcal{I}(\bar{T})$. It indicates that $\exists t \in \bar{T}, t \in \mathcal{T}(s)$, which contradicts that $\mathcal{T}(s) \subseteq T$. Therefore, $s \in \mathcal{S}(T)$. □

Proposition 3.16 tells that for any schema, as long as its whole set of ontainers is subset of a test set T, then this schema must be one of the special schema of this test set T. As an example, consider test set T to be $\{t_1, t_2\}$, where t_1 and t_2 are the same as the previous examples. Let the schema s' to be $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$. We can learn that $\mathcal{T}(s')$ is $\{ \{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\} \}$, which is one subset of T. Hence, s' is one special schema of T. The details of this example are shown in Table 9.

One step further, for two subsuming test case set, we have the relationship between their minimal special schemas as the following.

Proposition 3.17 (Minimal special schemas in the smaller test set). *For T_1 and T_2 of the SUT with $T_2 \subseteq T_1$, $\forall s_2 \in \mathcal{C}(T_2)$, $\exists s_1 \in \mathcal{C}(T_1), s.t., s_1 \preceq s_2$.*

Proof. $\forall s_2 \in \mathcal{C}(T_2)$, $\mathcal{T}(s_2) \subseteq T_2 \subseteq T_1$. According to Proposition 3.16, $s_2 \in \mathcal{S}(T_1)$. By definitions of $\mathcal{S}(T)$ and $\mathcal{C}(T)$, $\exists s_1 \in \mathcal{C}(T_1), s.t., s_1 \preceq s_2$. □

Table 9. An example of the schema of which the whole set of containers is the subset of test set T

Test Case set T	
t_1	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,0)\}$
t_2	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,1)\}$
$\mathcal{S}(T)$	
s_1	$\{(p_1,1),(p_2,0),(p_3,0)\}$
s_2	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,0)\}$
s_3	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,1)\}$
Schema s'	
s'	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,0)\}$
$\mathcal{T}(s')$	
t_2	$\{(p_1,1),(p_2,0),(p_3,0),(p_4 0)\}$

Proposition 3.17 tells that for two subsuming test set, the minimal special schema of the smaller test set (subset) must be the super-schema of one of the minimal special schemas of the larger test set (superset). As an example, consider the test set T_1 to be $\{t_1, t_2\}$, and T_2 to be $\{t_1\}$, where t_1 and t_2 are the same as previous examples. It is easy to find $T_2 \subseteq T_1$. Table 10 shows the minimal special schemas of these two test sets. We can find that the minimal special schema of T_2 is the super-schema of the minimal special schema of T_1.

Table 10. An example of the minimal special schemas of two subsuming test set

Test Case set T_1	
t_1	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,0)\}$
t_2	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,1)\}$
$C(T_1)$	
s_1	$\{(p_1,1),(p_2,0),(p_3,0)\}$
Test case set T_2	
s'	$\{(p_1,1),(p_2,0),(p_3,0),(p_4,0)\}$
$C(T_2)$	
s_2	$\{(p_1,1),(p_2,0),(p_3,0),(p_4 0)\}$

These propositions build the foundation for the analysis of minimal failure-causing schemas we will discuss later.

3.2 Faulty and healthy

We consider any abnormally execution of a test case as a *failure*, e.g., a thrown exception, compilation error, assertion failure or constraint violation. Such a test case (abnormally executed) is called a *failing* case. Otherwise, a test case is a *passing* test case if it normally executed without triggering any failure. In this paper, we focus on the failures that are related with schemas. That is, the failure discussed in this paper is caused by or triggered by specific input schemas. To facilitate our discussion, we introduce the following two 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, while our previous study gives the multiple execution to alleviate such problem.

Assumption 2. *A failure would always be detected by testers.*

This assumption shows that we can always observe the failure by failing test case. 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. Our previous study discussed the problem and some related consequences caused if this assumption does not hold. In this paper, we will also discuss the impacts on caused by these two assumptions, as well as how to alleviate them later.

Next, let us discuss the relationships between schemas and failures.

Definition 3.18. A schema is *faulty schema* if and only if all the test cases in its *whole set* of containers, i.e., $\mathcal{T}(s)$, are failing test cases.

This definition tells that to determine a schema to be faulty schema, we need to make sure all the possible test cases that contain it must be failing test cases. The justifications of this definition of faulty schemas are two points. First, in this paper, the failure that we focus is triggered by some specific schema. Second, a failure is expected to be observed. According to these two points, we can conclude that if a schema is a faulty schema, all the test cases contain it will trigger a failure caused by this schema. Table 11 shows an faulty schema example, i.e., $\{(p_1, 1), (p_2, 0)\}$. We can observe that all the test cases that contain it are failing test cases.

Table 11. A faulty schema $\{(p_1, 1), (p_2, 0)\}$

Schema		
s	$\{(p_1, 1), (p_2, 0)\}$	
$\mathcal{T}(s)$		
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$	Fail
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$	Fail
t_3	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$	Fail
t_4	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$	Fail

As the other side of this coin, the non-faulty schema is defined as the following.

Definition 3.19. A schema is *healthy schema* if and only if at least one test case of its whole set of containers is a passing test case.

Formally, a schema s is healthy schema if and only if $\exists t \in \mathcal{T}(s)$, s.t., t is a passing test case. This definition tells that a healthy schema is not the cause of a failure if there is at least one passing test case contains it. This is because if it is a faulty schema, non-test case that contains it is a passing test case (the failure will always be observed). Note that the healthy schema is the non-faulty schema we introduced in the previous section. Table 12 shows an healthy schema example, i.e., $\{(p_1, 1), (p_2, 0)\}$. We can observe that test case t_1 contains it and passed after execution.

With respect to healthy schemas and faulty schemas, we conclude the following propositions in this paper.

Table 12. A healthy schema $\{(p_1, 1), (p_2, 0)\}$

Schema		
s	$\{(p_1, 1), (p_2, 0)\}$	
$\mathcal{T}(s)$		
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$	Pass
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$	Fail
t_3	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$	Fail
t_4	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$	Fail

Proposition 3.20 (Sub of Healthy). *Given a healthy schema s_1, then $\forall s_2, s_2 \preceq s_1$, s_2 is a healthy schema.*

Proof. According to the definition of healthy schema, $\exists t \in \mathcal{T}(s_1)$, t is a passing test case. Obviously, $s_1 \preceq t$. Then $\forall s_2 \preceq s_1$, we have $s_2 \preceq s_1 \preceq t$ according to Proposition 3.6. That is, $t \in \mathcal{T}(s_2)$. According to the definition of healthy schema, s_2 is a healthy schema. □

This proposition shows that any subschema of a healthy schema is also a healthy schema. In fact, this proposition can deduce the first rule in Section 2.

As an example, if schema $\{(p_1, 1), (p_2, 0)\}$ is a healthy schema, then its sub-schemas, i.e., $\{(p_1, 1)\}$ and $\{(p_2, 0)\}$ are all healthy schemas.

Proposition 3.21 (Super of Faulty). *Given a faulty schema s_1, then $\forall s_2, s_1 \preceq s_2$, s_2 is a faulty schema.*

Proof. Since $s_1 \preceq s_2$, $\mathcal{T}(s_2) \subseteq \mathcal{T}(s_1)$ according to the Proposition 3.9. Also, as s_1 is a faulty schema, all the tests in $\mathcal{T}(s_1)$ are failing test cases. Therefore, all the tests in $\mathcal{T}(s_2)$ are also failing test cases. According to the definition of faulty schema, s_2 is a faulty schema. □

This proposition shows that any super-schema of a faulty schema is also a faulty schema. Note that his proposition can deduce the second rule in Section 2.

As an example, if schema $\{(p_1, 1), (p_2, 0), (p_3, 0)\}$ is a faulty schema, then its super-schemas, i.e., $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$ and $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$ are all faulty schemas.

Next, let us discuss the relationships among tests, faulty schemas, and healthy schemas. We let the set of failing test cases in the SUT be T_{fail}, and the passing test cases in the SUT be T_{pass}. Obviously, $T_{pass} \bigcap T_{fail} = \emptyset$ and $T_{pass} \bigcup T_{fail} = T^*$, where T^* indicates all the possible test cases in the SUT. In other word, $T_{pass} = \overline{T_{failing}}$. We can further conclude the following propositions.

Proposition 3.22 (Healty schemas and passing test set). *Given T_{pass} in the SUT, all the healthy schemas are in the set $\mathcal{I}(T_{pass})$.*

Proof. We first prove that, $\forall s \in \mathcal{I}(T_{pass})$, it must be a healthy schema. Obviously $\forall s \in \mathcal{I}(T_{pass})$, $\exists t \in T_{pass}, s \preceq t$. That is, $\exists t \in T_{pass}, t \in \mathcal{T}(s)$, which indicates that s is a healthy schema according to the definition of healthy schema.

