Identifying MFSs with considering masking effect*

[Extended Abstract] †

Xintao Niu
State Key Laboratory for Novel Software Technology
Nanjing University
China, 210093
niuxintao@smail.nju.edu.cn

Changhai Nie
State Key Laboratory for Novel Software Technology
Nanjing University
China, 210093
changhainie@nju.edu.cn

Alvin Chain
Department of computing
Hong Kong Polytechnic University
Hong Kong
cstschan@comp.polyu.edu.hk

ABSTRACT

Minimal failure-inducing schema(MFS) is a important concept in combinatorial testing that indicate the failure-inducing interaction of parameters in the software under test(SUT). Identify the MFS can help developers quickly reduce the search space needed to find the buggy source. Many algorithms are proposed to find these MFSs in the failing test cases. In practice, however, we find that these algorithms cannot behave as expected in the condition that SUT has multiple faults with different levels. For if so, a fault with higher level may be triggered and leaving the code which will trigger the fault with lower level not executed. Thus we cannot observe the fault with lower level, as a result, we will omit the MFS that related to the fault with lower level. We call this a masking effect.

In this paper, we propose a framework which can help the algorithms to avoid this masking effect when identify MFSs in the test cases. In this framework, we first static analysis the data flow of the test cases. Second we record fault as well as the code lines related to this fault during executing test cases. Then we will determine the levels of different faults we triggered during testing through finding the relationships of the code lines of each fault. By doing so we can judge whether a masking effect is happened during the process of identifying MFS and avoiding them by generating new test case that exposing the fault with lower level.We have applied our framework into several algorithms which focus on identifying the MFS in test cases and empirically studied the framework on two widely used open source software. Our result of the studies shows that our framework can effectively reduce the influence of masking effect.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

Minimal failure-inducing schemas, Masking effect

1. INTRODUCTION

With the increase of requirements for more features and customisable, modern software are designed to be configurable and modular. While it can make software portable and flexible, it also bring many challenges to the testers when testing them. The major one of these challenges is that we must make sure the components or options in the software coexistence with each other. As exhaustive testing each possible combination is not impractical when there are large amount of components or options in the SUT, we should choose some of them to test with considering the cost. There are many strategies for one to select the test configurations among all the possible configurations. Combinatorial testing is one of them that can reach the coverage of all the interactions of components with the number of component(we called interaction strength) not more than t.

Which criteria we should choose or how to generate these test cases is not the point in this paper, however, we will focus on the followed problem: if we find some test configurations failed during executing, which subset of the combination of component is the source of this failure? In another word, we want to identify the failure-inducing interactions of component rather than just detect them. Many works (as well as our previous work) are proposed to solve this problem. Most of these works focus on how to identify as more MFSs as possible while just generating small size of extra test configurations.

In our recent studies, however, we find these algorithms cannot behave as expected in some subject software. Through a deep analysis, we find that there are multiple faults with different levels in these subject. It means that when we set

*(Does NOT produce the permission block, copyright information nor page numbering). For use with ACM_PROC_ARTICLE-SP.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

up a test configuration and execute the SUT to observe the result, the high level fault will trigger first and perturb we examining the code that may trigger the low level fault. As a result we will omit some options or component in this test configuration that may be the cause of the low level fault. We call this a masking effect which make the MFS identifying algorithms not able to work properly.

In this paper, we propose a approach that can assist these algorithms to avoid these masking effect. Our framework consists of three parts: first, it will use the statistic analysis technique–dominate tree to analysis the test script and then collect the information of code lines in this script. Second, we will support a interface called "Record" for the MFS identifying algorithms that each time the algorithm encounter a fault should call this interface. So that we can record this fault as well as the code lines that trigger this fault. Last, this framework support these algorithms the interface "analysis" that can tell them whether the fault they encounter having masked some fault else.

To evaluate the effectiveness of our framework, we took two widely-used open source software as our experiment subject. And then we will choose five MFSs identifying algorithms, for each algorithm, we will compare the identifying result among two versions of this algorithm, one using our framework while another one not. The result of the empirical studies shows that our framework can assist the MFS identifying algorithm in getting a more accurate result.

The main contributions of this paper are:

1. We show that the fault corresponding to the MFS has different levels, i.e., fault with high level can mask the fault with low level.
2. We give a framework to assist algorithms to avoid the bad influence of the masking effect in different levels of fault.
3. We empirically studies that with considering the masking effect the algorithm can perform better than not considering the effect.

