Model Learning and Model-Based Testing

Bernhard K. Aichernig[1], Wojciech Mostowski[2], Mohammad Reza Mousavi[2,3],
Martin Tappler[1], and Masoumeh Taromirad[2]

[1] Institute of Software Technology
Graz University of Technology, Austria
[2] Centre for Research on Embedded Systems
Halmstad University, Sweden
[3] Department of Informatics,
University of Leicester, UK

Abstract. We present a survey of the recent research efforts in integrating model learning with model-based testing. We distinguished two strands of work in this domain, namely test-based learning (also called test-based modeling) and learning-based testing. We classify the results in terms of their underlying models, their test purpose and techniques, and their target domains.

1 Introduction

On one hand, learning (functional or behavioral) models of software and computer systems (e.g., hardware, communication protocols) has been studied extensively in the past two decades. Various machine learning techniques [Mit97,Alp14] have been adopted to this domain and new domain-specific techniques have been developed for model learning (cf. the chapters on (Extended) Finite Stat Machine learning in this volume).

On the other hand, testing has been the dominant verification and quality assurance technique in industrial practice. Traditionally, testing has been an unstructured and creative effort in which requirements and domain knowledge is turned into a set of test cases, also called a test suite, while trying to cover various artifacts (such as requirements, design, or implementation code). Model-based testing (MBT) [UPL12,UL07] is a structured approach to testing in which the testing process is driven by a model (e.g., defining the correct behavior of the system under test, or specifying the relevant interactions with the environment).

The focus of the present paper is precisely in the intersection of the above-mentioned two fields: learning (functional or behavioral) models and model-based testing. In this intersection fall two types of research:

1. *test-based learning*: various (active) learning techniques make queries to the to-be-learned system in order to verify a learning hypothesis. Such queries can be tests that are generated from a learned model. We refer to this strand of work as test-based learning or test-based modeling [MNRS04,Tre11].

2. *learning-based testing*: models are cornerstones of model-based testing; however, complete and up-to-date models hardly ever exist. Learning can hence be used to create and complement models for model-based testing. We refer to this category of work as learning-based testing [MS11].

To structure our survey of the field we focus on the following classification criteria:

1. Types of models: different types of models have been learned and have been used for model-based testing. We distinguish the following categories of models: predicates and functions, and logical structures (such as Kripke structures, cf. the chapter on logic-based learning in this volume), finite state machines (including their variants and extensions, cf. the chapters on FSM and Extended FSM learning, as well as learning-based testing in this volume), and labeled transition systems. The distinction between variants of these models is not always well-defined and there are several property-preserving translations among them. However, this classification gives us a general overview and a measure of matching between different learning and testing techniques.
2. Types of testing: requirement-based and conformance testing are the most prominent uses of model-based testing. However, other types of model-based testing have also been considered in combination with learning; these include: integration testing, performance testing, and security testing.
3. Domain: test-based learning and model-based testing have been applied to various domains, such as embedded systems, network protocols, and web services. If a research result considers a particular application domain, we classify the result in terms of the domain, as well.

The rest of this paper is organized as follows. In Section 2, an overview of model-based testing is provided. In Section 3, the basic ideas behind model learning and their relation to testing are presented. In Section 4, we review the types of models that have been used in integrating learning and testing and survey the different pieces of research related to each type of model. In Section 5, we classify the test purposes and testing techniques that have been considered in combination with learning. In Section 6, we review the domains to which the combination of testing and learning has been applied. Finally, we conclude the survey in Section 7 by pointing out some of the open challenges in this domain.

2 Model-Based Testing

Model-based testing (MBT) is a structured testing technique in which models are used to guide the testing process. Specification test models can, for example, describe the input-output functionality of a unit (function, class, module, or component) [HRD07,MN10], specify the state-based behavior of a unit [UL07] or a system [VT14], or sequences of interactions with graphical user interface [YCM09]. Ideally such specification models have a mathematical underpinning, i.e., have a formal semantics; such formal models include algebraic properties, finite state machines, and labeled transition systems. Once specification test models are in place, much of the testing process can be mechanized thanks to various MBT techniques and algorithms.

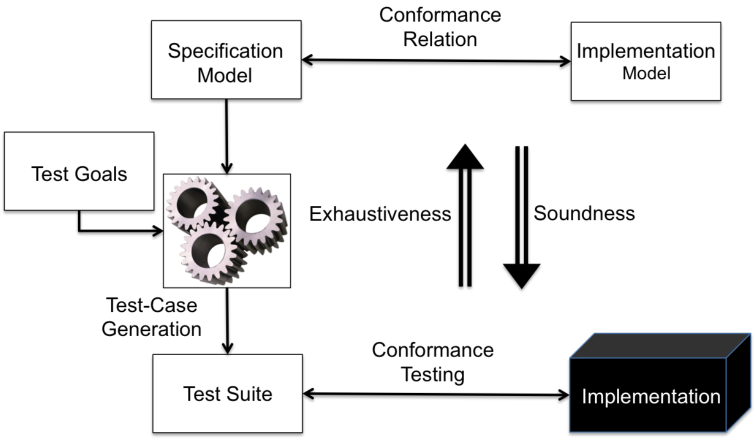

Fig. 1. An Overview of Model-Based Testing [ARM16,UPL12]

Figure 1 presents a general overview of MBT theory and practice. The underlying assumption of MBT is the existence of a formalization of the requirements in the form of a specification test model. This is a highly non-trivial assumption; models are often absent or incomplete in practice. Learning is a technique that can help reinstate the underlying assumption of MBT.

To put MBT on firm formal grounds, a common assumption is that the behavior of the implementation under test can be described by some (unknown) model with the same mathematical underpinning as the specification test model. This enables grounding the theory of MBT in a mathematical definition of a conformance relation between the specification model and the purported implementation model.

One of the most important ingredients of a practical MBT approach is a test-case generation algorithm that can automatically generate a test suite (a set of test cases) from the specification model (in an online or offline manner), taking into account the specified test goals. Then using a mechanized adapter the generated abstract test suite can be translated into concrete test cases that are executed on the system under test (which is traditionally considered to be a black box). The results of the test execution are then compared with the results prescribed by the specification test model.

The formal notion of conformance and the conformance testing algorithm are linked through soundness and completeness theorems. Soundness states that conformance testing never rejects a conforming implementation and exhaustiveness states that conformance testing is able to reject all non-conforming implementations. A sound and exhaustive conformance testing algorithm is called complete.

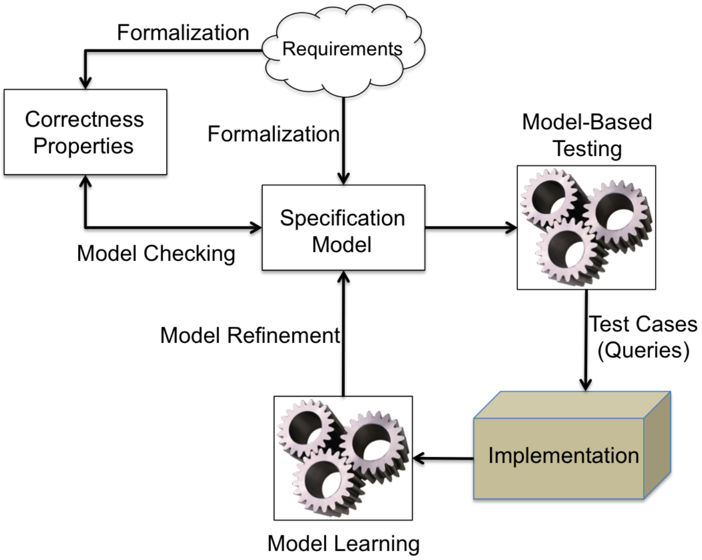

Fig. 2. Creating Models for Model-Based Testing

Specification test models can be learned from (reference) implementations and validated or verified by the domain experts, e.g., by manual inspection or model checking (as well as equivalence checking tools); Figure 2 illustrates this process. Also incomplete or outdated models can be augmented or corrected (possibly with user feedback) using learning techniques.

Since the scope of this paper is the combination of model-based testing and learning, we only explore the part of the literature that serves at least one of the following two categories of purposes (cf. the chapter on testing stateless black-box programs in this volume for a complementary survey):

1. Model-based test-based learning, i.e., the use of model-based testing as a teaching mechanism in learning models, or
2. Learning-based model-based testing, i.e., the use of learning techniques to come up with models (of specification or implementation) in the model-based testing process.

