The Oracle Problem for Testing against Quantified Properties

Patricia D. L. Machado and Wilkerson L. Andrade
Formal Methods Group, Federal University of Campina Grande, Campina Grande, Brazil
{patricia,wilker}@dsc.ufcg.edu.br

Abstract

The oracle problem for testing from quantified properties is discussed and illustrated by proposed solutions for first-order logic and computation tree logic (CTL) properties. Unless constraints are placed, it is not always possible to generate feasible test suites from which accurate conclusions on their execution results can be reached regarding conformity with the intended property.

1 Introduction

The combination of formal methods and testing can help to produce high integrity systems in a cost-effective way. Informal specifications are not effective to uncover faults and deriving test suites from a formal specification has proven to be feasible and very promising as well as this is another way to compensate for the costs of producing it. In fact, formal specifications have been pointed out to be fundamental to the establishment of a testing theory [8, 14].

Conformance testing consists in deriving test suites from formal specifications so that testing can be rigorously applied whenever full formal verification is not cost-effective. From test execution results, important properties can be established regarding the possibility of acceptance/rejection of conforming/non-conforming implementations [14]. However, there are still obstacles to be overcome in order to establish testing as a standard in formal frameworks. Accurate interpretation of test results is a critical one.

The main issue addressed in this paper is the so-called oracle problem, i.e., whether a decision procedure can be defined for interpreting the results of tests according to a formal specification. The focus is on property oriented testing with test cases derived from first-order logic and computation tree logic (CTL) properties. Property oriented testing is a kind of conformance testing where the selection criteria is associated with a given property, verified on the specification, that needs to be checked [11]. The goal is to focus on particular, eventually critical, properties of interest of the system, possibly not previously checked [7]. The specification of one or more properties drives the test process that checks whether they are satisfied by an implementation. Properties are often stated as a test purpose, targeting testing at a particular functionality. In this context, the problem is that when testing from quantified properties, it is not always possible to generate feasible test suites that have decidable oracles as quantifiers often demand infinite test sets. Current solutions can only handle the problem by placing constraints either on the kind of properties or on the kind of implementations under testing that can be considered.

This paper is structured as follows. Section 2 briefly presents fundamental concepts and state-of-the-art on conformance testing focusing on testing from algebraic specifications and property oriented testing. Then, Section 3 discusses and illustrates the oracle problem for testing from algebraic specification axioms. After that, Section 4 discusses and illustrates the oracle problem for testing from CTL properties. Finally, Section 5 presents remarks and suggests areas for further research.

2 Background

2.1 Testing from Algebraic Specifications

The main goal of testing from algebraic specifications is to check whether specification axioms are satisfied by programs. Thus, oracles are usually active procedures which drive the tests and interpret the results according to a given axiom which needs to be checked. Test cases are extracted from specifications together with test sets which are also defined at specification level. Then, oracles are defined for each test case or group of test cases. These basically consist of predicates to evaluate test cases according to test results.

Testing from algebraic specifications has its basis in the works of Bougé *et al.* [3], Bernot *et al.* [2], and Gaudel [8]. Test case selection and oracles to decide on discrepancies between a program behaviour and a functional specification together with modularity are the main issues investigated. Solutions to the oracle problem are proposed in [2, 8, 9].

In this context, the oracle problem often reduces to

the more general problem of comparing values of a non-observable sort[1] which makes oracles undecidable in general. But the use of universal and existential quantifiers in specifications can make the oracle problem even worse. While Bernot et al [2, 8] focus on solutions that consider axioms as positive conditional equations, Machado [9] focuses on solutions for axioms expressed in first-order logic, considering both universal and existential quantifiers.

Generally, the theory of testing considered is based on the notion of *testing context* which is a triple (H, T, O), where T is a test data set, H is a set of hypotheses and O is an oracle. Test sets are instances of axioms and oracles are partial predicates which are either undecidable or decide whether a test is successful or not [2, 8].

2.2 Formal Framework for Testing

Conformance testing relates a specification and an implementation under test (IUT) by the relation **conforms-to** $\subseteq IMPS \times SPECS$, where $IMPS$ represents the universe of implementations and $SPECS$ represents specifications [14]. Then, IUT **conforms-to** s if and only if IUT is a correct implementation of s.

The **conforms-to** relation is hard to be checked by testing and the implementations are generally unsuitable for formal reasoning. Therefore, a test hypothesis is assumed: any IUT can be modelled by a formal object $i_{\mathrm{IUT}} \in MODS$, where $MODS$ represents the universe of models [1]. Then, an implementation relation **imp** $\subseteq MODS \times SPECS$ is defined so that IUT **conforms-to** s if and only if i_{IUT} **imp** s.