We second prove that, for any healthy schema s, it must have $s \in \mathcal{I}(T_{pass})$. According to the definition of healthy schema, we have for any healthy schema s, $\exists t \in T_{pass}, t \in \mathcal{T}(s)$. That is, $\exists t \in T_{pass}, s \preceq t$. Hence, $s \in \mathcal{I}(t) \subseteq \mathcal{I}(T_{pass})$. □

Proposition 3.22 tells that the whole set of hit schemas of passing test case set are healthy schemas. As an example, consider the test case set T_{pass} has two test cases, which are $\{(p_1, 0), (p_2, 0), (p_3, 0), (p_4, 0)\}$, and $\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$, respectively. Then all the healthy schemas are in the set $\mathcal{I}(T_{pass})$, which are listed in Table 13.

Table 13. An example of the healthy schemas when given passing test case set

Passing Test Case Set T_{pass}	
t_1	$\{(p_1,0), (p_2,0), (p_3,0), (p_4,0)\}$
t_2	$\{(p_1,1), (p_2,0), (p_3,0), (p_4,0)\}$
$\mathcal{I}(T_{pass})$	
s_1	$\{(p_1,0)\}$
s_2	$\{(p_1,1)\}$
s_3	$\{(p_2,0)\}$
s_4	$\{(p_3,0)\}$
s_5	$\{(p_4,0)\}$
s_6	$\{(p_1,0),(p_2,0)\}$
s_7	$\{(p_1,0),(p_3,0)\}$
s_8	$\{(p_1,0),(p_4,0)\}$
s_9	$\{(p_1,1),(p_2,0)\}$
s_{10}	$\{(p_1,1),(p_3,0)\}$
s_{11}	$\{(p_1,1),(p_4,0)\}$
s_{12}	$\{(p_2,0),(p_3,0)\}$
s_{13}	$\{(p_2,0),(p_4,0)\}$
s_{14}	$\{(p_3,0),(p_4,0)\}$
s_{15}	$\{(p_1,0),(p_2,0),(p_3,0)\}$
s_{16}	$\{(p_1,0),(p_2,0),(p_4,0)\}$
s_{17}	$\{(p_1,0),(p_3,0),(p_4,0)\}$
s_{18}	$\{(p_1,1),(p_2,0),(p_3,0)\}$
s_{19}	$\{(p_1,1),(p_2,0),(p_4,0)\}$
s_{20}	$\{(p_1,1),(p_3,0),(p_4,0)\}$
s_{21}	$\{(p_2,0),(p_3,0),(p_4,0)\}$
s_{22}	$\{(p_1,0), (p_2,0), (p_3,0), (p_4,0)\}$
s_{23}	$\{(p_1,1), (p_2,0), (p_3,0), (p_4,0)\}$

Proposition 3.23 (Faulty schemas and failing test set). *Given T_{fail} in the SUT, all the faulty schemas are in the set $\mathcal{S}(T_{fail})$.*

Proof. We first prove that, $\forall s \in \mathcal{S}(T_{fail})$, it must be a faulty schema. Obviously $\forall s \in \mathcal{S}(T_{fail})$, $\mathcal{T}(s) \subseteq T_{fail}$ according to Proposition 3.13, indicating that s is a faulty schema according to the definition of faulty schema.

We second prove that, for any faulty schema s, it must have $s \in \mathcal{S}(T_{fail})$. According to the definition of faulty schema, we have for any faulty schema s, $\mathcal{T}(s) \subseteq T_{fail}$. According to the Proposition 3.16, $s \in \mathcal{S}(T_{fail})$. □

Proposition 3.23 tells that the special schemas of failing test case set are faulty schemas. As an example, let failing test cases set T_{fail} be $\{ \{(p_1,1), (p_2,0), (p_3,0), (p_4,0)\}, \{(p_1,1), (p_2,0), (p_3,0), (p_4,1)\} \}$. Table 14 listed all the faulty schemas, i.e., $\mathcal{S}(T_{fail})$.

Definition 3.24. The minimal set of schemas of the faulty schemas are called *minimal faulty schema (MFS).*

This definition is equal to the minimal failure-causing schema (MFS) which is first proposed in [12]. Other studies in CT focus on fault localization also aim to identifying this type of schemas

Table 14. An example of faulty schemas when given failing test case set

Failing test Case set T_{fail}	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$\mathcal{S}(T_{fail})$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$
s_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
s_3	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

[5, 12, 13, 22]. Figuring the MFS helps to identify the root cause of a failure and thus facilitate the debugging process.

Proposition 3.25 (Minimal failure-causing schemas and failing test set). *Given T_{fail} in the SUT, all the MFS are in the set $C(T_{fail})$.*

The proof of this proposition is obvious and hence omitted. This proposition tells that the MFS of the SUT are the minimal special schemas of the failing test case set. As an example, consider the same failing test cases used in the previous example. Table 15 listed all the MFS, i.e., $C(T_{fail})$.

Table 15. An example of MFS when given failing test case set

Failing test Case set T_{fail}	
t_1	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$
t_2	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$
$C(T_{fail})$	
s_1	$\{(p_1, 1), (p_2, 0), (p_3, 0)\}$

4 THE DETERMINABILITY OF SCHEMAS

In this section, we will introduce the notion of the determinability of schemas and its properties. Based on this, we give the formal definition of the pending schema.

4.1 The determinable schemas

Note that in Section 3, we introduced the definition of faulty schema and healthy schema, as well as some properties of them. Based on these definitions, we can conclude that in order to determine a schema to be healthy or faulty, we need to make sure whether the corresponding schema satisfies the specific conditions. For example, if we find the condition that one test case of the whole set of containers of a schema holds, then this schema is a healthy schema. Hence, if these conditions hold by given information or can be deduced from given information, we can determine the the schema to be healthy or faulty. Otherwise, we cannot determine the state of the schema for lack of information. Next, we will give the definition of the determinable schema.

Definition 4.1. A schema is determinable if and only if it is determined to be healthy schema or faulty schema according to given information.

Here, the "information" indicates the outcomes of the execution of some test cases or the determination state of existed schemas. If we are given the outcomes of all the test cases in the SUT or the determination results of all the possible schemas in the SUT, then for any schema in the SUT, we can easily determine it to be faulty or healthy according to Propositions 3.20, 3.21, 3.22

and 3.23. However, if only partial information is given, it is obvious that not all the schemas are determinable.

Next, we will discuss the determinability of schemas under a particular circumstance. Specifically, we are given a set of passing test cases T_{pass}, a set of failing test cases T_{fail}, and a set of test cases, i.e., $T_{unknown}$, of which the execution results are unknown. In this condition, we have $T_{pass} \bigcup T_{fail} \bigcup T_{unknown} = T^*$, where T^* indicates all the possible test cases in the SUT. Also, $T_{pass} \bigcap T_{fail} = T_{pass} \bigcap T_{unknown} = T_{fail} \bigcap T_{unknown} = \emptyset$. Note that, this condition is different from the condition we discussed in Section 3.2, where the test case is either belong to T_{fail} or T_{pass}. In this condition, a test case can belong to the third type of test case set $T_{unknown}$.

Proposition 4.2 (Determinable to be healthy). *Given T_{fail}, T_{pass}, $T_{unknown}$, the schemas that can be determined to be healthy are the set $\mathcal{I}(T_{pass})$.*

Proof. We first prove that, $\forall s \in \mathcal{I}(T_{pass})$, it must be a healthy schema. Obviously $\forall s \in \mathcal{I}(T_{pass})$, $\exists t \in T_{pass}, s \preceq t$. That is, $\exists t \in T_{pass}, t \in \mathcal{T}(s)$. According to the definition of the healthy schema, s is determined to be a healthy schema.

We second prove that, for any schema $s \notin \mathcal{I}(T_{pass})$, it cannot be determined to be a healthy schema. Since $s \notin \mathcal{I}(T_{pass})$, $\nexists t \in T_{pass}, s \preceq t$. Therefore, $\nexists t \in T_{pass}, t \in \mathcal{T}(s)$. Hence, $\mathcal{T}(s) \subseteq (T^* \backslash T_{pass}) = (T_{unknown} \bigcup T_{fail})$. Note that, there is none test case in $(T_{unknown} \bigcup T_{fail})$ is guaranteed to be passing test case. In fact, all the test cases in $T_{unknown}$ have the possibility to be failing test cases. Hence, schema s cannot be determined to be a healthy schema. □

Proposition 4.2 shows that only the schemas contained in those existed passing test cases can be determined to be healthy schemas. As an example, we consider the SUT with T_{pass}, T_{fail}, and $T_{unkonwn}$ shown as in Table 16. Then Table 17 shows the schemas that can be determined to be healthy schemas.