3 Learning

In this section, we review the main ideas concerning model learning and their connections to (model-based) testing. We mainly consider active automata learning in the minimally adequate teacher (MAT) framework as introduced by Angluin [Ang87], since it shares clear common grounds with testing; for other machine learning techniques (some of which are also used in combination with model-based testing), we refer to [Mit97,Alp14].

Generally, this framework requires the existence of a teacher (called MAT) with which the *learner* interacts in order to learn (1) how accurate the currently learned model is and (2) how the system reacts to some new patterns that are of interest for improving the model. To this end, the MAT must be able to answer two respective types of queries: (1) equivalence queries, which check whether the currently learned model is an accurate model of the system under learning and (2) membership queries, which provide the system reaction to specified patterns of input. This setup is shown in Figure 3. In fact, it illustrates an instantiation of this framework for black-box systems. Since ideal equivalence queries usually cannot be implemented, they have to approximated via model-based testing. Failing tests serve as counterexamples in such implementations, while the learned model and the system under learning are considered equivalent if they agree on all executed tests. The relationship between learning and testing is detailed further below.

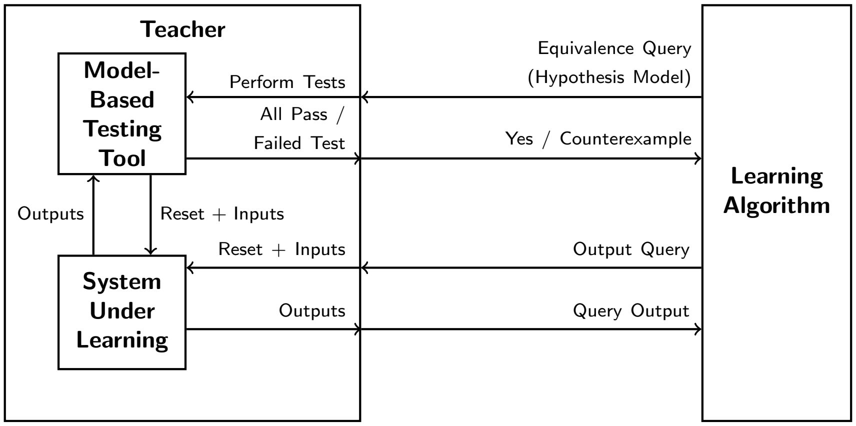

Fig. 3. Learning setup in the MAT framework. Figure adapted from a figure in [SMVJ15].

In the original L^* algorithm by Angluin, a deterministic finite automaton (DFA) representing an initially unknown regular language is learned. Membership queries correspond to the question whether some string is in the target language. In equivalence queries, the learner asks whether the language of a hypothesized DFA is equivalent to the target language.

These queries enable the learner to follow a two-step procedure in which it gains knowledge by posing membership queries. If there is sufficient information to create a hypothesis, an equivalence query is issued. The teacher either answers *yes*, signaling that learning is finished, or it responds with a counterexample to equivalence. Such a counterexample is then processed by the learner which eventually starts another round of learning.

Several variations of this general learning process have been proposed. All of them have in common that two types of queries are posed in an interleaved and iterative manner. As an example, consider learning of Mealy-machine models [Nie03,MNRS04,SG09]: instead of posing membership queries, the learner asks output queries [SG09], i.e., it asks for a sequence of outputs produced in response to a sequence of inputs. Analogously to L^*, equivalence queries are issued whereby a counterexample is a string of inputs for which the system under learning (SUL) and the current hypothesis produce different outputs.

3.1 Relation between Learning and Testing

Early work relating testing and program inference predates Angluin's L^* algorithm. Weyuker [Wey83] proposed a program-inference-based test-adequacy criterion. She points out the importance of distinguishing between test-selection criteria and test-adequacy criteria. The latter should be used to assess if a passing test set contains sufficient data. For that she proposes to infer a program from a test set and deem it adequate if the inferred program is equivalent to both program and specification. Noting that checking equivalence is in general undecidable, she suggest that equivalence checks may be approximated by testing as is usually done for equivalence queries in active automata learning.

More recently, Berg et al. [BGJ$^+$05] discussed the relationship between conformance testing and active automata learning, referred to as regular inference. Basically, both techniques try to gain information about a black-box system based on a limited number of observations, but with different goals. One technique solves a checking problem and the other a synthesis problem. They showed that a conformance test suite for a model m provides enough information to learn a model isomorphic to m. Conversely, observations made during learning a model m form a conformance test suite for m. This resembles the intuition behind Weyuker's work [Wey83]: a test set should contain information to infer a program equivalent to the original program.

Aside from the theoretical relationship, they referred to another connection between learning and testing. Since equivalence oracles do not exist in general, they can be approximated by conformance testing (as shown in Figure 3). Hence, in practice a testing problem has to be solved each time an equivalence query is issued. Two examples of commonly used equivalence testing methods are the W-method [Vas73,Cho78] and partial W-method [FvBK$^+$91], the latter aiming at improving efficiency. Both of these have for instance been implemented in the automata-learning library LearnLib [IHS15].

3.2 Test Case Selection vs. Query Minimization

Since exhaustive model-based testing is usually infeasible, it is necessary to select a subset of test cases based on some user-specified criterion [UPL12]. In other words, the number of tests has to be reduced. Because of the relationship described above, it can be concluded that a reduction of queries is required for learning as well. There are several possibilities for implementing such measures. Most importantly, abstraction is essential for learning to be feasible. While abstraction is mostly done manually, techniques have been developed to derive abstraction automatically through counterexample-guided abstraction refinement [AHK$^+$12,Aar14,HSM11]. In addition to that, we give three examples for ways to reduce the number of tests required for learning.

Algorithmic Adaptations. Following the work of Angluin [Ang87], shortcomings of the L^* algorithm have been identified and optimizations have been developed. A well-known example of such an optimization is the adapted counterexample processing proposed by Rivest and Schapire [RS93]. They extract a single suffix from a counterexample which distinguishes states in the current hypothesis. As a result, the observation table size and thereby the required membership queries are reduced.

Equivalence Testing Optimisations. Well-known methods for conformance testing are the W-method [Vas73,Cho78] and partial W-method [FvBK$^+$91]. Thus, they may be used to check whether the current hypothesis is equivalent to the SUL. However, they suffer from two drawbacks. Firstly, they assume a known upper bound on the number of states of the SUL. Since we consider black-box systems, we cannot know such a bound. Furthermore, their complexity grows exponentially in the difference of the number of states of hypothesis and SUL. This makes the application in industrial scenarios impractical. Alternative ways of selecting tests should thus be considered. The ZULU challenge [CdlHJ09] called for solutions to this issue. Competing approaches were only allowed to pose a limited number of membership queries/tests. This resembles a setting in which the cost of test execution matters and equivalence has to be checked via testing.

Howar et al. [HSM10] describe that a different interpretation of equivalence queries is necessary in this case. Rather than testing for equivalence, it is necessary to find counterexamples fast. This is a reasonable approach, as learning is inherently incomplete anyway, because of its relation to black-box testing. Furthermore, they discuss their approaches to selecting test cases which are based on heuristics. They consider hypotheses to be evolving, i.e. testing is not started from scratch once a new hypothesis is constructed. Additionally, they base their test selection on the improved counterexample handling [RS93], combined with randomization.

Efficient equivalence testing has been addressed by Smeenk et al. [SMVJ15] as well. Since their SUL is too large for testing with the W-method, they developed a randomized conformance testing technique. It is based on a method for finding adaptive distinguishing sequences described by Lee and Yannakakis [LY94]. In addition to that, they selected a subset of the original alphabet which they tested more thoroughly. This is done to ensure that specific sequences relevant to the initialization of the considered application are covered although it would be unlikely to select them otherwise.

Another randomized conformance testing technique for automata learning has been presented in [AT17a]. It addresses coverage by mutation-based test-case selection whereby the applied mutations are tailored to the specifics of learning. Furthermore, stochastic equivalence checking has for instance been applied in learning-based testing to measure convergence [MN15].

Purely random testing, without taking heuristics into account, is a viable option as well. It has successively been used for experiments with the tool Tomte [AHK$^+$12,AFBKV15]. However, Aarts et al. [AKT$^+$14] also point out that while being effective in most cases, random testing may also fail if the probability of reaching some state is low. Still, quantitative analysis of learned models, e.g. giving some confidence for the correctness of the models, are mostly lacking. This is despite early work discussing such ideas [Ang87,RS93].

