Project no.: **ICT-FP7-STREP-214755**

Project full title: **Quantitative System Properties in Model-Driven Design**

Project Acronym: **QUASIMODO**

Deliverable no.: D4.4

Title of Deliverable: Approximate Testing

Contractual Date of Delivery to the CEC:	**Month 36**
Actual Date of Delivery to the CEC:	**Month 40 (30. April 2011)**
Organisation name of lead contractor for this deliverable:	**P05 Saarland University**
Author(s):	**Hernán Baró Graf, Holger Hermanns, Brian Nielsen**
Participants(s):	**P01, P02, P05**
Work package contributing to the deliverable:	**WP4**
Nature:	**R**
Version:	**1.0**
Total number of pages:	**20**
Start date of project:	1 Jan. 2008 **Duration:** 36 month

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable surveys the results achieved in the area of aproximate testing, produced within the QUASIMODO project.

Keyword list: Approximation, Learning, Testing, Models.

Contents

Abbreviations

AAU: Aalborg University, DK (P01)

ESI/RU: Radboud University Nijmegen, NL (P02)

ESI/UT: University Twente, Enschede, NL (under the auspices of P02)

1 Introduction

Software Testing is an important and indispensable part of the software development process. Model-Based Testing (MBT) is an approach to formalising and mechanising the often practised ad hoc (or non-existing) manual methodology when testing a software artefact. In MBT an implementation under test is tested for compliance with a model that describes the required behaviour of the implementation. A prominent model-based testing approach is rooted in the *input/output conformance* (ioco) testing theory, where models are expressed as labelled transition systems, and compliance is defined with the ioco implementation relation [25]. This provides a sound foundation for labelled transition system testing, and has proved to be a practical basis for several model-based test generation tools and applications. This technique makes it possible to generate and execute many more tests than would ever be possible manually.

However, MBT (whether or not based on ioco) still has deficiencies both in theory and in practice. An important aspect of testing is that it is inherently incomplete, since infinitely many tests were required to show the absence of software flaws. Thus, any testing methodology must be approximative, if taking absolute correctness as a yardstick. But even in cases looser interpretation of correctness is acceptable, there are several aspects of MBT that have driven the Quasimodo consortium to study notions of approximative testing.

Uncertainty in timed testing. Testing embedded real-time systems is a very demanding endeavour. Because these systems are working in a continuous environment – with respect to time and other continuous quantities (temperature, pressure etc). These quantities are observed through possibly imprecise sensors, and discretized by them. The MBT testing machinery, interacts with the system under test (SUT) on the basis of such discretized values. At the same time, it must evaluate the timeliness of the SUT .

To account for these complications, the testing process needs to be made adaptive to uncertainty in the implementation's responses, but in a sound and effective manner. Section 2 describes a variety of techniques developed by Quasimodo and incorporated in the UPPAAL-TRON testing environment. They indeed approximate the uncertainty and imprecision properly and effectively.

Asyncrony in interaction between tester and SUT . Another source of approximativity in the ioco-approach to conformance testing is rooted in the assumption that communication between the testing tool and the SUT is synchronous and stepwise. In real applications, we often find situations where synchronicity can only be approximated by a asynchronous communications, and where the SUT may send message bursts.

In the testing tool TorX the handling of these aspects is the task of the Adaptor. But although this is a crucial component, there is no systematic method of obtaining one. This means that for each SUT one would need to develop a new Adaptor to be able to communicate with it. Section 3 reports on efforts launched by Quasimodo to to create a more generic Adaptor and thereby making it easier to support communication with a new SUT in the future. This is done in the context of a LEGO bricks sorting machine.

Approximative learning of specifications via testing. MBT relies on the existence of a model of the SUT . Since this model is not always available (especially for legacy software), Quasimodo has embarked on concerted efforts to turn the MBT testing technology into an automata learning technology. The original approaches to automata learning relies – as a crucial component – on a component that is able to decide language equivalence queries.

This is approximated in the automata learning approach described in Section 4 by an MBT testing tool, that feeds the SUT with long test sequences, thereby approximating the original question. The approach has several additional interesting aspects, and appears very promising as a whole.