Table 16. An example of the state of the test cases in the SUT

Failing test Case set T_{fail}	
t_1	$\{(p_1,0), (p_2,0), (p_3,1), (p_4,0)\}$
t_2	$\{(p_1,0), (p_2,0), (p_3,1), (p_4,1)\}$
t_3	$\{(p_1,0), (p_2,1), (p_3,1), (p_4,0)\}$
t_4	$\{(p_1,0), (p_2,1), (p_3,1), (p_4,1)\}$
t_5	$\{(p_1,1), (p_2,0), (p_3,1), (p_4,0)\}$
t_6	$\{(p_1,1), (p_2,0), (p_3,1), (p_4,1)\}$
t_7	$\{(p_1,1), (p_2,1), (p_3,1), (p_4,0)\}$
t_8	$\{(p_1,1), (p_2,1), (p_3,1), (p_4,1)\}$
t_9	$\{(p_1,1), (p_2,1), (p_3,0), (p_4,0)\}$
t_{10}	$\{(p_1,1), (p_2,1), (p_3,0), (p_4,1)\}$
Passing Test Case Set T_{pass}	
t_{11}	$\{(p_1,0), (p_2,1), (p_3,0), (p_4,1)\}$
t_{12}	$\{(p_1,1), (p_2,0), (p_3,0), (p_4,1)\}$
Unknown Test Case Set $T_{unknown}$	
t_{13}	$\{(p_1,0), (p_2,0), (p_3,0), (p_4,0)\}$
t_{14}	$\{(p_1,0), (p_2,0), (p_3,0), (p_4,1)\}$
t_{15}	$\{(p_1,0), (p_2,1), (p_3,0), (p_4,0)\}$
t_{16}	$\{(p_1,1), (p_2,0), (p_3,0), (p_4,0)\}$

Table 17. An example of schemas determined to be healthy schemas

Determined to be healthy	$\mathcal{I}(T_{pass})$
s_1	$\{(p_1, 0)\}$
s_2	$\{(p_1, 1)\}$
s_3	$\{(p_2, 0)\}$
s_4	$\{(p_2, 1)\}$
s_5	$\{(p_3, 0)\}$
s_6	$\{(p_4, 1)\}$
s_7	$\{(p_1, 0),(p_2, 1)\}$
s_8	$\{(p_1, 0),(p_3, 0)\}$
s_9	$\{(p_1, 0),(p_4, 1)\}$
s_{10}	$\{(p_1, 1),(p_2, 0)\}$
s_{11}	$\{(p_1, 1),(p_3, 0)\}$
s_{12}	$\{(p_1, 1),(p_4, 1)\}$
s_{13}	$\{(p_2, 1),(p_3, 0)\}$
s_{14}	$\{(p_2, 1),(p_4, 1)\}$
s_{15}	$\{(p_2, 0),(p_3, 0)\}$
s_{16}	$\{(p_2, 0),(p_4, 1)\}$
s_{17}	$\{(p_3, 0),(p_4, 1)\}$
s_{18}	$\{(p_1, 0),(p_2, 1),(p_3, 0)\}$
s_{19}	$\{(p_1, 0),(p_2, 1),(p_4, 1)\}$
s_{20}	$\{(p_1, 0),(p_3, 0),(p_4, 1)\}$
s_{21}	$\{(p_1, 1),(p_2, 0),(p_3, 0)\}$
s_{22}	$\{(p_1, 1),(p_2, 0),(p_4, 1)\}$
s_{23}	$\{(p_1, 1),(p_3, 0),(p_4, 1)\}$
s_{24}	$\{(p_2, 1),(p_3, 0),(p_4, 1)\}$
s_{25}	$\{(p_2, 0),(p_3, 0),(p_4, 1)\}$
s_{26}	$\{(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 1)\}$
s_{27}	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 1)\}$

Proposition 4.3 (Determined to be faulty). *Given T_{fail}, T_{pass}, $T_{unknown}$,the schemas that can be determined to be faulty are the set $\mathcal{S}(T_{fail})$.*

Proof. We first prove that, $\forall s \in \mathcal{S}(T_{fail})$, it must be a faulty schema. Obviously $\forall s \in \mathcal{S}(T_{fail})$, $\mathcal{T}(s) \subseteq T_{fail}$ according to Proposition 3.13, indicating that s is a faulty schema according to the definition of faulty schema.

We second prove that, for any schema $s \notin \mathcal{S}(T_{fail})$, it cannot be determined to be a faulty schema. Since $s \notin \mathcal{S}(T_{fail})$, i.e., $s \notin \mathcal{I}(T_{fail}) \backslash \mathcal{I}(\overline{T_{fail}})$, we have $\exists t \in \overline{T_{fail}}, s \preceq t$. Therefore, $\exists t \in \overline{T_{fail}}, t \in \mathcal{T}(s)$. Note that $\overline{T_{fail}}) = T^* \backslash T_{fail} = (T_{unknown} \bigcup T_{pass})$. Since none test case in $T_{unknown} \bigcup T_{pass}$ can be guaranteed to be failing test case (all the test cases in $T_{unknown}$ have the possibility to be passing test cases), we have $\exists t \in \mathcal{T}(s)$ such that t cannot be guaranteed to failing test case. Hence, schema s cannot be determined to be a faulty schema. □

Proposition 4.3 shows that only special schemas of existed failing test cases can be determined to be faulty schemas. For the same example discussed previously, Table 18 lists all the schemas that are determined to be faulty schemas. We can observe that all the schemas in $\mathcal{S}(T_{fail})$ are faulty schemas. In other words, faulty schemas are the super-schemas of the schemas in $C(T_{fail})$ = { {(p_3, 1)}, {(p_1, 1), (p_2, 1)} }.

Table 18. An example of schemas determined to be faulty schemas

Determined to be faulty $\mathcal{S}(T_{fail})$	
s_{28}	$\{(p_3, 1)\}$
s_{29}	$\{(p_1, 1), (p_2, 1)\}$
s_{30}	$\{(p_1, 0), (p_3, 1)\}$
s_{31}	$\{(p_1, 1), (p_3, 1)\}$
s_{32}	$\{(p_2, 0), (p_3, 1)\}$
s_{33}	$\{(p_2, 1), (p_3, 1)\}$
s_{34}	$\{(p_3, 1), (p_4, 0)\}$
s_{35}	$\{(p_3, 1), (p_4, 1)\}$
s_{36}	$\{(p_1, 1), (p_2, 1), (p_3, 0)\}$
s_{37}	$\{(p_1, 1), (p_2, 1), (p_3, 1)\}$
s_{38}	$\{(p_1, 1), (p_2, 1), (p_4, 0)\}$
s_{39}	$\{(p_1, 1), (p_2, 1), (p_4, 1)\}$
s_{40}	$\{(p_1, 0), (p_2, 0), (p_3, 1)\}$
s_{41}	$\{(p_1, 0), (p_2, 1), (p_3, 1)\}$
s_{42}	$\{(p_1, 0), (p_3, 1), (p_4, 0)\}$
s_{43}	$\{(p_1, 0), (p_3, 1), (p_4, 1)\}$
s_{44}	$\{(p_1, 1), (p_2, 0), (p_3, 1)\}$
s_{45}	$\{(p_1, 1), (p_3, 1), (p_4, 0)\}$
s_{46}	$\{(p_1, 1), (p_3, 1), (p_4, 1)\}$
s_{47}	$\{(p_2, 0), (p_3, 1), (p_4, 0)\}$
s_{48}	$\{(p_2, 0), (p_3, 1), (p_4, 1)\}$
s_{49}	$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$
s_{50}	$\{(p_2, 1), (p_3, 1), (p_4, 1)\}$
s_{51}	$\{(p_1, 0), (p_2, 0), (p_3, 1), (p_4, 0)\}$
s_{52}	$\{(p_1, 0), (p_2, 0), (p_3, 1), (p_4, 1)\}$
s_{53}	$\{(p_1, 0), (p_2, 1), (p_3, 1), (p_4, 0)\}$
s_{54}	$\{(p_1, 0), (p_2, 1), (p_3, 1), (p_4, 1)\}$
s_{55}	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 0)\}$
s_{56}	$\{(p_1, 1), (p_2, 0), (p_3, 1), (p_4, 1)\}$
s_{57}	$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 0)\}$
s_{58}	$\{(p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1)\}$
s_{59}	$\{(p_1, 1), (p_2, 1), (p_3, 0), (p_4, 0)\}$
s_{60}	$\{(p_1, 1), (p_2, 1), (p_3, 0), (p_4, 1)\}$

PROPOSITION 4.4 (DETERMINABLE SCHEMAS). *Given T_{fail}, T_{pass}, $T_{unknown}$, only schemas $\mathcal{I}(T_{pass})$ and $\mathcal{S}(T_{fail})$ are determinable.*

This proposition gives the set of determinable schemas when given T_{fail}, T_{pass}, and $T_{unknown}$. The proof of this propositions can be easily obtained by Propositions 4.2 and 4.3, hence, we omit it here. For the same example discussed previously, Table 17 together with Table 18 list all the determinable schemas, which are $\mathcal{S}(T_{fail}) \bigcup \mathcal{I}(T_{pass})$.

Based on Proposition 4.4, we directly obtain the following formula to describe the determinable schemas set DSS, when given T_{fail}, T_{pass}, $T_{unknown}$.

$$DSS = \{s \mid s \in \mathcal{I}(T_{pass}) \mid\mid s \in \mathcal{S}(T_{fail})\}. \tag{1}$$

We can further evolve Formula 1 to the following formula.

$$DSS = \{s \mid \exists t \in T_{pass}, s \preceq t \mid\mid \exists m \in \mathcal{C}(T_{fail}), m \preceq s\}. \tag{2}$$

Note that Formula 2 is equal to the formula 1. The proof of the equality is omitted since it can be directly obtained by the definition of the whole set of hit schemas and the definition of minimal special schema.

4.2 The pending schemas

After the introduction of determinable schemas and their properties, we can easily define the indeterminable schemas, i.e., the pending schemas.