Domain-Specific Optimisations. Another important insight is that the inclusion of knowledge about the application domain can increase learning performance. This has for instance been shown by Hungar et al. [HNS03], who applied techniques such as partial-order reduction methods to reduce the number of queries. Another example of a domain-specific optimization is the modification of the W-method by de Ruiter and Poll [dRP15].

3.3 State Merging Techniques

A prominent alternative to learning in the MAT framework is learning via state merging. State merging techniques infer models from given samples, that is, sequences of symbols. This is usually done passively, i.e. without interaction with a teacher. Prominent examples are the RPNI algorithm [OG92] and ALERGIA [CO94]. In a first step, state merging techniques generally build a prefix tree acceptor (PTA) from the given samples. They then iteratively check nodes in the tree for compatibility and merge them if they are compatible. The tree is transformed into a finite automaton through this procedure. Depending on the actual algorithm, different techniques are used for the steps in this generic procedure and different types of models are created.

In the case of RPNI for instance, a deterministic finite automaton is inferred and samples are split into negative and positive samples. Furthermore, the PTA is built from positive samples while negative samples are used to check whether two nodes may be merged. ALERGIA requires only positive samples to learn a probabilistic finite automaton. Therefore, it augments the PTA with frequencies and bases its compatibility check on a statistical test.

The QSM algorithm is an interactive state-merging algorithm with membership queries [DLDvL08]. Hence, it is a query-driven State-Merging DFA induction technique. The induction process starts by constructing an initial DFA covering all positive scenarios only. The induced DFA is then successively generalized under the control of the available negative scenarios and newly generated scenarios classified by the end-user (membership queries). This generalization is carried out by successively merging well-selected state pairs of the initial automaton.

4 Models

In this section, we provide an overview of the kind of models that have been learned for testing. Most of the work concentrates on different types of finite state machines and labeled transition systems. Some researchers have considered other models, e.g. for stateless systems.

4.1 Finite State Machines

In [AKT$^+$12,AKT$^+$14], the authors use a combination of automata learning techniques to learn a model of the implementation, which is then compared to a reference specification model using equivalence checking techniques,.

In [LGS06a], the authors use an approach based on L^* to learn Mealy machines, which is extended and more thoroughly described in [SG09]. Other work considers more expressive versions of Mealy machines [LGS06b,SLG07a], which include parameters for actions, predicates over input parameters and allow for observable non-determinism.

Margaria et al. [MNRS04] optimized the L* algorithm for generalized Mealy machines, i.e. Mealy machines that may produce a sequence of outputs rather than exactly one output in response to a single input. They report significant performance gains as compared to learning DFA models.

In [CHJS14,CHJS16], Cassel et al. consider generating models from test cases and present a framework for generating a class of EFSM models, called register automata, from black-box components using active automata learning. They introduce an extension to the L* algorithm called *SL** (for *Symbolic L**). However, they do not explicitly mention any particular testing technique. They only suggest using conformance testing in hypothesis validation (i.e., providing counterexamples). The SL* algorithm is available as an extension to LearnLib [IHS15], namely RaLib.

Ipate et al. [ISD15] propose an approach which, given a state-transition model of a system (EFSM), constructs an approximate automaton model and a test suite for the system. The approximate model construction relies on a variant of Angluins automata learning algorithm, adapted to *finite cover automata* [CSY99]. In parallel with automata construction, they incrementally generate conformance test suites for the investigated models, using the W-method [Cho78] adapted to bounded sequences. These test suites are used to find counterexamples in the learning process. Their approach is presented and implemented in the context of the Event-B modeling language [DIMS12,DIS12].

Arts et al. [AT10] automatically extract finite state machines from sets of unit tests using an FSM inference technique, namely StateChum [WBHS07]. Then, the inferred FSMs are used to provide feedback on the adequacy of the set of tests and to develop properties for testing state-based systems. They use QuickCheck for testing and thus, consider generating QuickCheck properties. An FSM model is incrementally extracted from the test suite as it evolves.

In [RMSM09] a method for learning-based testing is presented, where the alphabet of the system under learning is progressively extended during the process based on previous interactions. This extension, and the knowledge gained about the system is used to further derive test cases. The method uses classic deterministic Mealy machines and the LearnLib for learning, and it is showcased with the Mantis Bug Tracker case study.

Relying on a heuristic approach to model inference, Schulze et al. [SLBW15] discussed an model-based testing supported by model generation. They propose to generate a model from manually created test cases in order to generate further tests from this model which possibly find undetected issues. In the case study, they report on manual effort for GUI testing a web-based system.

4.2 Labeled Transition Systems

Hagerer et al. [HHNS02] presented a technique called regular extrapolation for learning labeled transition systems (LTS) with inputs and outputs. For testing purposes, labels and states may have additional observations, i.e. parameters and attributes. Their technique starts with a set of abstract traces, either gathered passively via log-files or actively via testing. These traces are merged into a tree and then states with equivalent observations, i.e. equivalent attributes, are merged. Furthermore, a user may specify independence relations in order to simplify the model via partial order reduction. Model checking is used to verify if the learned model satisfies a set of Linear Temporal Logic (LTL) specifications.

Hungar et al. [HNS03] used the L* algorithm to learn LTS models with inputs and outputs that are input-enabled and input-deterministic. Several optimizations for reducing the number of membership queries are presented, most notably the application of partial-order reduction techniques that exploit domain-specific independence and symmetry properties.

Walkinshaw et al. [WDG09] introduce a reverse-engineering technique which infers state machines, in the form of LTS, from implementations. They use active state-merging techniques [DLDvL08] for learning a model based on program executions and model-based testing in refining the hypothesis model. The learning process starts with an initially small set of execution traces, based on which an initial hypothesis model is constructed. Then, iteratively, a given MBT framework automatically generates tests from the hypothesis model which are executed in the program. Any test conflicting the expected behavior by the model would restart the process to construct a refined hypothesis model. The process iterates until no more conflicts can be found by testing. For model inference, they use StateChum, developed by the authors [WBHS07], and use QuickCheck for MBT [AHJW06].

Walkinshaw et al. [WBDP10] use the technique introduced in [WDG09] and propose inductive testing to increase functional coverage in the absence of a complete specification.

Tretmans [Tre11] discusses both learning-based testing as well as testing-based learning. It is rightfully noted that intermixing the two directions is dangerous due to a risk of a circular dependency in the resulting testing process. Most approaches by Tretmans, employ ioco-based conformance testing methods, and they treat both deterministic and non-deterministic models given as Mealy machines. The learning process is delegated to the LearnLib suite with custom extensions to facilitate better learning, Volpato and Tretmans [VT14] extend the Angluin's L* algorithm to work with non-determinism in input-output labeled transition systems. The ioco-based testing methodology is implemented in the TorXakis tool [TB03] and employs random model exploration to generate tests. The learning approach is further improved in subsequent work [VT15] which weakens assumptions related to the completeness of information obtained during learning. An important improvement is that the new approach does not require exhaustive equivalence checks.

Groz et al. [GLPS08] present inference of *k-quotients* of FSMs, but also of input output transition systems (IOTSs). They address the composition IOTSs and asynchronous communication between components. The latter is accounted for by introducing queues modeled by IOTSs.

4.3 Other Models

Meinke and Sindhu [MS11] apply the learning-based testing paradigm to reactive systems and present an incremental learning algorithm for Kripke structures.

For stateless behavior, predicates and functions provide a natural abstraction for the input-output functionality of programs. In [BG96], inductive program learning (and inductive logic programming) is used to learn the behavior of programs; the technique is used to generate adequate tests in order to distinguish the program under test from all other alternative programs that can be learned. In [HRD07], algebraic specifications of Java programs are learned. In [Mei04,MN10], functional models of numerical software are learned and the learned models are used for automatic generation of unit tests.

Walkinshaw and Fraser presented *Test by Committee*, test-case generation using *uncertainty sampling* [WF17]. The approach is independent of the type of model that is inferred and an adaption of *Query By Committee*, a technique commonly used in active learning. In their implementation, they infer several hypotheses at each stage via genetic programming, generate random tests and select those tests which lead to the most disagreement between the inferred hypotheses. In contrast to most other works considered, their implementation infers non-sequential programs. It infers functions mapping from numerical inputs to single outputs. Papadopoulos and Walkinshaw also considered similar types of programs, but in a more general learning-based testing setting [PW15]. Therefore, they presented the Model-Inference driven Testing (MINTEST) framework which they also instantiated and evaluated.