Let $TESTS$ be the domain of test cases and $t \in TESTS$ be a test case. Then EXEC(t,IUT) denotes the operational procedure of applying t to IUT. This procedure represents the test execution. Let an observation function that formally models EXEC(t,IUT) be defined as $obs : TESTS \times MODS \rightarrow \mathcal{P}(OBS)$. Then, $\forall$ IUT $\in IMPS$ $\exists i_{\mathrm{IUT}} \in MODS$ $\forall t \in TESTS \cdot$ EXEC(t,IUT) $= obs(t, i_{\mathrm{IUT}})$, according to the test hypothesis.

Let a family of verdict functions $v_t : \mathcal{P}(OBS) \rightarrow \{$**fail**, **pass**$\}$ which can be abbreviated to IUT **passes** $t \Leftrightarrow_{def} v_t($EXEC($t$,IUT)$) =$ **pass**. Then, for any test suite $T \subseteq TESTS$, IUT **passes** $T \Leftrightarrow \forall t \in T \cdot$ IUT **passes** t. Also, IUT **fails** $T \Leftrightarrow \neg($IUT **passes** $T)$. A test suite that can distinguish between all conforming and non-conforming implementations is called *complete*. Let $T_s \subseteq TESTS$ be complete. Then, IUT **conforms-to** s if and only if IUT **passes** T_s.

A complete test suite is a very strong requirement for practical testing. Then, weaker requirements are needed. A test suite is *sound* when all correct implementations and possibly some incorrect implementations pass it, that is, any detected faulty implementation is non-conforming, but not the other way around. Let $T \subseteq TESTS$ be sound. Then, IUT **conforms-to** $s \Rightarrow$ IUT **passes** T. The other direction of the implication is called *exhaustiveness*, meaning that all non-conforming implementations will be detected.

[1] A non-observable sort is the one that it is not identified with any particular concrete representation of a type in the target programming language.

2.3 Formal Test Purposes

Test purposes describe desired observations that we wish to see from the implementation during the test. Test purposes are related to implementations that are able to exhibit them by a well chosen set of experiments. This is defined by the relation **exhibits** $\subseteq IMPS \times TOBS$, where $TOBS$ is the universe of test purposes [6]. To reason about exhibition, we also need to consider the test hypothesis from Subsection 2.2 by defining the *reveal* relation **rev** $\subseteq MODS \times TOBS$, so that, for $e \in TOBS$, IUT **exhibits** e if and only if i_{IUT} **rev** e, with $i_{\mathrm{IUT}} \in MODS$ of IUT.

Let a verdict function $H_e : \mathcal{P}(OBS) \rightarrow \{$**hit**,**miss**$\}$ which can decides whether a test purpose is exhibited by an implementation. Then, IUT **hits** e **by** $t_e =_{def} H_e($EXEC(t_e,IUT)$) =$ **hit**. This is extended to a test suite T_e as IUT **hits** e **by** $T_e =_{def} H_e(\bigcup\{$EXEC($t$,IUT) $\mid t \in T_e\}) =$ **hit**, which differs from the **passes** abbreviation.

An *e-complete* test suite can distinguish among all exhibiting and non-exhibiting implementations, such that, IUT **exhibits** e if and only if IUT **hits** e **by** T_e. An *e-exhaustive* test suite can only detect non-exhibiting implementations (IUT **exhibits** e implies IUT **hits** e **by** T_e), whereas an *e-sound* test suite can only detect exhibiting ones (IUT **exhibits** e if IUT **hits** e **by** T_e). Note that the purpose of the *sound* test suites and *e-sound* test suites is similar, although the implications are relatively inverted. *Sound* test suites can reveal the presence of faults, whereas the *e-sound* can reveal intended behavior.

2.4 Relating Formal Test Purposes to Model Checking Theory

From [5], the model checking problem is defined as: given a kripke structure M, which models a concurrent finite state system and a temporal logic formula f expressing a property p, identify the set of states S of M that satisfy f. Formally: $\{s \in S \mid M, s \models f\}$.

Consider a given specification m_{IUT} as a kripke structure and a model $i_{IUT} \in MODS$ that implements it. If there is a set of states in m_{IUT} that satisfies a given property p, then i_{IUT} is able to reveal p. Assuming that p can be expressed as a temporal logic formula f and by a test purpose e, we can establish that: i_{IUT} **rev** $e \Longleftrightarrow \exists s \in S : m_{IUT}, s \models f$.