Definition 4.5. A schema is called a ***pending* schema**, if and only if it cannot be determined to be healthy schema or faulty schema according to given information.

From this definition, it is obvious that the pending schema is the converse of determinable schema. That is, with given information, if a schema is determinable, then it is not a pending schema; and if it is not determinable, it must be a pending schema. The *pending* schema is actually the *cannot-be-determined* schema discussed in Section 1 and Section 2, which is the key schema we focus on in this paper. To analyse the pending schemas in one failing test case is important to evaluate the completeness of the test. Next, we will give the first property about the pending schema.

Proposition 4.6 (Sandwich rule). *Given two pending schemas s_1, s_2, and $s_1 \preceq s_2$. Then $\forall s_3, s_1 \preceq s_3 \preceq s_2$, s_3 is a pending schema.*

Proof. We give the proof by contradiction. First, we assume that s_3 is a faulty schema. Since $s_3 \preceq s_2$, then according to Proposition 3.21, s_2 is also a faulty schema. This is contradiction as s_2 is a pending schema. Hence, s_3 is not a faulty schema. Next, we assume that s_3 is a healthy schema. Since $s_1 \preceq s_3$, then according to Proposition 3.20, s_1 is also a healthy schema. This is also contradiction as s_1 is a pending schema. Hence, s_3 is not a healthy schema. At last, according to the definition of pending schema, s_3 is a pending schema. □

This proposition gives the determination of pending schemas in the presence of two subsuming pending schemas. As an example, consider schemas $\{(p_1, 0)\}$, and $\{(p_1, 0), (p_2, 0), (p_3, 0)\}$ are two pending schemas, then schema $\{(p_1, 0), (p_2, 0)\}$ must also be pending schema.

Similarly as before, we will discuss the pending schemas under the circumstance when given a set of passing test cases T_{pass}, a set of failing test cases T_{fail}, and a set of test cases of which the execution results are unknown $T_{unknown}$.

Proposition 4.7 (Pending schemas). *Given T_{pass}, T_{fail}, and $T_{unknown}$, for any schema, say, s, is pending schema if only if $s \notin \mathcal{I}(T_{pass})$ and $s \notin \mathcal{S}(T_{fail})$.*

The proof of this proposition is omitted (can be easily deduced based on Proposition 4.4). Proposition 4.7 gives the set of pending schemas when given T_{fail}, T_{pass}, and $T_{unknown}$. As an example, we consider the same T_{fail}, T_{pass}, and $T_{unknown}$ as shown in Table 16. The pending schemas for this example are listed in Table 19. We can learn that each schema in Table 19 is neither contained in set $\mathcal{I}(T_{pass})$ of Table 17 nor contained in set $\mathcal{S}(T_{fail})$ of Table 18. Another observation from this Table is that all the pending schemas plus the determinable schemas are equal to whole set of schemas that contained in the SUT, of which the number is $3^4 - 1 = 80$.

Table 19. An example of pending schemas

$s \notin \mathcal{S}(T_{fail})$ **and** $s \notin \mathcal{I}(T_{pass})$	
s_{61}	$\{(p_4, 0)\}$
s_{62}	$\{(p_1, 0), (p_2, 0)\}$
s_{63}	$\{(p_1, 0), (p_4, 0)\}$
s_{64}	$\{(p_1, 1), (p_4, 0)\}$
s_{65}	$\{(p_2, 0), (p_4, 0)\}$
s_{66}	$\{(p_2, 1), (p_4, 0)\}$
s_{67}	$\{(p_3, 0), (p_4, 0)\}$
s_{68}	$\{(p_1, 0), (p_2, 0), (p_3, 0)\}$
s_{69}	$\{(p_1, 0), (p_2, 0), (p_4, 0)\}$
s_{70}	$\{(p_1, 0), (p_2, 0), (p_4, 1)\}$
s_{71}	$\{(p_1, 0), (p_2, 1), (p_4, 0)\}$
s_{72}	$\{(p_1, 0), (p_3, 0), (p_4, 0)\}$
s_{73}	$\{(p_1, 1), (p_2, 0), (p_4, 0)\}$
s_{74}	$\{(p_1, 1), (p_3, 0), (p_4, 0)\}$
s_{75}	$\{(p_2, 0), (p_3, 0), (p_4, 0)\}$
s_{76}	$\{(p_2, 1), (p_3, 0), (p_4, 0)\}$
s_{77}	$\{(p_1, 0), (p_2, 0), (p_3, 0), (p_4, 0)\}$
s_{78}	$\{(p_1, 0), (p_2, 0), (p_3, 0), (p_4, 1)\}$
s_{79}	$\{(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 0)\}$
s_{80}	$\{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0)\}$

More formally, the set of the pending schemas set PSS can be formulated as the following formula.

$$PSS = \{s \mid s \notin \mathcal{I}(T_{pass}) \ \&\& \ s \notin \mathcal{S}(T_{fail})\}. \tag{3}$$

Similarly, we can further evolve Formula 3 to the following formula.

$$PSS = \{s \mid \forall t \in T_{pass}, s \npreceq t \ \&\& \ \forall m \in C(T_{fail}), m \npreceq s\}. \tag{4}$$

It is obviously PSS and DSS are complemented.

5 OBTAINING THE PENDING SCHEMAS IN ONE FAILING TEST CASE

In the previous section, we have already obtained the formula to describe the set of pending schemas for given information. However, this set is usually too large and contains many schemas that never appear in any executed test cases. As an example, in Table 19, the schema s_{29} $\{(p_1, 1), (p_3, 1)\}$ neither appears in the T_{pass} nor appears in T_{fail}. These schemas are evidently pending schemas since they do not even have the chance to be tested. In this paper, we only focus on these schemas that have already appeared in the executed test cases. This is because after testing and MFS identification, testers may have a false confidence on the fact that all the schemas appeared in these test cases are determinable, which may result in hidden dangers to the system under testing if the fact is not true (some of them are not determinable). For example in Table 19, the schema s_{24} $\{(p_1, 0), (p_3, 0)\}$ is a pending schema, but it has already appeared in the executed failing test case t_1.

More formally, given a failing test case t =$\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, a set of passing test cases T_{pass}, a set of failing test cases T_{fail}, our goal is to find the set of pending schemas in test case t.

We will settle this problem step by step. According to Formula 4, we can easily obtain the pending schemas in the failing test case are in the following set.

$$PSS = \{s \mid s \preceq t \ \&\& \ \forall t \in T_{pass}, s \npreceq t \ \&\& \ \forall m \in C(T_{fail}), m \npreceq s\}. \tag{5}$$

The proof of this equation is obvious and omitted here. In fact, Formula 5 is obtained by selecting those pending schemas (based on Formula 4) that are in test case t. Since Formula 5 focus on these schemas that are in test case t, it reduces a lot of schemas that are needed to be considered when comparing with Formula 4. As an example, we still use the same T_{fail}, T_{pass}, and $T_{unknown}$ as shown in Table 16. Different from Table 19 which lists all the pending schemas, we only focus on the pending schemas in one failing test case, say, t_1 $\{(p_1, 0), (p_2, 0), (p_3, 1), (p_4, 0)\}$. Table 20 lists all the pending schemas in this test case. We can observe that there are only 5 schemas are selected from all the pending schemas listed in Table 19.

Table 20. An example of pending schemas in one test case

The test Case set	
t_1	$\{(p_1, 0), (p_2, 0), (p_3, 1), (p_4, 0)\}$
pending schemas in this test case	
s_{61}	$\{(p_4, 0)\}$
s_{62}	$\{(p_1, 0),(p_2, 0)\}$
s_{63}	$\{(p_1, 0),(p_4, 0)\}$
s_{65}	$\{(p_2, 0),(p_4, 0)\}$
s_{69}	$\{(p_1, 0),(p_2, 0),(p_4, 0)\}$

Although focusing on the pending schemas in executed test cases reduce a lot of computation, however, to directly utilize Formula 5 is still not practical, especially when the number of factors in one test case is large. This is because in the worst case it needs to check every schema in a test case t. Note that the number of schemas in t is $2^n - 1$, where n is the number of factors in this test case. For each schema we need to check whether it is the sub-schemas of any test case in T_{pass} (the number of the test cases in T_{pass} is $|T_{pass}|$). We also need to check whether it is the super-schemas of any schema in $C(T_{fail})$ (the number of the faulty schemas in $C(T_{fail})$ is $|C(T_{fail})|$). As a result, the complexity of directly utilizing Formula 5 is $2^n \times |T_{pass}| \times |C(T_{fail})|$, which is infeasible to handle when n is large. Hence, we need to find another formula which is equivalent to Formula 5, but with much lower complexity.

5.1 Reduce the space of schemas to be checked

Our first step is to reduce the number of minimal faulty schemas to be checked for a pending schema. For this, let $C(T_{fail})^t$ be the minimal faulty schemas that are the subschemas of test case t. That is, $C(T_{fail})^t = \{s \mid s \in C(T_{fail}), s \preceq t\}$. It is obvious that the complexity of obtaining $C(T_{fail})^t$ from $C(T_{fail})$ is $O(|C(T_{fail})|)$.

Proposition 5.1 (Reduce to fault schemas in t). *Let schema set* $A = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail}), m \npreceq s\}$, *schema set* $B = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^t, m \npreceq s\}$. *Then we have* $A = B$.