5 Test Purposes and Types of Testing

5.1 Behavioral Conformance Testing

Behavioral conformance testing is a common form of model-based testing, in which tests are generated in order to establish whether the behavior of the implementation under test is "equivalent" to that of the specification model, according to a well-defined notion of equivalence. Typically behavioral conformance testing is integrated with model-learning in that the specification test models are learned and are subsequently used for generating a conformance test suite [VT15,ASV10]. However, in [AKT$^+$14], an alternative integration is also explored. Namely, model learning is used to learn both a model of a reference implementation and the implementation under test and then equivalence checking tools are used to check the equivalence between the two learned model. This way conformance checking is performed in an intensional manner by comparing models rather than by generating test cases from the specification model and executing test cases on the implementation.

A case study following a similar approach is presented in [TAB17]. However, instead of comparing to the model of a reference implementation, learned models of implementations are compared among each other. Detected differences are considered to point to possible bugs which should be analyzed manually. Experiments involving five implementations of the MQTT protocol revealed 18 errors in all but one of the implementations. The system HVLearn described by Sivakorn et al. [SAP$^+$17] follows a similar approach. It learns DFA-models of SSL/TLS hostname verification implementations via the KV algorithm [KV94]. Given learned models, HVLearn is able to list unique differences between pairs models and additionally provides analysis capabilities for single models. The authors reported that they found eight previously unknown unique RFC violations by comparing inferred models. Another example using a similar technique in the security domain is SFADiff [ASJ$^+$16]. In contrast to the other approaches, it learns symbolic finite automata (SFA) and is able to find differences between pairs of sets of programs, e.g., for fingerprinting or creating evasion attacks against security measures. It has been evaluated in case studies considering TCP state machines, web application firewalls and parsers in web browsers.

These approaches to conformance testing between implementations can in general not guarantee exhaustiveness. In other words, if models are found to be equivalent this does neither imply that the implementations are equivalent nor that the implementations are free of errors. In testing of complex systems, however, the reverse will often hold, i.e. there will be differences. These may either help to extend the learned models in case learning introduced the differences, or may point to actual differences between systems. The discussed case studies showed that such differences can be exploited in practice, e.g., to find bugs.

5.2 Requirements-based Testing

With the introduction of black box checking, Peled et al. [PVY99] pioneered a line of research combining learning, black-box testing and formal verification.

In order to check whether a black-box system satisfies some formally-defined property, a model is learned with Angluin's L^*-algorithm and the property is checked on this model. If a counterexample is found, it either shows that the property is violated or it is spurious and can be used to extend the model. To avoid false positives, conformance testing as described by Vasilevskii [Vas73] and Chow [Cho78] is also used to extend the model, i.e., to implement equivalence queries.

Following that, several optimisations and variations have been proposed. Adaptive model checking [GPY02a,GPY02b] optimizes black box checking by using a model of the system which is assumed to be inaccurate but relevant. Another early developed variation is grey-box checking [EGPQ06], which considers a setting in which a system is composed of some completely-specified components and some black-box systems. With regard to testing, the VC-method [Vas73,Cho78] and other conformance testing approaches, taking the grey-box setting into account, are used and compared.

Adaptive model-checking combined with assume-guarantee verification has also been considered for the verification of composed systems [HK08]. Furthermore, another variation of adaptive model-checking has been described by Lai et al. [LCJ06]. They use genetic algorithms instead of L^* in order to learn a system model. Their results show promising performance for prefix-closed languages.

Meinke and Sindhu [MS11] applied the learning-based testing paradigm to reactive systems and present an incremental learning algorithm for Kripke structures. Here, an intermediate learned model is model checked against a temporal specification in order to produce a counter-example input stimulus. The SUT is then tested with this input. If the resulting output satisfies the specification, then this new input-output pair is integrated into the model. Otherwise, a fault has been found and the algorithm terminates.

Following ideas of black box checking, a testing approach for stochastic systems is presented in [AT17b]. It focuses on reachability properties and basically infers testing strategies which optimize the probability of observing certain outputs. This is done via iterated model-inference, strategy generation via probabilistic model-checking, and property-directed sampling, i.e. testing, of the SUT.

5.3 Security Testing

Based on black box checking [PVY99], Shu and Lee had described an approach to learning-based security testing [SL07]. Instead of checking more general properties, they try to find violations of security properties in the composition of learned models of components. In following work, they presented a combination of learning and model-based fuzz testing and considered both active and passive model inference [SHL08]. This approach is more extensively described in [HSL08] with a focus on passive model inference. For this purpose they detail their state-merging-based inference approach, discuss the type of fuzz functions and the coverage criteria they used. Additionally, they provide a more exhaustive evaluation.

The compositional approach is also taken in [ORT$^+$07], where several methods are used to study the security of cryptographic protocols, where learning by testing black-box implementations is one of the techniques employed. The secrecy and authenticity properties are then checked on both the protocol specifications and the actual implementations through the learned model of the implementation.

Hossen et al. [HGOR14] presented an approach to model inference specifically tailored to security testing of web applications. The approach is based on the Z-quotient algorithm [PLG$^+$14].

Cho et al. [CBP$^+$11] developed a security testing tool called MACE. This tool combines the learning of a Mealy machine with concolic execution of the source code in order to explore the state space of protocol implementations more efficiently. Here, the learning algorithm guides the concolic execution in order to gain more control over the search process. When applied to four server applications, MACE could detect seven vulnerabilities.

5.4 Integration Testing

Tackling the issue that complex systems commonly integrate third-party components without specification, Li et al. [LGS06a] proposed a learning-based approach to integration testing. They follow an integrated approach in which they learn models of components from tests and based on the composition of these models, they generate integration tests. The execution of such tests may eventually lead to an update of the learned models if discrepancies are detected. Integration testing thus serves also as equivalence oracle. In following work, Li, Shahbaz and Groz [LGS06b,SLG07a,SLG07b] extended their learning-based integration testing approach to more expressive models. These models also account for data, through the introduction of parameters for actions and predicates over input parameters. Additionally, they also allow for observable non-determinism [SLG07a,SLG07b].

Groz et al. present an alternative approach to inference of component models [GLPS08]. Instead of learning each component model separately, they infer a *k-quotient* of the composed system and by projection they infer component models. With an initial model at hand, they perform a reachability analysis to detect compositional problems. If a detected problem can be confirmed, they warn that a problem exists, otherwise they refine the inferred models if the problem could not be confirmed. Testing is stopped when no potential compositional problem can be found.

In a similar setting as [LGS06a] and using the same algorithm, Shahbaz et al. [SPK07] described an approach to detect feature interaction in an integrated system. Basically, they infer models of components by testing, and execute the same tests of the composed system again. If the observations in the second phase do not conform to the inferred models, a feature interaction is detected.

Based on their previous works, Shahbaz and Groz [SG14] present an approach for analyzing and testing black-box components by combining model learning and MBT techniques. The procedure starts by learning each component's (partial) behavioral model and composing them as a product. The product is then fed to a model-based test case generator. The tests are then applied on the real system. Any discrepancies between the learned models and the system's real behavior counts as counterexample for the learned models, to be used to refine the models. For a more extensive discussion of learning-based integration testing, see also the corresponding chapter in the volume.

In [KMMV16] a test-based learning approach is devised, where an *already specified system* under test is executed to find and record deviations from that specification. Based on the collection of these deviations, a fault-model is learned, which is then used to perform model-based testing with QuickCheck [AHJW06] for the discovery of similar faults in other implementations. Being a preliminary work, it uses classic deterministic Mealy machines in the learning process with the LearnLib implementation. The models utilized in this approach are rich state-based models with full support for predicates. It falls into the integration testing category in that overall goal of the work is to test implementations composed of different versions of components, some of which may exhibit deviations from the reference model.

5.5 Regression Testing

Hagerer et al. [HHNS02] and Hungar et al. [HNS03] consider regression testing as a particularly fruitful application scenario for model learning. With the possibility of automatically maintaining models during the evolution of a system regression testing could be largely improved.