The states satisfying f form sets of states that represent the property p w.r.t. the specification m_{IUT}. These sets contain states related by a predecessor/successor relation,

i.e., traces of the kripke structure representing p. As these traces correspond to abstract specifications of p, they may be used to guide the generation of test purposes.

3 Interpreting Test Results from First-Order Logic Axioms

Machado [9] presents a well-founded solution to the oracle problem, named the grey-box approach, for testing from first-order logic axioms of algebraic specifications. The main goal is to address the problem posed by checking equality between non-observable sorts as well as taking the quantifiers problem into account so that a verdict on test cases execution results can be reached regarding conformity. The solution proposes the definition of two approximate equalities - one defined from the specification (black-box) and the other from the implementation (white-box). These equality procedures are applied to evaluated a given equality according to its syntactic position in the first-order formula. For the sake of space, the equality problem and its corresponding part of the solution are not detailed here and are in fact ignored from this point on.

As for quantifiers, the solution can only be applied under the following constraints. Let f be a first-order formula that represents the axiom from which test cases are derived.

1. If f contains only positive occurrences of $\forall$ and negative occurrences of $\exists$, then failure in testing means non-conformity, i.e, any test suite generated from f is sound. But success does not mean conformity.

2. If f contains only negative occurrences of $\forall$ and positive occurrences of $\exists$, then success in testing means conformity, i.e., any test suite generated from f is exhaustive. But failure thus not imply non-conformity.

For other combinations of occurrences of $\forall$ and $\exists$ in positive/negative positions, it is not possible to reach general conclusions on soundness and exhaustiveness of generated test suites. Positive and negative positions are defined as usual in logic. Negative positions are those with an odd number of negations applied to and left hand side of an implication. Parenthesis also need to be considered.

Constraint (1) is not a problem in practice, since it is satisfied by most common specification idioms and also this is the most common form of axioms - outermost universally quantified axioms. Accepting incorrect programs is tolerable, whereas rejecting correct ones is not acceptable due to the misleading and unnecessary costs that this may incur when trying to pinpoint non existent faults.

Note that even the empty test set is sound in (1). However, in practice, we must pursue effective test sets that uncover the biggest number of important faults. Thus, the practical use of this result is to guide the appropriated interpretation of test results. One has to consider effective test case selection techniques for taking real benefits of testing.

Proofs regarding (1) and (2) are presented in [9, 10]. Roughly, the idea is that, in practice, we can only define finite test sets. When the universal quantifier is in a positive position and it evaluates to true, this may make f evaluate to true as well. Also, if f is to be evaluated to true on the complete, possibly infinite test set, it will also be evaluated to true on finite test sets. On the other hand, the existential quantifier is subject to failure whenever the witness is not included in the test suite. Being in a negative position, this failure does not lead to evaluate the expression to false. This avoids the rejection of conforming implementations.

4 Interpreting Test Results from CTL Properties

Model checking is a technique used to verify, in a rigorous and automated way, the correctness of reactive, concurrent and distributed system models [5]. Using a formal model and a property specified in temporal logic formalisms (e.g. LTL [12], CTL [4]), a software called model checker must verify the satisfiability of the formula w.r.t. the model. However, the same rigor is not usually applied when testing implementations, creating a large gap between these processes and making the presence of failures on the implementation possible even in points where the specification was successfully corrected. A strategy to reduce this gap is presented in [13] where test purposes are generated from such properties specified in temporal logic formulas.

To achieve this goal, the work presented in [13] aims to perform analysis over the model through its state space, like model checking does. However, the process is adapted to get enough information for the test purpose generation in addition to the correctness verification of the model. The approach consists in an adaptation of a model checker algorithm to extract model traces representing examples and counter-examples (if there are any) from the state space and later analysis over these traces to generate an abstract graph representing the test purpose.

The temporal logic formalism used in the approach cited above is CTL. In contrast to LTL, that considers to each moment in the time only one future, CTL can consider different possible futures. The execution of the system can be represented based on ramifications, as a tree, being able to have a lot of possibilities. Such possibilities reveal executions that can be quantified based on one (existential quantifier) or in all (universal quantifier) the execution traces of the system.

CTL formulas can be composed of special operators, propositional logic operators and atomic propositions. The operators of this logic always occur in pairs: they are formed by the universal quantifier A or existential quanti-

fier E, followed by the operators F, G, U or X. The F operator indicates the future idea. The G operator implies that the formula must globally be satisfied, that is, during all the execution of the system. The U operator is a binary operator which implies that the first argument must be true until the second becomes true. Finally, the X operator implies the satisfaction of the formula in the state subsequent to the current one.