Proof. $A = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail}), m \npreceq s\} = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^t, m \npreceq s \&\& \ \forall m \in C(T_{fail}) \backslash C(T_{fail})^t, m \npreceq s\}$

Since $C(T_{fail}) \backslash C(T_{fail})^t = \{m \mid m \in C(T_{fail}), m \npreceq t\}$.

Hence, $\forall s \preceq t, m \in C(T_{fail}) \backslash C(T_{fail})^t, m \npreceq s$ (Otherwise, $m \preceq s \preceq t$, which is contradiction).

Since the condition $\forall s \preceq t, \forall m \in C(T_{fail}) \backslash C(T_{fail})^t, m \npreceq s$ does always hold, therefore, A = $\{s \mid s \preceq t \;\&\&\; \forall m \in C(T_{fail})^t, m \npreceq s \&\& \; \forall m \in C(T_{fail}) \backslash C(T_{fail})^t, m \npreceq s\} = \{s \mid s \preceq t \;\&\&\; \forall m \in C(T_{fail})^t, m \npreceq s\}$ = B. □

With this proposition, we only need to focus on the minimal faulty schemas that only contained in the failing test case t instead of all the possible minimal faulty schemas to obtain the pending schemas.

For the same example used in the previous section, we consider the SUT with T_{pass}, T_{fail}, and $T_{unkonwn}$ shown as in Table 16. In this example, we have $C(T_{fail}) = \{ \{(p_3, 1)\}, \{(p_1, 1), (p_2, 1)\} \}$. We next focus on the failing test case $t_1 = \{ (p_1, 0), (p_2, 0), (p_3, 1), (p_4, 0)\}$. For this test case, we have $C(T_{fail})^{t_1} = \{ \{(p_3, 1)\} \}$.

Table 21. An example of the reduction the faulty schemas needs to be checed

$\{s \mid s \preceq t \;\&\&\; \forall m \in C(T_{fail}), m \npreceq s\}$	
s_1	$\{(p_1, 0)\}$
s_3	$\{(p_2, 0)\}$
s_{61}	$\{(p_4, 0)\}$
s_{62}	$\{(p_1, 0),(p_2, 0)\}$
s_{63}	$\{(p_1, 0),(p_4, 0)\}$
s_{65}	$\{(p_2, 0),(p_4, 0)\}$
s_{69}	$\{(p_1, 0),(p_2, 0),(p_4, 0)\}$

$\{s \mid s \preceq t \;\&\&\; \forall m \in C(T_{fail})^t, m \npreceq s\}$	
s_1	$\{(p_1, 0)\}$
s_3	$\{(p_2, 0)\}$
s_{61}	$\{(p_4, 0)\}$
s_{62}	$\{(p_1, 0),(p_2, 0)\}$
s_{63}	$\{(p_1, 0),(p_4, 0)\}$
s_{65}	$\{(p_2, 0),(p_4, 0)\}$
s_{69}	$\{(p_1, 0),(p_2, 0),(p_4, 0)\}$

Next, we consider to reduce the number of passing test cases to be checked for a pending schema.

Let T_{pass}^t be the schemas that are intersection subschemas between each test case in T_{pass} and test case t. That is, $T_{pass}^t = \{s \mid \exists t' \in T_{pass}, s = t' \cap t\}$. Note that to obtain T_{pass}^t from T_{pass}, the complexity is $O(|T_{pass}|)$. It is easy to know that $\forall s \in T_{pass}^t, s \preceq t \;\&\&\; \exists s' \in T_{pass}, s \preceq s'$.

Proposition 5.2. *Let schema set* $A = \{s \mid s \preceq t \;\&\&\; \forall t_i \in T_{pass}, s \npreceq t_i\}$, *schema set* $B = \{s \mid s \preceq t \;\&\&\; \forall p \in T_{pass}^t, s \npreceq p\}$. *Then we have* $A = B$.

Proof. Note that $T_{pass}^t = \{s \mid \exists t' \in T_{pass}, s = t' \cap t\}$.

We first prove that $A \subseteq B$. Note for each $s \in A$, we have $s \preceq t \&\& \forall t_i \in T_{pass}, s \npreceq t_i$. We assume that $\exists s \in A, \exists s' \in T_{pass}^t, s \preceq s'$. For s', $\exists t' \in T_{pass}, s' \preceq t'$. Hence, $s \preceq s' \preceq t'$. This is contradiction. Hence the assumption does not hold, indicating that $\forall s \in A, \forall s' \in T_{pass}^t, s \npreceq s' \&\& s \preceq t$. That is $\forall s \in A$, $s \in B$. Hence, $A \subseteq B$.

Next we prove that $B \subseteq A$. Since $\forall s \in B$, we have $s \preceq t \&\& \forall t_i \in T_{pass}^t, s \npreceq t_i$. We assume that $\exists s \in B, \exists t' \in T_{pass}, s \preceq t'$. Also $s \preceq t$, which indicates that $s \preceq (t \cap t')$. Since $(t \cap t') \in T_{pass}^t$, $\exists s \in B, \exists t_i \in T_{pass}^t, s \preceq t_i$. This is contradiction, indicating that $\forall s \in B, \forall t' \in T_{pass}, s \npreceq t' \&\& s \preceq t$. That is, $\forall s \in B$, $s \in A$. Hence, $B \subseteq A$.

Since we have $A \subseteq B$ and $B \subseteq A$, we have $A = B$. □

With this proposition, we do not need to check each passing test case. Instead, we only need to focus on the intersection part of passing test cases and the failing test case.

We can further reduce the number of schemas need to checked. Let $T_{pass}^{t\triangle}$ be the maximal schemas in T_{pass}^t. That is, $T_{pass}^{t\triangle} = \{s \mid s \in T_{pass}^t \&\& \nexists s' \in T_{pass}^t, s \prec s'\}$. To obtain $T_{pass}^{t\triangle}$ from T_{pass} with complexity of $O(|T_{pass}|^2)$. It is easy to know that $\forall s \in T_{pass}^t, \exists s' \in T_{pass}^{t\triangle}, s \preceq s'$.

Proposition 5.3. *Let schema set* $A = \{s \mid s \preceq t \;\&\&\; \forall p \in T_{pass}^t, s \npreceq p\}$, *schema set* $B = \{s \mid s \preceq t \;\&\&\; \forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$. *Then we have* $A = B$.

Proof. First, we will prove $A \subseteq B$.

Since $T_{pass}^{t\triangle} \subseteq T_{pass}^{t}$, we have $\{s|\forall p \in T_{pass}^{t}, s \npreceq p\} = \{s|\forall p \in T_{pass}^{t\triangle}, s \npreceq p \ \&\& \ \forall p \in T_{pass}^{t} \backslash T_{pass}^{t\triangle}, s \npreceq p\} \subseteq \{s|\forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$. Hence, $\{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t}, s \npreceq p\} \subseteq \{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$. That is, $A \subseteq B$.

Next, we will prove $B \subseteq A$.

We assume that $\exists s \in \{s|\forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$, $s \notin \{s|\forall p \in T_{pass}^{t}, s \npreceq p\}$. It indicates that $\exists p' \in T_{pass}^{t}, s \preceq p'$. Since $p' \in T_{pass}^{t}$, $\exists p'' \in T_{pass}^{t\triangle}, p' \preceq p''$. It has $s \preceq p' \preceq p''$. This is contradiction. Hence, the assumption does not hold, indicating that $\forall s \in \{s|\forall p \in T_{pass}^{t\triangle}, s \in \{s|\forall p \in T_{pass}^{t}, s \npreceq p\}$. Hence, $\{s|\forall p \in T_{pass}^{t\triangle} \subseteq \{s|\forall p \in T_{pass}^{t}, s \npreceq p\}$. Therefore, $\{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\} \subseteq \{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t}, s \npreceq p\}$. That is $B \subseteq A$.

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

With this proposition, we do not need to check each schema in T_{pass}^{t}, instead, we only need to focus on the maximal schemas in this set.

With these two set of schemas, we can obtain the second formula to compute the pending schemas in a failing test case t, which is listed in the following.

$$PSS = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^{t}, m \npreceq s \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\}. \tag{6}$$

Proof. Let A = $\{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail}), m \npreceq s\}$, B = $\{s \mid s \preceq t \ \&\&\forall p \in T_{pass}, s \npreceq p\}$. Obviously, PSS = $A \cap B$.

According to Proposition 5.1, A = $\{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^{t}, m \npreceq s\}$. According to Proposition 5.2 and Proposition 5.3, B = $\{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t}, s \npreceq p\} = \{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$.