Regression testing and learning is also related in [LS14], however, in a slightly different fashion and not directly connected to model learning. Namely, machine-learning techniques are used to identify, select, and prioritize tests for regression testing based on test results from previous iterations and test meta-data.

Selection and extension of test cases, consequently leading to the refinement of the software model used for MBT, is also considered in [GS16]. Additional tests are recorded from the Exploratory Testing process [MSB11] and checked to be covered in the existing MBT model. If they are not, the model undergoes a refinement procedure to include the new execution traces. This can be classified as expert supported continuous learning process to build an MBT model.

5.6 Performance Testing

Adamis et al. proposed an approach to passively learn FSM models from conformance test logs to aid performance testing [AKR15]. Since the learned models may be inaccurate, manual postprocessing is required.

5.7 GUI Testing

Choi et al. described *Swifthand* a passive-learning-based testing tool for user interfaces of Android apps [CNS13]. They interleave learning and testing: (1)

they use the learned model to steer testing to previously unexplored states and (2) refine the model based on test observations. Their test selection strategy aims at minimizing the number of restarts, the most time-consuming action in the considered domain, while maximizing (code) coverage. The evaluation shows that *Swifthand* outperforms L^*-based and random testing.

6 Domain

Model learning and model-based testing has been applied to many different domains with different characteristics. In this section, we provide an overview of such application domains.

6.1 Embedded Systems

Embedded systems are a very suitable application domain for model learning and model-based testing; they often have a confined interaction with the environment through an interface. One of the earliest application of such techniques to the embedded system domain has been the application of model learning to telephone systems with large legacy subsystems [HHNS02,HNS03]. Meinke and Sindhu [MS11] applied their learning algorithm to a cruise control and an elevator controller.

Test-based learning (based on a variant of the well-known FSM-based testing, called the W-method) has been applied in [SMVJ15] to learn an industrial embedded control software.

The combination of learning and testing has also been applied in the automotive domain. In [KMMV16], the basic ideas about learning faulty behavior of AUTOSAR components is explored in order to predict possible failures in component integration. In [KMR] learning-based testing is applied to testing ECU applications.

6.2 Network and Security Protocols

Another application area often explored in the context of learning and testing is that of security protocols and protocol implementations. Using the abstraction technology described in [AHK$^+$12] and Mealy machines learned through LearnLib, [FBJV16] reports on learning different TCP stack implementations. Instead of for testing, the learned models are used for model checking to verify properties of these implementations in an off-line fashion. A similar case study carried out in a security setting focused on SSH implementations [FBLP$^+$17]. Model checking the learned models of different implementations revealed minor violations of the standard but no security-critical issues. In [MCWKK09], the learned protocols are used as an input for fuzzing tools in order to reveal security vulnerabilities. Learning-based fuzz testing has also been applied for the Microsoft MSN instant messaging protocol [SHL08,HSL08]. Furthermore, learning-based testing of security protocols is addressed in [SL07] as well.

The authors of [MCWKK09] learned a number of malware, text-based and binary protocols using some domain-specific and heuristic-based learning techniques. Aarts, Kuppens, Tretmans, Vaandrager and Verwer [AKT$^+$12,AKT$^+$14] combined various learning techniques to learn and test the bounded re-transmission protocol and Fiterau-Brostean, Janssen, Vaandrager [FBJV14] extended this work to fragments of TCP. Walkinshaw et al. [WBDP10] applied their inductive testing approach to explore the behavior of the Linux TCP stack.

Test-based learning has been extensively used to learn models of different sorts of smart-card based applications. Being black-box systems and typically specified using imprecise language, test-based learning helped to devise more precise models of such applications. In particular, the models of a biometric passport and a bank card have been produced this way, see [ASV10] and [AdRP13], respectively. In both works, a suitable data abstraction between the learning alphabet and the actual system inputs and output had to be developed to facilitate the learning process. This led to the development of Tomte, a framework for automated data abstraction for the purpose of real system learning [AHK$^+$12,Aar14]. The learned model produced [ASV10] was also compared to the manually developed model for the conformance testing of the Dutch biometric passport [MPS$^+$09].

6.3 Web Services

Raffelt et al. applied dynamic testing on web applications [RMSM08]. More concretely, they described a test environment *Webtest*, combining traditional testing methods, like record-and-replay, and dynamic testing. The latter provides benefits such as systematic exploration and model extrapolation, while the former eases dynamic testing by defining possible input actions.

Bertolino, Inverardi, Pelliccione, and Tivoli [BIPT09] used test-based learning (based on finite state machines) to learn the behavioral interfaces for web services.

6.4 Biological Systems

Biological systems have been recently studied as instances of reactive systems [BFFK09]. This provides the prospect of using models of reactive and hybrid systems to replace in vivo and in vitro experiments on living organisms and cells with in silico experiments (e.g., replacing the experiments with model checking or model-based testing) [BFFH14,Col14]. In [AL13], test-based learning is used to learn hybrid automata models of biological systems (cell models). In [MHR$^+$06], automata learning technique is integrated with requirement-driven engineering to create and improve models of biological systems.

7 Conclusions

Learning-based testing is an active research area that has produced impressive results despite being a relatively young discipline. Different systems in various critical domains have been tested successfully including controllers, communication protocols, web applications, mobile apps and smart cards. Every year new algorithms, techniques and tools are proposed in order to learn and test increasingly complex systems.

The prevailing concern in the domain of model-learning (in the context of testing) is the scalability and applicability to real systems. For such applications, abstraction techniques for input and output data are needed to support the learning process. The researchers are actively looking into automating this process, which in many cases is still manual and requires either domain-specific knowledge, or apriori knowledge about the system under test. Several discussed papers either mention this as an issue, or provide some solution for it.

Another open issue surfacing in the described works is the treatment of richer models, both in the context of learning and testing. For example, stochastic models, or models that consider time or system dynamics. Such rich models bring new challenges in both research domains, moreover, they underline the scalability issues mentioned above.

Completeness (or a quantified approximation thereof) is another major concern in this domain. A property of algorithms in the MAT framework is "that a learned model is either complete and correct, or not correct at all" [VT15]. Note that in this context, correctness expresses that the learned model and the system under learning agree on all possible inputs. In [VT15], this property has been dropped by learning an over- and an underapproximation and preserving ioco-conformance during learning. In other words, there are two learned models which may not agree with the system under learning on all inputs but which are in a conformance relation with the system. However, such an adaptation may not be possible for all types of models. Steffen et al. [SHM11] also mention this property, stating that it must be accepted and that incompletely learned models may still provide benefits in certain scenarios, e.g., for test-case generation [HHNS02].

Scenarios like black-box checking [PVY99] on the other hand suffer from incompleteness[4]. They can guarantee that a verified property either holds or the number of states of the system is larger than an assumed upper bound. More quantitative measures of correctness would be useful for this type of verification such that, e.g., statistical guarantees could be given with a certain confidence. Although already early work discussed such matters, there has not been much research in this direction. In fact, Angluin considered learning without equivalence queries in a stochastic setting in her seminal paper [Ang87]. Furthermore, Rivest & Schapire also gave probabilities for learning the correct model [RS93]. Despite its practical usefulness, recent work usually does not assign probabilities or confidence levels to the learning result, also in case stochastic (testing) strategies are applied.

Testing has always been a challenge due to (1) its incompleteness by nature, (2) the lack of good specifications and (3) by its high demand for resources. With the growing complexity of the systems-under-tests this process is not going to be easier. Learning-based testing offers an opportunity to master this complexity

[4] The authors also briefly discuss stochastic properties of Mealy machines, though.

with modern learning-based techniques. It represents a natural evolution of testing: with the trend of our environment becoming "smarter", e.g. smart homes, smart cars, smart production, smart energy, our testing process needs to be smart as well. We are seeing the advent of smart testing.

Acknowledgments.

The insightful comments of Karl Meinke and Neil Walkinshaw on an earlier draft led to improvements and are gratefully acknowledged.

The work of B. K. Aichernig and M. Tappler was supported by the TU Graz LEAD project "Dependable Internet of Things in Adverse Environments". The work of M. R. Mousavi and M. Taromirad has been partially supported by the Swedish Research Council (Vetenskapsradet) award number: 621-2014-5057 (Effective Model-Based Testing of Concurrent Systems) and the Strategic Research Environment ELLIIT. The work of M. R. Mousavi has also been partially supported by the Swedish Knowledge Foundation (Stiftelsen for Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS HöG project (number: 20140312).

