Thank you for downloading this document from the RMIT Research Repository.

The RMIT Research Repository is an open access database showcasing the research outputs of RMIT University researchers.


**Citation:**

See this record in the RMIT Research Repository at:

**Version:** Published Version

**Copyright Statement:** © 2009 IADIS

**Link to Published Version:**

PLEASE DO NOT REMOVE THIS PAGE
ON THE INTEGRATION OF METAMORPHIC TESTING
AND MODEL CHECKING

Huai Liu1, Daoming Wang2, Huimin Lin2, Tsong Yueh Chen1

1Centre for Software Analysis and Testing, Swinburne University of Technology, Australia
2Institute of Software, Chinese Academy of Sciences, China

ABSTRACT

Metamorphic testing, an innovative software testing technique, generates test cases based on domain specific properties. Model checking is a technique that verifies software designs against system properties. Motivated by the fact that both techniques are based on some properties of software under development, we investigate how to integrate metamorphic testing and model checking. This paper will introduce and discuss some potential topics on this new research direction.

1. INTRODUCTION

In many software testing techniques, a set of program inputs are selected as test cases according to some criteria. The software under test is then run against these test cases. Testers verify the program outputs (namely, the testing results) based on a mechanism called oracle. However, in practice, there may exist a problem called oracle problem where (i) “there does not exist an oracle” or (ii) “it is theoretically possible, but practically too difficult to determine the correct output” (Chen et al., 2003). Recently a new software testing technique, namely metamorphic testing (Chen et al., 1998; Gotlieb and Botella, 2003), has been proposed to alleviate the oracle problem. In metamorphic testing, some properties are first identified from the software under test. Then, metamorphic testing generates test cases based on these properties. In addition, the testing results are verified against these properties, not the oracle. Metamorphic testing technique has been widely used in various areas of software engineering (Chen et al., 2003, 2009a, 2009b; Murphy et al., 2008).

Model checking (Clarke et al., 1999) is an automatic technique to check if a finite state system satisfies its specifications which are expressed as temporal formulas. In model checking, a software design is checked against a set of properties specified by such formulas. When the design violates a property, a counterexample will be generated to point out the source of the error. This feature makes model checking practically useful.

Although metamorphic testing and model checking deal with different problems in different areas, they are both based on some properties of the system under test. In this paper, we are motivated to look at whether these two innovative techniques can be integrated, aiming at improving the effectiveness of software verification and testing. This paper is organized as follows. In Sections 2 and 3, we briefly introduce metamorphic testing and model checking, respectively. In Section 4, we propose some potential research topics about the integration of metamorphic testing and model checking. In Section 5, we conclude this paper.

2. METAMORPHIC TESTING

In metamorphic testing, the program developers, testers, and/or end-users first identify some properties of the software under test. Some relations called metamorphic relations can then be constructed based on these properties. Source test cases are generated by some traditional test case selection techniques. Follow-up test cases are then constructed based on source test cases and metamorphic relations. The software under test is executed with all test cases as the program inputs. The outputs of the source and follow-up test cases are compared according to the metamorphic relations. If testing results do not satisfy a metamorphic relation, a failure is said to be detected. The basic process of metamorphic testing is shown in Figure 1.

* Corresponding author.
Since the proposal of metamorphic testing, it has been successfully applied to alleviate the oracle problem in testing various software products. For example, Murphy et al. (2008) have proposed some metamorphic relations that can be used for testing machine learning applications. Chen et al. (2009) detected a real-life bug in a bioinformatics program using metamorphic testing technique. In addition, metamorphic testing is also applicable to improve other software testing techniques (Chen et al., 2003; Chen et al., 2009b).

3. MODEL CHECKING

Model checking is an automatic technique for verifying finite state concurrent systems. Figure 2 shows the basic process of model checking. Model checking mainly consists of three tasks. The first task is to convert a design into a formalism accepted by a model checker. Kripke structure (Clarke et al., 1999) is normally used to model the system. A Kripke structure is a type of state transition graph that consists of a set of states, a set of transitions between states, and a labeling function assigning atomic propositions to states.

The second task is to state the properties that the system must satisfy. These properties are usually expressed using formulas of temporal logics (Emerson, 1990). Several temporal logics have been proposed to be used in model checking, such as linear temporal logic and computation tree logic. These logics provide different operators to express temporal properties, and vary in their expressive power.