Hence, $PSS = A \cap B = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^{t}, m \npreceq s\} \cap \{s \mid s \preceq t \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\} = \{s \mid s \preceq t \ \&\& \ \forall m \in C(T_{fail})^{t}, m \npreceq s \ \&\& \ \forall p \in T_{pass}^{t\triangle}, s \npreceq p\}$. □

With respect to Formula 6, the complexity of obtaining the pending schemas is computed as follows. We first need to obtain $C(T_{fail})^{t}$ and $T_{pass}^{t\triangle}$, for which the complexities are $O(|C(T_{fail})|)$ and $O(|T_{pass}|^2)$. Then we still need to check each schema in test case t, to figure whether it is subschema of some schema in $T_{pass}^{t\triangle}$ or super-schema of some schema in $C(T_{fail})^{t}$, for which the complexity is $O(2^n * |C(T_{fail})^{t}| * |T_{pass}^{t\triangle}|)$, where n is the number of factors in the test case. At last, the complexity is $O(|C(T_{fail})| + |T_{pass}|^2 + 2^n * |C(T_{fail})^{t}| * |T_{pass}^{t\triangle}|)$. Obviously, this is still not feasible when n is large.

5.2 Change to CMXS and CMNS

Next, we will try to eliminate the influence of the number of n to obtain the pending schemas. For this purpose, we defined the following two sets, i.e. $CMXS$ and $CMNS$.

Definition 5.4. For a k-degree schema $s = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), ..., (p_{x_k}, v_{x_k})\}$, a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and $s \preceq t$. We denote the candidate maximal non-super schema set as $CMXS(s, t) = \{t \backslash (p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in s\}$.

Note that $CMXS$ is the set of schemas that remove one distinct factor value in s, such that all these schemas will not be the super-schema of s. For example assume the test case $\{(p_1, v_1), (p_2, v_2), (p_3, v_3), (p_4, v_4)\}$, and a 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 $CMXS$ with respect to one schema and one test case 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 schema, we can get the following proposition:

Proposition 5.5 (No to be super schema of one schema). *Given a schema s_1, a test case t, where $s_1 \preceq t$, we have $\{s \mid s \preceq t \ \&\& \ s_1 \npreceq s\} = \{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1\}$.*

Proof. Let A = $\{s \mid s \preceq t \ \&\& \ s_1 \npreceq s\}$, and B = $\{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1\}$.

Let $s_1 = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), ..., (p_{x_k}, v_{x_k})\}$, t = $\{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and CMXS($s_1$,$t$)= $\{t \backslash (p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in s_1\}$.

First we will show A ⊆ B.

With respect to set A, $\forall s' \in A$, it has $s' \preceq t$ and $s_1 \npreceq s'$. That is, $\forall e \in s', e \in t$, and $\exists e' \in s_1, e' \notin s'$. As $s_1 \preceq t$, $e' \in t$. Hence, we have $\forall e \in s', e \in t \backslash e'$, i.e., $s' \preceq t \backslash e'$.

Since $t \backslash e' \in CMXS(s_1, t)$, $s' \in \{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1\}$ = B. Hence, A ⊆ B.

Second we will show B ⊆ A.

With respect to set B, $\forall s' \in B$, it has $\exists s'_1 \in CMXS(s_1, t), s' \preceq s'_1$. Since $s'_1 \in CMXS(s_1, t)$, $\exists e' \in s_1, s'_1 = t \backslash e'$. Consequently, $s' \preceq t \backslash e'$. Hence, $s_1 \npreceq s'$. Also, $s' \preceq t \backslash e' \prec t$. Consequently, $s \in \{s \mid s \prec t \ \&\& \ s_1 \npreceq s\}$ = A, which indicates that B ⊆ A.

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

□

In the equation of Proposition 5.5, the schemas of the left side, i.e., $\{s \mid s \preceq t \ \&\& \ s_1 \npreceq s\}$, are the sub-schemas of test case t, but not the super-schemas of schema s_1. The right side set in this equation, i.e., $\{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1\}$, are schemas which are sub-schemas of at least one schema in $CMXS(s_1, t)$. Proposition 5.5 indicates that these two schema sets are equivalent. As an example, considering a test case t – $\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$, and a schema s_f – $\{(p_3, 1)\}$. Table 22 shows the schema set $\{s \mid s \prec t \ \&\& \ s_f \npreceq s\}$, $CMXS(s_f, t)$ and $\{s \mid \exists s'_1 \in CMXS(s_f, t), s \preceq s'_1\}$.

Table 22. An example of Proposition 5.5

Test case t	**$\{s \mid s \prec t \ \&\& \ s_f \npreceq s\}$**	**$CMXS(s_f, t)$**	**$\{s \mid \exists s'_1 \in CMXS(s_f, t), s \preceq s'_1\}$**
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_2, 1),(p_4, 1)\}$
Schema s_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 schemas. For this, we need the following notation: For two schemas s_1, s_2, and a test case t ($s_1 \preceq t$ and $s_2 \preceq t$), let $CMXS(s_1, t) \bigwedge CMXS(s_2, t) = \{s \mid s = s'_1 \cap s'_2,\ where\ s'_1 \in CMXS(s_1, t),\ and\ s'_2 \in CMXS(s_2, t)\}$.

For example, let $t = \{(p_1, v_1), (p_2, v_2), (p_3, v_3)\}$, $s_1 = \{(p_1, v_1), (p_2, v_2)\}$, $c_2 = \{(p_2, v_2), (p_3, v_3)\}$. Then we have $CMXS\ (s_1, t) = \{\ \{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2), (p_3, v_3)\}\ \}$, $CMXS(s_2,\ t) = \{\ \{(p_1, v_1), (p_2, v_2)\}, \{(p_1, v_1), (p_3, v_3)\}\ \}$, and $CMXS(s_1,\ t) \bigwedge CMXS(s_2, t) = \{\{(p_1, v_1)\}, \{(p_1, v_1), (p_3, v_3)\}, \{(p_2, v_2)\}, \{(p_3, v_3)\}\}$. It is easy to know the complexity of obtaining $CMXS$ of two schemas is O(τ^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 schemas.

Definition 5.6. Given a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of schemas $FSS = \{s_1, s_2, ...s_i, ...\}$, where $s_i \preceq t$, we denote the candidate maximal non-super schemas of this set as $CMXS(FSS, t)$= $\bigwedge_{s_i \in FSS} CMXS(s_i, t)$.

Note to compute the $CMXS$ of a set of 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 schemas is O($\tau^{|FSS|}$), where $|FSS|$ is the number of schemas in the schema set, and τ is the degree of the schema. According to Proposition 5.5, we have:

Proposition 5.7 (No to be super schema of a set of schemas). *Given a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of schemas $FSS = \{s_1, s_2, \ldots s_i, \ldots\}$, where $s_i \preceq t$, we have $\{s \mid s \preceq t \ \&\& \ \forall s_i \in FSS, s_i \npreceq s\} = \{s \mid \exists s'_1 \in CMXS(FSS, t), s \preceq s'_1\}$.*

Proof. We just need to prove that for two schemas s_1, s_2, and a test case t ($s_1 \preceq t$, $s_2 \preceq t$), we have $\{s \mid s \prec t \ \&\& \ \forall s_i \in \{s_1, s_2\}, s_i \npreceq s\} = \{s \mid \exists s'_1 \in CMXS(s_1, t) \bigwedge CMXS(s_2, t), s \preceq s'_1\}$.

Let $A = \{s \mid s \prec t \ \&\& \ \forall s_i \in \{s_1, s_2\}, s_i \npreceq s\}$, $A_1 = \{s \mid s \prec t \ \&\& \ s_1 \npreceq s\}$, $A_2 = \{s \mid s \prec t \ \&\& \ s_2 \npreceq s\}$. It is easily to get $A = A_1 \bigcap A_2$.

Let $B = \{s \mid \exists s'_1 \in CMXS(s_1, t) \bigwedge CMXS(s_2, t), s \preceq s'_1\}$. Here, $CMXS(s_1, t) \bigwedge CMXS(s_2, t) = \{s \mid s = s'_1 \cap s'_2, \ where \ s'_1 \in CMXS(s_1, t), \ and \ s'_2 \in CMXS(s_2, t)\}$.

Let $B_1 = \{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1\}$, and $B_2 = \{s \mid \exists s'_2 \in CMXS(s_2, t), s \preceq s'_2\}$. $B_1 \bigcap B_2 = \{s \mid \exists s'_1 \in CMXS(s_1, t), s \preceq s'_1 \ \&\& \ \exists s'_2 \in CMXS(s_2, t), s \preceq s'_2\}$. Note that, $s \preceq s'_1 \ \&\& \ s \preceq s'_2 \equiv s \preceq s'_1 \cap s'_2$. Hence, $B_1 \bigcap B_2 = \{s \mid \exists s'_1, s'_2, \ s'_1 \in CMXS(s_1, t)$, and $s'_2 \in CMXS(s_2, t), s \preceq s'_1 \cap s'_2\}$ = B.

Based on Proposition 5.5, $A_1 = B_1$, $A_2 = B_2$. Consequently, $A = A_1 \bigcap A_2 = B_1 \bigcap B_2$ = B. □

Proposition 5.7 extends Proposition 5.5 from a single schema to a set of schemas.

As an example, considering a test case $t - \{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$, and a set of schemas $FSS - \{ \{(p_3, 1)\}, \{(p_1, 1), (p_2, 1)\} \}$. Table 23 shows the schema set $\{s \mid s \preceq t \ \&\& \ \forall s_i \in FSS, s_i \npreceq s\}$, $CMXS(FSS, t)$ and $\{s \mid \exists s'_1 \in CMXS(FSS, t), s \preceq s'_1\}$.