References

Aar14. Fides Aarts. *Tomte: bridging the gap between active learning and real-world systems.* PhD thesis, Department of Computer Science, 2014.

AdRP13. Fides Aarts, Joeri de Ruiter, and Erik Poll. Formal models of bank cards for free. In *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '13, pages 461–468, Washington, DC, USA, 2013. IEEE Computer Society.

AFBKV15. Fides Aarts, Paul Fiterău-Broştean, Harco Kuppens, and Frits W. Vaandrager. Learning register automata with fresh value generation. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 165–183. Springer, 2015.

AHJW06. Thomas Arts, John Hughes, Joakim Johansson, and Ulf T. Wiger. Testing telecoms software with QuviQ QuickCheck. In Marc Feeley and Philip W. Trinder, editors, *Proceedings of the 2006 ACM SIGPLAN Workshop on Erlang, Portland, Oregon, USA, September 16, 2006*, pages 2–10. ACM, 2006.

AHK$^+$12. Fides Aarts, Faranak Heidarian, Harco Kuppens, Petur Olsen, and Frits W. Vaandrager. Automata learning through counterexample guided abstraction refinement. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods: 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, pages 10–27, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

AKR15. Gusztáv Adamis, Gábor Kovács, and György Réthy. Generating performance test model from conformance test logs. In Joachim Fischer, Markus Scheidgen, Ina Schieferdecker, and Rick Reed, editors, *SDL 2015:*

Model-Driven Engineering for Smart Cities - 17th International SDL Forum, Berlin, Germany, October 12-14, 2015, Proceedings, volume 9369 of *Lecture Notes in Computer Science*, pages 268–284. Springer, 2015.

AKT+12. Fides Aarts, Harco Kuppens, Jan Tretmans, Frits W. Vaandrager, and Sicco Verwer. Learning and testing the bounded retransmission protocol. In Jeffrey Heinz, Colin de la Higuera, and Tim Oates, editors, *Proceedings of the Eleventh International Conference on Grammatical Inference, ICGI 2012, University of Maryland, College Park, USA, September 5-8, 2012*, volume 21 of *JMLR Proceedings*, pages 4–18. JMLR.org, 2012.

AKT+14. Fides Aarts, Harco Kuppens, Jan Tretmans, Frits W. Vaandrager, and Sicco Verwer. Improving active mealy machine learning for protocol conformance testing. *Machine Learning*, 96(1-2):189–224, 2014.

AL13. Rasmus Ansin and Didrik Lundberg. Automated inference of excitable cell models as hybrid automata, 2013. Bachelor Thesis, School of Computer Science and Communication, KTH Stockholm.

Alp14. Ethem Alpaydin. *Introduction to Machine Learning, Third Edition*. MIT Press, 2014.

Ang87. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, November 1987.

ARM16. Arend Aerts, Michel A. Reniers, and Mohammad Reza Mousavi. Model-based testing of cyber-physical systems. In Houbing Song, Danda B. Rawat, Sabina Jeschke, and Christian Brecher, editors, *Cyber-Physical Systems Foundations, Principles and Applications*, chapter 19, pages 287–304. Elsevier, 2016.

ASJ+16. George Argyros, Ioannis Stais, Suman Jana, Angelos D. Keromytis, and Aggelos Kiayias. SFADiff: Automated evasion attacks and fingerprinting using black-box differential automata learning. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1690–1701. ACM, 2016.

ASV10. Fides Aarts, Julien Schmaltz, and Frits W. Vaandrager. Inference and abstraction of the biometric passport. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I*, pages 673–686, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

AT10. Thomas Arts and Simon Thompson. From test cases to FSMs: augmented test-driven development and property inference. In *Proceedings of the 9th ACM SIGPLAN workshop on Erlang*, Erlang '10, 2010.

AT17a. Bernhard K. Aichernig and Martin Tappler. Learning from faults: Mutation testing in active automata learning - mutation testing in active automata learning. In Clark Barrett, Misty Davies, and Temesghen Kahsai, editors, *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, volume 10227 of *Lecture Notes in Computer Science*, pages 19–34, 2017.

AT17b. Bernhard K. Aichernig and Martin Tappler. Probabilistic black-box reachability checking. In *Runtime Verification - 17th International Conference, RV 2017, Seattle, USA, September 13-16, 2017, Proceedings*, 2017. In press.

BFFH14. Nicola Bonzanni, K. Anton Feenstra, Wan Fokkink, and Jaap Heringa. Petri nets are a biologist's best friend. In Francois Fages and Carla Piazza, editors, *Proceedings of th First International Conference on Formal Methods in Macro-Biology (FMMB 2014)*, volume 8738 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 2014.

BFFK09. Nicola Bonzanni, K. Anton Feenstra, Wan Fokkink, and Elzbieta Krepska. What can formal methods bring to systems biology? In Cavalcanti and Dams [CD09], pages 16–22.

BG96. Francesco Bergadano and Daniele Gunetti. Testing by means of inductive program learning. *ACM Trans. Softw. Eng. Methodol.*, 5(2):119–145, 1996.

BGJ$^+$05. Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3442 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2005.

BI11. Marco Bernardo and Valérie Issarny, editors. *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, volume 6659 of *Lecture Notes in Computer Science*. Springer, 2011.

BIPT09. Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Automatic synthesis of behavior protocols for composable webservices. In Hans van Vliet and Valérie Issarny, editors, *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*, pages 141–150. ACM, 2009.

CBP$^+$11. Chia Yuan Cho, Domagoj Babić, Pongsin Poosankam, Kevin Zhijie Chen, Edward XueJun Wu, and Dawn Song. MACE: model-inference-assisted concolic exploration for protocol and vulnerability discovery. In *Proceedings of the 20th USENIX conference on Security*. USENIX Association, 2011.

CD09. Ana Cavalcanti and Dennis Dams, editors. *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*. Springer, 2009.

CdlHJ09. David Combe, Colin de la Higuera, and Jean-Christophe Janodet. Zulu: An interactive learning competition. In Anssi Yli-Jyrä, András Kornai, Jacques Sakarovitch, and Bruce W. Watson, editors, *Finite-State Methods and Natural Language Processing, 8th International Workshop, FSMNLP 2009, Pretoria, South Africa, July 21-24, 2009, Revised Selected Papers*, volume 6062 of *Lecture Notes in Computer Science*, pages 139–146. Springer, 2009.

CHJS14. Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Learning extended finite state machines. In Dimitra Giannakopoulou and Gwen Salaün, editors, *Software Engineering and Formal Methods: 12th Interna-*

tional Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings, pages 250–264, Cham, 2014. Springer International Publishing.

CHJS16. Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Aspects of Computing*, 28(2):233–263, 2016.

Cho78. T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, 4(3):178–187, May 1978.

CNS13. Wontae Choi, George C. Necula, and Koushik Sen. Guided GUI testing of android apps with minimal restart and approximate learning. In Antony L. Hosking, Patrick Th. Eugster, and Cristina V. Lopes, editors, *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 623–640. ACM, 2013.

CO94. Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and José Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 1994.

Col14. Pieter Collins. Model-checking in systems biology - from micro to macro. In Francois Fages and Carla Piazza, editors, *Proceedings of the First International Conference on Formal Methods in Macro-Biology (FMMB 2014)*, volume 8738 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2014.

CSY99. Cezar Câmpeanu, Nicolae Sântean, and Sheng Yu. Minimal cover-automata for finite languages. In *Automata Implementation: Third International Workshop on Implementing Automata, WIA'98, Revised Papers*, volume 1660, pages 43–56. Springer Berlin Heidelberg, 1999.

DIMS12. Ionut Dinca, Florentin Ipate, Laurentiu Mierla, and Alin Stefanescu. Learn and test for Event-B – A Rodin plugin. In John Derrick, John S. Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, volume 7316 of *Lecture Notes in Computer Science*, pages 361–364. Springer, 2012.

DIS12. Ionut Dinca, Florentin Ipate, and Alin Stefanescu. Model learning and test generation for Event-B decomposition. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I*, volume 7609 of *Lecture Notes in Computer Science*, pages 539–553. Springer, 2012.

DLDvL08. Pierre Dupont, Bernard Lambeau, Christophe Damas, and Axel van Lamsweerde. The QSM algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22(1-2):77–115, 2008.

dRP15. Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Secu-*

rity Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015., pages 193–206. USENIX Association, 2015.