The last task is to verify the design against the properties. If the constructed Kripke structure satisfies the specifications, the model checker terminates with a “yes” answer, which implies that the design is guaranteed free of error with respect to the property. Otherwise, the model checker terminates with a “no” answer and provides a counterexample, a sequence of states that demonstrates where the design violates the expected property. This will help developers find the error in the system design. Many model checking algorithms (Gerth et al., 1995) have been proposed. When the system being verified has many concurrent parts or data variables, model checking suffers from the “state explosion problem”. Although techniques such as symbolic representation, abstraction, and partial order reduction have been developed to alleviate this problem (Clarke et al., 1999), it is still one of the most challenging problems in model checking research. Another technique called bounded model checking was introduced by Biere et al. (1999). Bounded model checking is particularly suitable for certain kinds of infinite state systems such as software systems.

4. INTEGRATING METAMORPHIC TESTING AND MODEL CHECKING

Some researchers have investigated how to combine model checking with some traditional software testing techniques (Peled et al., 1999; Groce et al., 2002). In this section, we attempt to highlight several potential research topics on how to integrate metamorphic testing and model checking, as listed in the following. Such integration is inspired by the fact that both metamorphic testing and model checking are conducted based on some properties of the system under test, as shown in Sections 2 and 3.

• Selection of system properties.
In model checking, the system design is checked against some properties, which serve as the system specifications. In metamorphic testing, many properties are also derived from the system specifications, but these properties are identified specifically to examine whether the software under test is correct from certain perspectives, especially from the end-user's perspective. Metamorphic testing involves a lot of user domain knowledge, so the properties identified in metamorphic testing are very useful to ensure that the software under test can be correctly executed by end-users. Research (Chen et al., 2004; Mayer and Guderlei, 2006) has been conducted to develop some approaches to selecting effective metamorphic relations. It is worthwhile to see whether such approaches could be successfully applied in model checking.

**Enhancement of the effectiveness of detecting counterexamples.**

Counterexample generation is an important and useful feature of model checking, as counterexamples provide precise information for error fixing. Some researchers even used counterexamples to generate test cases (Beyer et al., 2004). Basically, model checker searches the state space of the Kripke structure to check if the structure satisfies its specifications. In case that it does not satisfy the specifications, a counterexample, usually in the form of a sequence of states, will be output to pinpoint the source of the error. Previous work (Chen et al., 1998) has shown that metamorphic testing can provide information about the constraints of the inputs that effectively reveal failures. Such information is very useful for debugging. It is thus natural to investigate the relationship between these failure-revealing constraints in metamorphic testing and the counterexamples in model checking. As shown in Figure 3, we propose to apply metamorphic relations into the selection of state sequences for checking. If the metamorphic relations are effective in revealing failure, it is intuitive that the resultant sequences of states will have a high chance to become a counterexample.

**Integration of model-based testing and metamorphic testing.**

Model-based testing (Pretschner, 2005; Utting and Legeard, 2007) is a software testing technique that generates test cases based on the model of the software under test. Various algorithms (Wu and Lin, 2006; Satpathy and Ramesh, 2007) have been proposed to derive test cases from a model. Model checking can also be used to guide the model-based test case generation. Since different test cases may have different failure-detection effectiveness, how to select “good” test cases is an important research topic for model-based testing. Since there are some successful investigations (Chen et al., 2004; Mayer and Guderlei, 2006) into selection of metamorphic relationship that could generate follow-up test cases which are more likely to reveal failures, it is intuitive to investigate how these results could be applied in the selection of good test cases for model-based testing. Figure 4 gives a diagram to integrate metamorphic testing and model-based testing, where some metamorphic relations are used to help select model-based test cases.

**Combination of partial order reduction and properties in metamorphic testing.**

Partial order reduction (Peled, 1996) is a technique to reduce the size of the state space that is searched by a model checker. Briefly speaking, this technique will provide the model checker a reduced state graph instead of the original full state graph, and such a reduced state graph will present a sufficient number of behaviours such that the model checker still provides right answers (Clarke et al., 1999). Apparently, the reduced state graph is a subset of the full state graph. The properties identified by metamorphic testing are also a subset of the system specifications. It is interesting to study whether the technique of deriving a subset
(that is, the system properties) of specifications in metamorphic testing can be used to improve the technique of constructing a subset (that is, the reduced state graph) of the whole state space in partial order reduction.

5. CONCLUSION

Metamorphic testing and model checking are two innovative techniques for software testing and verification. Since both techniques are conducted based on some properties of the system under test, it is intuitive to consider the integration of these two innovative techniques to improve the effectiveness of software testing and verification. A few potential topics were proposed in this paper, and we believe that they are all promising and worthwhile to be investigated.

ACKNOWLEDGEMENT

This project is supported by the Australian Research Council and a National Natural Science Foundation of China grant (No. 60721061).

REFERENCES