Rest of paper is organised as follows: section 2 gives a simple example to motivate our work. Section 3 describe our framework in detail. Section 4 illustrate the experiment and reports the result. Section 5 discusses the related works. Section 6 provides some concluding remarks.

2. MOTIVATION EXAMPLE

Following we have construct a example to illustrate the motivation of our approach. Assume we have a method *foo* which has four input parameters : *a, b, c, d.* The types of these four parameters are all integers and the values that they can take are: $d_a = \{7, 11\}, d_b = \{2, 5\}, d_c = \{2, 6\}, d_b = \{3, 6\} respectively. The detail code of this method is listed as following:$

```
public static float foo(int a, int b, int c, int d){
  //step 1 will cause a exception when b == c
  float x = (float)a / (b - c);
  //step 2 will cause a exception when c == d
  float y = x / (c - d);

  return x+y;
  }
```

From the code above, we can find two faults: first we can get a diversion exception when b is equal to c, that is b = 2 c = 2.

consider the a scenario: we want to read the integer numbers in a file and sort them in a descending order.

The following code contain two kinds of fault.

the allowed values for these are:

from this code we can obvious find two MFS: a = b = c = . while another is a = c= ,.

However, when we use traditional algorithms to apply in this example, we can find the result is not this t. Consider the followed test case, yes, it failed with, however, when we change a value of it to another , is it fails, but with different error message. So this time how can we . if else ...

while.

traditional identifying algorithms.

3. RELATED WORKS

combiantorial testing has may factors, Nie give a survey, at some are foucs,.

Identify MFS are

then consider the masking effect is , to my best knowledege, only one paper, unfortuntately, it just consider the .

Ylimaz propose a work that is feedback driven combinatorial testing, different from our work, it first using CTA classify the possible MFS and then elimate them and generate new test cases to detect possible masked interaction in the next iteration. The difference is that the main focus of that work is to generate test cases that didn't omit some schemas that may be masked by other schemas. And our work is main focus on identifying the MFS and avoiding the masking effect.

4. PRELIMINARY

SUT,

fault. level. given this definition, we can also take the constraint as a fault, usually it will have the highest level, for that if we take a combination which is constraint, it will may didn't compile at all, so that we can't exam any code in this software.

test configuration?

schema. faulty, healthy. we need to identify the minimal faulty schema, that is MFS.

note that in our paper, we just consider the failure that is deterministic.

5. APPROACH

We have use the static analysis tool: dominate tree to record the relationships among the code lines in the test script.

6. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller sections. The command `\section` that precedes this paragraph is part of such a hierarchy.[1] LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

6.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[2] typeface, but that is handled by the document class file. Take care with the use of[3] the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[?].

6.2 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

6.2.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[?]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

6.2.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

6.3 Citations

Citations to articles [?, ?, ?, ?], conference proceedings [?] or books [?, ?] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [?]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[?].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed.

6.4 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly

[1] This is the second footnote. It starts a series of three footnotes that add nothing informational, but just give an idea of how footnotes work and look. It is a wordy one, just so you see how a longish one plays out.

[2] A third footnote, here. Let's make this a rather short one to see how it looks.

[3] A fourth, and last, footnote.

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 1: A sample black and white graphic (.eps format).

in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

6.5 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption.

Figure 2: A sample black and white graphic (.eps format) that has been resized with the `epsfig` command.

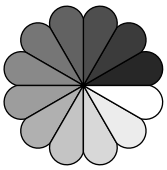

Figure 3: A sample black and white graphic (.ps format) that has been resized with the `psfig` command.

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

6.6 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

and don't forget to end the environment with figure*, not figure!

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

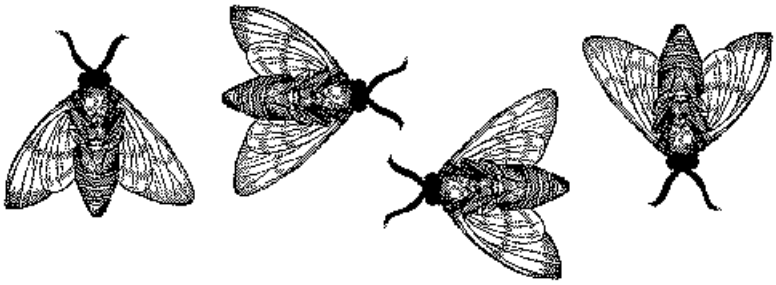

Figure 4: A sample black and white graphic (.eps format) that needs to span two columns of text.

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TEX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TEX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LATEX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

7. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LATEX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

8. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.