EGPQ06. Edith Elkind, Blaise Genest, Doron A. Peled, and Hongyang Qu. Grey-box checking. In Najm et al. [NPD06], pages 420–435.

FBJV14. Paul Fiterău-Broştean, Ramon Janssen, and Frits W. Vaandrager. Learning fragments of the TCP network protocol. In Frédéric Lang and Francesco Flammini, editors, *Formal Methods for Industrial Critical Systems - 19th International Conference, FMICS 2014, Florence, Italy, September 11-12, 2014. Proceedings*, volume 8718 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2014.

FBJV16. Paul Fiterău-Broştean, Ramon Janssen, and Frits W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 454–471, Cham, 2016. Springer International Publishing.

FBLP$^+$17. Paul Fiterău-Broştean, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits W. Vaandrager, and Patrick Verleg. Model learning and model checking of SSH implementations. In *Proceedings 24th International SPIN Symposium on Model Checking of Software, 13-14 July 2017, Santa Barbara, California*, 2017. In press.

FvBK$^+$91. Susumu Fujiwara, Gregor von Bochmann, Ferhat Khendek, Mokhtar Amalou, and Abderrazak Ghedamsi. Test selection based on finite state models. *IEEE Trans. Softw. Eng.*, 17(6):591–603, June 1991.

GLPS08. Roland Groz, Keqin Li, Alexandre Petrenko, and Muzammil Shahbaz. Modular system verification by inference, testing and reachability analysis. In Kenji Suzuki, Teruo Higashino, Andreas Ulrich, and Toru Hasegawa, editors, *Testing of Software and Communicating Systems, 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008, 8th International Workshop, FATES 2008, Tokyo, Japan, June 10-13, 2008, Proceedings*, volume 5047 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 2008.

GPY02a. Alex Groce, Doron A. Peled, and Mihalis Yannakakis. Adaptive model checking. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2002.

GPY02b. Alex Groce, Doron A. Peled, and Mihalis Yannakakis. AMC: an adaptive model checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 521–525. Springer, 2002.

GS16. Ceren Şahin Gebizli and Hasan Sözer. Automated refinement of models for model-based testing using exploratory testing. *Software Quality Journal*, pages 1–27, 2016.

HGOR14. Karim Hossen, Roland Groz, Catherine Oriat, and Jean-Luc Richier. Automatic model inference of web applications for security testing. In *Seventh IEEE International Conference on Software Testing, Verification* *and Validation, ICST 2014 Workshops Proceedings, March 31 - April 4, 2014, Cleveland, Ohio, USA*, pages 22–23. IEEE Computer Society, 2014.

HHNS02. Andreas Hagerer, Hardi Hungar, Oliver Niese, and Bernhard Steffen. Model generation by moderated regular extrapolation. In *International Conference on Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, pages 80–95. Springer, 2002.

HK08. Pham Ngoc Hung and Takuya Katayama. Modular conformance testing and assume-guarantee verification for evolving component-based software. In *15th Asia-Pacific Software Engineering Conference (APSEC 2008), 3-5 December 2008, Beijing, China*, pages 479–486. IEEE Computer Society, 2008.

HNS03. Hardi Hungar, Oliver Niese, and Bernhard Steffen. Domain-specific optimization in automata learning. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2003.

HRD07. Johannes Henkel, Christoph Reichenbach, and Amer Diwan. Discovering documentation for Java container classes. *IEEE Trans. Software Eng.*, 33(8):526–543, 2007.

HSL08. Yating Hsu, Guoqiang Shu, and David Lee. A model-based approach to security flaw detection of network protocol implementations. In *Proceedings of the 16th annual IEEE International Conference on Network Protocols, 2008. ICNP 2008, Orlando, Florida, USA, 19-22 October 2008*, pages 114–123. IEEE Computer Society, 2008.

HSM10. Falk Howar, Bernhard Steffen, and Maik Merten. From ZULU to RERS - lessons learned in the ZULU challenge. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I*, volume 6415 of *Lecture Notes in Computer Science*, pages 687–704. Springer, 2010.

HSM11. Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2011.

IHS15. Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source LearnLib - A framework for active automata learning. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015.

ISD15. Florentin Ipate, Alin Stefanescu, and Ionut Dinca. Model learning and test generation using cover automata. *Comput. J.*, 58(5):1140–1159, 2015.

KMMV16. Sebastian Kunze, Wojciech Mostowski, Mohammad Reza Mousavi, and Mahsa Varshosaz. Generation of failure models through automata learning. In *Workshop on Automotive Systems/Software Architectures (WASA'16)*, pages 22–25. IEEE Computer Society, April 2016.

KMR. Hojat Khosrowjerdi, Karl Meinke, and Andreas Rasmusson. Automated behavioral requirements testing for automotive ECU applications. Submitted, 2016.

KV94. Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, USA, 1994.

LCJ06. Zhifeng Lai, S. C. Cheung, and Yunfei Jiang. Dynamic model learning using genetic algorithm under adaptive model checking framework. In *Sixth International Conference on Quality Software (QSIC 2006), 26-28 October 2006, Beijing, China*, pages 410–417. IEEE Computer Society, 2006.

LGS06a. Keqin Li, Roland Groz, and Muzammil Shahbaz. Integration testing of components guided by incremental state machine learning. In Phil McMinn, editor, *Testing: Academia and Industry Conference - Practice And Research Techniques (TAIC PART 2006), 29-31 August 2006, Windsor, United Kingdom*, pages 59–70. IEEE Computer Society, 2006.

LGS06b. Keqin Li, Roland Groz, and Muzammil Shahbaz. Integration testing of distributed components based on learning parameterized I/O models. In Najm et al. [NPD06], pages 436–450.

LS14. Remo Lachmann and Ina Schaefer. Towards efficient and effective testing in automotive software development. In Erhard Plödereder, Lars Grunske, Eric Schneider, and Dominik Ull, editors, *44. Jahrestagung der Gesellschaft für Informatik, Informatik 2014, Big Data - Komplexität meistern, 22.-26. September 2014 in Stuttgart, Deutschland*, volume 232 of *Lecture Notes in Informatics*, pages 2181–2192. GI, 2014.

LY94. David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Trans. Computers*, 43(3):306–320, 1994.

MCWKK09. Paolo Milani Comparetti, Gilbert Wondracek, Christopher Krügel, and Engin Kirda. Prospex: Protocol specification extraction. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 110–125. IEEE Computer Society, 2009.

Mei04. Karl Meinke. Automated black-box testing of functional correctness using function approximation. *SIGSOFT Softw. Eng. Notes*, 29(4):143–153, July 2004.

MHR$^+$06. Tiziana Margaria, Michael G. Hinchey, Harald Raffelt, James L. Rash, Christopher A. Rouff, and Bernhard Steffen. Completing and adapting models of biological processes. In Yi Pan, Franz J. Rammig, Hartmut Schmeck, and Mauricio Solar, editors, *Proceedings of the IFIP 19th World Computer Congress on Biologically Inspired Cooperative Computing*, pages 43–54, Boston, MA, 2006. Springer US.

Mit97. Tom M. Mitchel. *Machine Learning.* McGraw Hill, 1997.

MN10. Karl Meinke and Fei Niu. A learning-based approach to unit testing of numerical software. In Alexandre Petrenko, Adenilso da Silva Simão, and José Carlos Maldonado, editors, *Testing Software and Systems - 22nd IFIP WG 6.1 International Conference, ICTSS 2010, Natal, Brazil, November 8-10, 2010. Proceedings*, volume 6435 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 2010.

MN15. Karl Meinke and Peter Nycander. Learning-based testing of distributed microservice architectures: Correctness and fault injection. In Domenico Bianculli, Radu Calinescu, and Bernhard Rumpe, editors, *Software Engineering and Formal Methods - SEFM 2015 Collocated Workshops: ATSE,* *HOFM, MoKMaSD, and VERY*SCART, York, UK, September 7-8, 2015, Revised Selected Papers*, volume 9509 of *Lecture Notes in Computer Science*, pages 3–10. Springer, 2015.

MNRS04. Tiziana Margaria, Oliver Niese, Harald Raffelt, and Bernhard Steffen. Efficient test-based model generation for legacy reactive systems. In *High-Level Design Validation and Test Workshop, 2004. Ninth IEEE International*, pages 95–100. IEEE, 2004.