Table 23. An example of Proposition 5.7

Test case t	$\{\mathbf{s} \mid \mathbf{s} \preceq \mathbf{t} \ \&\& \ \forall \mathbf{s_i} \in FSS, \mathbf{s_i} \npreceq \mathbf{s}\}$	**CMXS(FSS, t)**	$\{\mathbf{s} \mid \exists \mathbf{s'_1} \in$ **CMXS(FSS, t)** $, \mathbf{s} \preceq \mathbf{s'_1}\}$
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$	$\{(p_1, 1),(p_4, 1)\}$
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 5.8. For a k-degree schema $s = \{(p_{x_1}, v_{x_1}), (p_{x_2}, v_{x_2}), ..., (p_{x_k}, v_{x_k})\}$, a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and $s \preceq t$. We denote the candidate minimal non-sub schema set as $CMNS(s, t)$=$\{\{(p_{x_i}, v_{x_i})\} \mid (p_{x_i}, v_{x_i}) \in t \backslash s\}$.

Note that $CMNS$ is the set of schemas that are assigned to one distinct factor value that is not in s, such that all these schemas will not be the sub-schema of s. For example assume the test case $\{(p_1, v_1), (p_2, v_2), (p_3, v_3), (p_4, v_4)\}$, and a 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 schema, we can get the following proposition:

Proposition 5.9 (Not to be subschema of one schema). *Given a schema s_1, a test case t, where $s_1 \preceq t$, we have $\{s \mid s \preceq t \ \&\& \ s \npreceq s_1\} = \{s \mid s \preceq t \ \&\& \ \exists s'_1 \in CMNS(s_1, t), s'_1 \preceq s\}$.*

Proof. Let A = $\{s \mid s \preceq t \ \&\& \ s \npreceq s_1\}$. B = $\{s \mid s \preceq t \ \&\& \ \exists s'_1 \in CMNS(s_1, t), s'_1 \preceq s\}$. $CMNS(s_1, t)$=$\{(p_{x_i}, v_{x_i}) \mid (p_{x_i}, v_{x_i}) \in t \backslash s_1\}$.

First we will show A $\subseteq$ B.

With respect to set A, $\forall s' \in A$, it has $s' \preceq t$ and $s' \npreceq s_1$. That is, $\forall e \in s', e \in t$, and $\exists e' \in s', e' \notin s_1$. Hence, $\{e'\} \preceq s'$, $e' \in t \backslash s_1$, which indicates that $s' \in \{s \mid s \preceq t \ \&\& \ \exists s'_1 \in CMNS(s_1, t), s'_1 \preceq s\}$ = B. So, A $\subseteq$ B.

Second we will show B ⊆ A.

With respect to set B, $\forall s' \in B$, it has $s' \preceq t$ and $\exists s_1' \in CMNS(s_1, t), s_1' \preceq s'$. As $s_1' \in CMNS(s_1, t)$, $\exists e' \in t \backslash s_1, s_1' = \{e'\}$. Hence, $\{e'\} \preceq s'$. Hence, $s' \npreceq s_1$. Consequently, $s' \in \{s \mid s \preceq t \ \&\& \ s \npreceq s_1\}$ = A, which indicates that B ⊆ A.

□

In the equation of Proposition 5.9, the schemas of the left side, i.e., $\{s \mid s \preceq t \ \&\& \ s \npreceq s_1\}$, are the sub-schemas of test case t, but not the sub-schemas of schema s_1. The right side set in this equation, i.e.,$\{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_1, t), \ s_1' \preceq s\}$, are sub-schemas of test case t, and also are the super-schemas of at least one schema in $CMNS(s_1, t)$. Proposition 5.9 indicates that these two schema sets are equivalent. As an example, considering a test case t – $\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$, and a schema c_h – $\{(p_1, 1),(p_2, 1),(p_3, 1)\}$. Table 24 shows the schema set $\{s \mid s \preceq t \ \&\& \ s_h \npreceq s\}$, $CMNS(s_h, t)$ and $s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_h, t), \ s_1' \preceq s\}$.

Table 24. An example of Proposition 5.9

Test case t	$\{s \mid s \preceq t \ \&\& \ s \npreceq s_h\}$	$CMNS(s_h, t)$	$\{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_h, t), s_1' \preceq s\}$
$\{(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)\}$	$\{(p_4, 1)\}$	$\{(p_4, 1)\}$	$\{((p_4, 1)\}$
Schema s_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 s_1, s_2, and a test case t ($s_1 \preceq t$, $s_2 \preceq t$), let $CMNS(s_1, t) \bigvee CMNS(s_2, t) = \{s \mid s = s_1' \cup s_2', \ where \ s_1' \in CMNS(s_1, t), \ and \ s_2' \in CMNS(s_2, t)\}$.

For example, let $t = \{(p_1, v_1), (p_2, v_2), (p_3, v_3)\}$, $s_1 = \{(p_1 , v_1), (p_2, v_2)\}$, $s_2 = \{(p_2, v_2), (p_3, v_3)\}$. Then we have $CMNS\ (s_1, t) = \{\ \{(p_3, v_3)\}\ \}$, $CMNS(s_2, t) = \{\ \{(p_1, v_1)\}\ \}$, and $CMNS(s_1, t) \bigvee CMNS(s_2, t) = \{\{(p_1, v_1), (p_3, v_3)\}\}$. Based on this, we denote $CMNS(HSS, t)$ for a set of faulty schemas.

Definition 5.10. Given a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of schemas $HSS = \{s_1, s_2, ...s_i, ...\}$, where $s_i \preceq t$, we denote the candidate minimal non-sub schema of this set as $CMNS(HSS, t) = \bigvee_{s_i \in HSS} CMNS(s_i, t)$.

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

Proposition 5.11 (Not to be subschema of a set of schemas). *Given a test case $t = \{(p_1, v_1), (p_2, v_2), ..., (p_n, v_n)\}$, and a set of schemas $HSS = \{s_1, s_2, ...s_i, ...\}$, where $s_i \preceq t$, we have $\{s \mid s \preceq t \ \&\& \ \forall s_i \in HSS, s \npreceq s_i\} = \{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(HSS, t), \ s_1' \preceq s\}$.*

Proof. We just need to prove that for two schemas s_1, s_2, and a test case t ($s_1 \preceq t$, $s_2 \preceq t$), we have $\{s \mid s \preceq t \ \&\& \ \forall s_i \in \{s_1, s_2\}, s \npreceq s_i\} = \{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_1, t) \bigvee CMNS(s_2, t), \ s_1' \preceq s\}$.

Let $A = \{s \mid s \preceq t \ \&\& \ \forall s_i \in \{s_1, s_2\}, s \npreceq s_i\}$, $A_1 = \{s \mid s \preceq t \ \&\& \ s \npreceq s_1\}$, $A_2 = \{s \mid s \preceq t \ \&\& \ s \npreceq s_2\}$. It is easily to get $A = A_1 \bigcap A_2$.

Let $B = \{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_1, t) \bigvee CMNS(s_2, t), s \preceq s_1'\}$. Here, $CMNS(s_1, t) \bigvee CMNS(s_2, t) = \{s \mid s = s_1' \cup s_2', \ where \ s_1' \in CMNS(s_1, t), \ and \ s_2' \in CMNS(c_2, t)\}$.

Let $B_1 = \{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_1, t), \ s_1' \preceq s\}$, and $B_2 = \{s \mid s \preceq t \ \&\& \ \exists s_2' \in CMNS(s_2, t), \ s_2' \preceq s\}$. $B_1 \bigcap B_2 = \{s \mid s \preceq t \ \&\& \ \exists s_1' \in CMNS(s_1, t), \ s_1' \preceq s \ \&\& \ \exists s_2' \in CMNS(s_2, t), \ s_2' \preceq s\}$. Note that, $s_1' \preceq s$ && $s_2' \preceq s \equiv s_1' \cup s_2' \preceq s$. Hence, $B_1 \bigcap B_2 = \{s \mid s \preceq t$ && $\exists s_1', s_2',\ s_1' \in CMNS(s_1, t)$, and $s_2' \in CMNS(s_2, t),\ s_1' \cup s_2' \preceq s\ \}$ = B.

Based on Proposition 5.9, $A_1 = B_1$, $A_2 = B_2$. Consequently, $A = A_1 \bigcap A_2 = B_1 \bigcap B_2$ = B.

□

Similar to Proposition 5.5 and 5.7, Proposition 5.11 extends Proposition 5.9 from a single schema to a set of schemas.

As an example, considering a test case t – {(p_1, 1),(p_2, 1),(p_3, 1),(p_4, 1)}, and a set of schemas HSS – { {(p_1, 1),(p_2, 1),(p_3, 1)}, {(p_3, 1),(p_4, 1)} }. Table 25 shows the schema set $\{s \mid s \preceq t$ && $\forall s_i \in HSS, s \npreceq s_i\}$, $CMNS(HSS, t)$ and $\{s \mid s \preceq t$ && $\exists s_1' \in CMNS(HSS, t),\ s_1' \preceq s\}$.