The verification of a CTL model is based on the verification of the state space represented as ramifications. The verification process of a CTL formula is reduced to an exhaustive search problem, where the model checker traverses all state space looking for states that satisfy the formula.

Reactive systems are inherently non-terminating, since they are systems that interact with their environment by accepting inputs and producing outputs. In this context, the state space is infinite, causing some problems in the test purpose generation from quantified properties.

The first problem is that the test purpose obtained cannot guarantee e-completeness of the resulting complete test graph (CTG) from which test cases are generated. There can be infinitely many examples and counter-examples, but the technique presented in [13] considers only finite examples and counter-examples because the test cases need to be finite. Moreover, the test purpose obtained can guarantee neither e-exhaustiveness of the CTG, because not all examples are considered, nor e-soundness, because not all counter-examples are considered.

The second problem is that the test cases obtained from that CTG cannot guarantee soundness due to the fact that only finite examples and counter-examples are considered. In this context, correct systems can be rejected if the formula is satisfied after the point considered by the extracted example or counter-example.

5 Concluding Remarks

The oracle problem constitutes a serious obstacle to the development of a testing theory that can provide a solid background from which testing activities can be more effective, reliable and productive in practice. The main goal of this paper is to present a taste of this problem considering different domains and motivate further research in this area. We focused on testing of quantified properties.

The problem discussed when testing from algebraic specification has also been investigated in the context of class testing considering different formalisms. Nevertheless, not much progress has been made. Solutions should necessarily be constrained by testing hypothesis and tailored to particular application domains. Also, test set selection may play an important role: properties of test suites may eliminate the syntactic constraints (valid and unbiased test sets [1, 10]).

As for testing from CTL properties, the problem is even harder. Soundness and exhaustiveness may not be met at all in the general case. For instance, when considering existentially quantified properties, witnesses may be infinite traces of the model, that may be casually reached by an implementation. On the other hand, even if some witnesses are found, since the systems considered are usually non-deterministic (distributed systems), it may not be feasible to force the deterministic execution of any of the traces. Solutions on this domain also need to place constraints as testing hypotheses such as finite/partial models, syntactic position of quantifiers and be supported by effective test case selection. Also, on-the-fly generation techniques are critical. Furthermore, the real challenge is to find testing hypothesis that can be met in practice.

References

[1] G. Bernot. Testing against formal specifications: a theoretical view. In *Proceedings of TAPSOFT'91: Vol. 2*, pages 99–119, New York, NY, USA, 1991. Springer-Verlag.

[2] G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, 1991.

[3] L. Bougé. A contribution to the theory of program testing. *Theoretical Computer Science*, 37:151–181, 1985.

[4] E. Clarke and E. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. *Lecture Notes in Computer Science*, 131, 1981.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[6] R. G. de Vries and J. Tretmans. Towards formal test purposes. In *Proceedings of (FATES'01)*, volume NS-01-4 of *BRICS Notes Series*, pages 61–76, Aarhus, Denmark, 2001.

[7] J. Fernandez, L. Mounier, and C. Pachon. Property oriented test case generation. In *Proceedings of FATES 2003*, volume 2931 of *LNCS*, pages 147–163. Springer, 2004.

[8] M.-C. Gaudel. Testing can be formal, too. In *Proceedings of TAPSOFT'95*, volume 915 of *LNCS*. Springer, 1995.

[9] P. D. L. Machado. On oracles for interpreting test results against algebraic specifications. In *Proceedings of AMAST'98*, volume 1548 of *LNCS*, pages 502–518. 1999.

[10] P. D. L. Machado. *Testing from Structured Algebraic Specifications: The Oracle Problem*. PhD thesis, LFCS, University of Edinburgh, UK, 2000.

[11] P. D. L. Machado, D. A. Silva, and A. C. Mota. Towards property oriented testing. *Electronic Notes in Theoretical Computer Science*, 184:3–19, 2007.

[12] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium Foundations of Computer Science*, pages 46–57, 1977.

[13] D. A. Silva and P. D. L. Machado. Towards test purpose generation from ctl properties for reactive systems. *Electr. Notes Theor. Computer Science*, 164(4):29–40, 2006.

[14] J. Tretmans. Testing concurrent systems: A formal approach. In *Proceedings of CONCUR'99*, pages 46–65. Springer, 1999.