MPS+09. Wojciech Mostowski, Erik Poll, Julien Schmaltz, Jan Tretmans, and Ronny Wichers Schreur. Model-based testing of electronic passports. In María Alpuente, Byron Cook, and Christophe Joubert, editors, *Formal Methods for Industrial Critical Systems 2009, Proceedings*, volume 5825 of *Lecture Notes in Computer Science*, pages 207–209. Springer, November 2009.

MS11. Karl Meinke and Muddassar A. Sindhu. Incremental learning-based testing for reactive systems. In Martin Gogolla and Burkhart Wolff, editors, *Tests and Proofs - 5th International Conference, TAP 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings*, volume 6706 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2011.

MSB11. Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011.

Nie03. Oliver Niese. *An integrated approach to testing complex systems*. PhD thesis, Dortmund University of Technology, 2003.

NPD06. Elie Najm, Jean-François Pradat-Peyre, and Véronique Donzeau-Gouge, editors. *Formal Techniques for Networked and Distributed Systems - FORTE 2006, 26th IFIP WG 6.1 International Conference, Paris, France, September 26-29, 2006*, volume 4229 of *Lecture Notes in Computer Science*. Springer, 2006.

OG92. Jose Oncina and Pedro Garcia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition. Volume 5 of Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

ORT+07. Martijn Oostdijk, Vlad Rusu, Jan Tretmans, R. G. de Vries, and T. A. C. Willemse. Integrating verification, testing, and learning for cryptographic protocols. In Jim Davies and Jeremy Gibbons, editors, *Integrated Formal Methods: 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007. Proceedings*, pages 538–557, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

PLG+14. Alexandre Petrenko, Keqin Li, Roland Groz, Karim Hossen, and Catherine Oriat. Inferring approximated models for systems engineering. In *15th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2014, Miami Beach, FL, USA, January 9-11, 2014*, pages 249–253. IEEE Computer Society, 2014.

PVY99. Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China*, volume 156 of *IFIP Conference Proceedings*, pages 225–240. Kluwer, 1999.

PW15. Petros Papadopoulos and Neil Walkinshaw. Black-box test generation from inferred models. In Rachel Harrison, Ayse Basar Bener, and Burak Turhan, editors, *4th IEEE/ACM International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2015, Florence, Italy, May 17, 2015*, pages 19–24. IEEE Computer Society, 2015.

RMSM08. Harald Raffelt, Tiziana Margaria, Bernhard Steffen, and Maik Merten. Hybrid test of web applications with Webtest. In Tevfik Bultan and Tao Xie, editors, *Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), TAV-WEB 2008, Seattle, Washington, USA, July 21, 2008*, pages 1–7. ACM, 2008.

RMSM09. Harald Raffelt, Maik Merten, Bernhard Steffen, and Tiziana Margaria. Dynamic testing via automata learning. *STTT*, 11(4):307–324, 2009.

RS93. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.

SAP$^+$17. Suphannee Sivakorn, George Argyros, Kexin Pei, Angelos D. Keromytis, and Suman Jana. HVLearn: Automated black-box analysis of hostname verification in SSL/TLS implementations. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 521–538. IEEE Computer Society, 2017.

SG09. Muzammil Shahbaz and Roland Groz. Inferring Mealy machines. In Cavalcanti and Dams [CD09], pages 207–222.

SG14. Muzammil Shahbaz and Roland Groz. Analysis and testing of black-box component-based systems by inferring partial models. *Software Testing, Verification, and Reliability*, 24(4):253–288, 2014.

SHL08. Guoqiang Shu, Yating Hsu, and David Lee. Detecting communication protocol security flaws by formal fuzz testing and machine learning. In Kenji Suzuki, Teruo Higashino, Keiichi Yasumoto, and Khaled El-Fakih, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2008, 28th IFIP WG 6.1 International Conference, Tokyo, Japan, June 10-13, 2008, Proceedings*, volume 5048 of *Lecture Notes in Computer Science*, pages 299–304. Springer, 2008.

SHM11. Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In Bernardo and Issarny [BI11], pages 256–296.

SL07. Guoqiang Shu and David Lee. Testing security properties of protocol implementations - a machine learning based approach. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*, page 25. IEEE Computer Society, 2007.

SLBW15. Christoph Schulze, Mikael Lindvall, Sigurthor Bjorgvinsson, and Robert Wiegand. Model generation to support model-based testing applied on the NASA DAT web-application - an experience report. In *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*, pages 77–87. IEEE Computer Society, 2015.

SLG07a. Muzammil Shahbaz, Keqin Li, and Roland Groz. Learning and integration of parameterized components through testing. In Alexandre Petrenko, Margus Veanes, Jan Tretmans, and Wolfgang Grieskamp, editors, *Testing of Software and Communicating Systems, 19th IFIP TC6/WG6.1 International Conference, TestCom 2007, 7th International Workshop, FATES 2007, Tallinn, Estonia, June 26-29, 2007, Proceedings*, volume 4581 of *Lecture Notes in Computer Science*, pages 319–334. Springer, 2007.

SLG07b. Muzammil Shahbaz, Keqin Li, and Roland Groz. Learning parameterized state machine model for integration testing. In *31st Annual International Computer Software and Applications Conference, COMPSAC 2007, Beijing, China, July 24-27, 2007. Volume 2*, pages 755–760. IEEE Computer Society, 2007.

SMVJ15. Wouter Smeenk, Joshua Moerman, Frits W. Vaandrager, and David N. Jansen. Applying automata learning to embedded control software. In Michael Butler, Sylvain Conchon, and Fatiha Zaïdi, editors, *Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings*, volume 9407 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2015.

SPK07. Muzammil Shahbaz, Benoît Parreaux, and Francis Klay. Model inference approach for detecting feature interactions in integrated systems. In Lydie du Bousquet and Jean-Luc Richier, editors, *Feature Interactions in Software and Communication Systems IX, International Co nference on Feature Interactions in Software and Communication Systems, ICFI 2007, 3-5 September 2007, Grenoble, France*, pages 161–171. IOS Press, 2007.

TAB17. Martin Tappler, Bernhard K. Aichernig, and Roderick Bloem. Model-based testing iot communication via active automata learning. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 276–287, 2017.

TB03. Jan Tretmans and Ed Brinksma. TorX: Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, editors, *First European Conference on Model-Driven Software Engineering*, pages 31–43, December 2003.

Tre11. Jan Tretmans. Model-based testing and some steps towards test-based modelling. In Bernardo and Issarny [BI11], pages 297–326.

UL07. Mark Utting and Bruno Legeard. *Practical Model-Based Testing - A Tools Approach.* Morgan Kaufmann, 2007.

UPL12. Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, August 2012.

Vas73. M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetics*, 9(4):653–665, 1973.

VT14. Michele Volpato and Jan Tretmans. Active learning of nondeterministic systems from an IOCO perspective. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part I*, volume 8802 of *Lecture Notes in Computer Science*, pages 220–235. Springer, 2014.

VT15. Michele Volpato and Jan Tretmans. Approximate active learning of nondeterministic input output transition systems. *ECEASST*, 72, 2015.

WBDP10. Neil Walkinshaw, Kirill Bogdanov, John Derrick, and Javier Paris. Increasing functional coverage by inductive testing: A case study. In Alexandre Petrenko, Adenilso Simão, and José Carlos Maldonado, editors, *Testing Software and Systems: 22nd IFIP WG 6.1 International Conference, ICTSS 2010, Natal, Brazil, November 8-10, 2010. Proceedings*, pages 126–141, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

WBHS07. Neil Walkinshaw, Kirill Bogdanov, Mike Holcombe, and Sarah Salahuddin. Reverse engineering state machines by interactive grammar inference. In *14th Working Conference on Reverse Engineering (WCRE 2007), 28-31 October 2007, Vancouver, BC, Canada*, pages 209–218. IEEE Computer Society, 2007.

WDG09. Neil Walkinshaw, John Derrick, and Qiang Guo. Iterative refinement of reverse-engineered models by model-based testing. In Cavalcanti and Dams [CD09], pages 305–320.

Wey83. Elaine J. Weyuker. Assessing test data adequacy through program inference. *ACM Trans. Program. Lang. Syst.*, 5(4):641–655, 1983.

WF17. Neil Walkinshaw and Gordon Fraser. Uncertainty-driven black-box test data generation. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 253–263, 2017.

YCM09. Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 183–192. ACM, 2009.