Table 25. An example of Proposition 5.11

Test case t	{s | s ⪯ t && ∀s_i ∈ HSS, s ⋠ s_i}	CMNS(HSS, t)	{s | s ⪯ t && ∃s'₁ ∈ CMNS(HSS, t), s'₁ ⪯ s}
{(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)}
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)}

Based on Proposition 5.7 and 5.11, we can easily learn that $\{s \mid s \preceq t$ && $\forall s_i \in FSS, s_i \npreceq s\} \bigcap \{s \mid s \preceq t$ && $\forall s_i \in HSS, s \npreceq s_i\} = \{s \mid \exists s_1' \in CMXS(FSS, t),\ s \preceq s_1'\} \bigcap \{s \mid s \preceq t$ && $\exists s_1' \in CMNS(HSS, t),\ s_1' \preceq s\}$. Considering $\forall s \in \{s \mid \exists s_1' \in CMXS(FSS, t),\ s \preceq s_1'\}$, $s \preceq t$, we have $\{s \mid \exists s_1' \in CMXS(FSS, t),\ s \preceq s_1'\} \bigcap \{s \mid s \preceq t$ && $\exists s_1' \in CMNS(HSS, t),\ s_1' \preceq s\} = \{s \mid \exists s_1' \in CMXS(FSS, t),\ s \preceq s_1'\} \bigcap \{s \mid \exists s_1' \in CMNS(HSS, t),\ s_1' \preceq s\}$.

Let t be the failing test case, the schemas set $T_{pass}^{t\triangle}$ be the set of schemas HSS (it satisfies that $\forall s \in T_{pass}^{t\triangle}, s \preceq t$), the minimal faulty schemas $C(T_{fail})^t$ be the set of schemas FSS (it also satisfies that $\forall s \in C(T_{fail})^t, s \preceq t$). Then we transform the aforementioned formula (Formula 5) to be the following equation.

$\{s \mid s \preceq t$ && $\forall t_i \in T_{pass}^{t\triangle}, s \npreceq t_i$&& $\forall m \in C(T_{fail})^t, m \npreceq s\} = \{s \mid \exists s_1' \in CMNS(T_{pass}^{t\triangle}, t),\ s_1' \preceq s$ && $\exists s_1' \in CMXS(C(T_{fail})^t, t),\ s \preceq s_1'\ \} = \{s \mid \exists s_1' \in CMNS(T_{pass}^{t\triangle}, t), s_2' \in CMXS(C(T_{fail})^t, t),\ s_1' \preceq s \preceq s_2'\ \}$.

At last, we can learn that the pending schemas set is equal to the following formula.

$$PSS = \{s \mid \exists s_1' \in CMNS(T_{pass}^{t\triangle}, t), s_2' \in CMXS(C(T_{fail})^t, t), s_1' \preceq s \preceq s_2'\}. \tag{7}$$

According to Formula 7, the complexity of obtaining one pending schema is $O(\tau^{|C(T_{fail})^t|} \times \tau^{|T_{pass}^{t\triangle}|})$. This is because to obtain one pending schema, we only need to search the schemas in $CMXS(C(T_{fail})^t, t)$ and $CMNS(T_{pass}^{t\triangle}, t)$, of which the complexity are $O(\tau^{|C(T_{fail})^t|})$ and $O(\tau^{|T_{pass}^{t\triangle}|})$, respectively. Then we need to check each pair of schemas in these two sets, to find whether exists $c_1 \in CMXS(C(T_{fail})^t, t)$, $c_2 \in CMNS(T_{pass}^{t\triangle}, t)$, such that $c_2 \preceq c_1$. If so, then both c_2 and c_1 satisfy Formula 7. Furthermore, $\forall c_3, c_2 \preceq c_3 \preceq c_1$, c_3 also satisfy Formula 7. Hence, the complexity of obtaining one pending schema is $O(\tau^{|C(T_{fail})^t|} \times \tau^{|T_{pass}^{t\triangle}|})$. Note that with Formula 7, we have eliminated the influence of the number n (See Formula 5 and Formula 6) to obtaining the pending schemas.

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

Failing Test Case t: { (p1, 1), (p2, 1), (p3, 1), (p4, 1)}
Faulty Schema Set (FSS): {(p1, 1), (p2, 1), (p3, 1)}, {(p1, 1), (p2, 1)}
Healthy Schema Set (HSS): {(p2, 1), (p3, 1), (p4, 1)}, {(p2, 1), (p3, 1)}
Minimal Faulty Schema Set (FSS$^{\perp}$): {(p1, 1), (p2, 1)}
Maximal Healthy Schema Set (HSS$^{\top}$): {(p2, 1), (p3, 1), (p4, 1)}

Faulty Schemas	Healthy Schemas	Pending Schemas
{(p1, 1), (p2, 1), (p3, 1), (p4, 1)} {(p1, 1), (p2, 1), (p3, 1)} {(p1, 1), (p2, 1), (p4, 1)} {(p1, 1), (p2, 1)}	{(p2, 1), (p3, 1), (p4, 1)} {(p2, 1), (p3, 1)} {(p2, 1), (p4, 1)} {(p3, 1), (p4, 1)} {(p2, 1)} {(p3, 1)} {(p4, 1)}	{(p1, 1), (p3, 1), (p4, 1)} {(p1, 1), (p4, 1)} {(p1, 1), (p3, 1)} {(p1, 1)}
CMXS(FSS, t)	**CMNS(HSS, t)**	**{c | ∃c1 ∈ CMXS(FSS,t), c2 ∈ CMNS(HSS,t), c2 ≼ c ≼ c1 }**
{(p2, 1), (p3, 1), (p4, 1)} {(p1, 1), (p3, 1), (p4, 1)} {(p3, 1), (p4, 1)} {(p2, 1), (p4, 1)} {(p1, 1), (p4, 1)}	{(p1, 1), (p4, 1)} {(p1, 1)}	{(p1, 1), (p3, 1), (p4, 1)} {(p1, 1), (p4, 1)} {(p1, 1), (p3, 1)} {(p1, 1)}
CMXS(FSS$^{\perp}$, t)	**CMNS(HSS$^{\top}$, t)**	**{c | ∃c1 ∈ CMXS(FSS$^{\perp}$,t), c2 ∈ CMNS(HSS$^{\top}$,t), c2 ≼ c ≼ c1 }**
{(p2, 1), (p3, 1), (p4, 1)} {(p1, 1), (p3, 1), (p4, 1)}	{(p1, 1)}	{(p1, 1), (p3, 1), (p4, 1)} {(p1, 1), (p4, 1)} {(p1, 1), (p3, 1)} {(p1, 1)}

Fig. 7. The example of Proposition

5.3 A case study of applying Formula

In this section, we will show a case study of how to obtain the pending schemas with Formula ??. For consistency, we use the example in the Section 2. Next, we will show how to apply Formula ?? to obtain the pending schemas of the failing test case t_5 after using FINOLP. Note that other approaches can be obtained in the same way. From Figure , we can learn that the maximal healthy schemas are the passing test cases themselves, which are t_1 ,..., and t_{16}. While the minimal faulty schemas are the MFS (Other faulty schemas are the super schemas of this one). Then we can learn that. Note that we can learn that .

6 EMPIRICAL STUDIES

6.1 The pending schemas for covering arrays

6.2 The existence of pending schemas for different MFS identification approaches

6.3 Does the exitentce of pending schemas contain some potential MFS which may be harmful

In fact, this is the motivation

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

6.5 The effectiveness of the approach

6.6 Threats to validity

7 DISCUSSION

8 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.

9 CONCLUSION

To identify the MFS in the SUT is important because it provides additional supports to find the cause of the failure. Although many efforts have been proposed to make the MFS identification approaches more efficient and effective, the existence of pending schemas still make them incomplete, which would be hidden dangers to the software under testing. Hence, to identify these pending schemas is of great importance to reveal such potential risks.

In this paper, we studied the characteristics of the relationships among faulty schemas, healthy schemas, minimal failure-causing schemas, and maximal healthy schemas. The subsuming relationships among them form the basis of the methodology for pending schemas. Particulary, we proposed several propositions to formulate the set of pending schemas and give three equivalent formulas to obtain them. Among the three formulas, the last formula reduce the complexity of obtaining pending schemas from O(2^n) to O($\tau^{|FSS^{\perp}|+|HSS^{\top}|}$), where n is the number of factors in the software, while $|FSS^{\perp}|$ and $|HSS^{\top}|$ are two relatively small numbers and independent on the number of n.

We conduct a series empirical studies on some real software systems with various number of parameters and values. Our results shows that the incompleteness in very common in the covering arrays and MFS identification approaches. We also observed that the third proposed formula is the most efficient when compared others in most cases.

As a future work, it is appealing to take the advantages of the pending schemas to assist in the MFS identification approaches. More specifically, we would like to develop a new framework for MFS identification, which take the pending schemas as a indicator. That is, any MFS identification algorithm that runs on this framework aims to empty the set of pending schemas. It is of interest to evaluate the efficiency and effectiveness of this framework with respect to various MFS identification approaches that run on it.

Another interesting futher work is to obtain a general theory of pending schemas, faulty schemas, and healthy schemas.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their hard work and valuable comments. This work was supported by the National Key Research and Development Plan(No. 2018YFB1003800), National Science Foundation Award CNS-1748109, US Department of Homeland Security(DHS-14-ST-062-001), and US National Institute of Standards and Technologies Award(70NANB15H199).

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