Section 5 describes the interplay of model-based testing and learning, and how their combined use leads to an 'agile' process of testing and learning, where the correctness of both the model and the SUT is approximated in cyclic, iterative steps.

Moreover, it is shown that the question of the level of approximation in learning, i.e., how precise is the learned model, is exactly the same as the level of completeness of model-based testing, i.e., test coverage.

Approximative learning of probabilistic specifications. The MBT technology can also be used to approximatively learn probabilistic specifications. For complex systems that are only partially observable via their interactions with the user, it might be unrealistic to assume that an adequate deterministic model exists.

Section 6 reports on efforts to learn probabilistic quantitative models of a system's observable (and possibly non-deterministic) behaviour. The data we require for learning only consists of previously observed system behaviours, that is obtainable through active or passive testing.

2 Timed Testing under Uncertainty

Participants: Alexandre David, Kim G. Larsen, Shuhao Li,
Marius Mikucionis and Brian Nielsen (AAU).

Context Testing embedded real-time systems is challenging because it must deal with timing, processing and computation of complex mixed discrete and continuous signals, limited observation and control, and concurrency.

First, testing must evaluate the timeliness of the system under test (SUT), i.e., the tester must execute input stimuli with different timings and loads, and evaluate its response times against the specified tolerances. Second, the system performs complex computations on continuous signals that are observed through (potentially imprecise) sensors, sampled and discretised, implying that the resulting internal state and output values are not completely fixed, but should be accurate within some required bound. Third, because the system is embedded it, is often problematic to create a test harness that gives a tester full observability and controllability of the internal state of the system. Some times testing has to be carried out via imprecise sensors and actuators. Finally, embedded systems are inherently highly concurrent and indeterminate because they need to control multiple simultaneous activities and detect and react to a multitude of events. This implies that executing the same input sequence may result it different output sequences, and that test cases in general need to branch out and be adaptive to uncertainty in the system's responses.

Contribution An important and powerful technique to deal with such imprecision and uncertainty in (formal) specification models is to use non-determinism to create an over approximation that captures the permissive variability.

Formal specification models therefore tend to use non-determinism in many ways:

- models must be able to capture the concurrent behaviour of real system implementations.
- models of large and complex systems are themselves constructed as parallel composition of models of its components.
- non-determinism is used in models to abstract away implementation details or aspects that are unknown or are too complex to fully model, e.g. details of data-representation and computation algorithms, or the state of pipelines, caches, bus-arbitration, and internal scheduling and resource allocation decisions.
- models should not capture one concrete implementation in all details, but specify its requirements, i.e. the permissible set of implementations. In particular with respect to timing, a model should capture requirements like the system should respond *within* x milliseconds, not that it responds after precisely x milliseconds.

- the systems makes imperfect observations and controls of its environment because of limitations on the precision with which it measures/produces physical quantities (especially time), and the states/actions that are observable/controllable to the system in the first place. These limitations also applies to a tester (itself being a computer component) of the system.

We have contributed to testing of such systems and system models with uncertainty in the following ways:

Timed Testing Under Uncertainty: In [11] we propose and compare a number of model-based test generation techniques that aims at enabling efficient testing of timed systems under uncertainty (in particular timing uncertainty) and that are realisable using existing state-exploration techniques and tools: 1) generate preset input sequences, and check the resulting trace for inclusion in the model post-mortem. 2) use online testing using e.g, UPPAAL-TRON , 3) formulate test generation as a timed game problem and use the UPPAAL-TIGA timed game-solver, and 4) an important variation of the timed game approach, but using partial observability. The approaches are described in detail and presented in the same formal framework.

Time over approximation in UPPAAL-TRON **:** In UPPAAL-TRON we have also created a mechanisms for automatically handling uncertainty on the time occurrence of an event. The UPPAAL-TRON manual documents (Section 4.3 of [18]) a carefully thought-out mechanism for mapping physical time into model-time units. Rather than simply mapping to the nearest model-time unit (which could lead to erroneous verdicts), the mechanism maps it into an *interval* of model-time units. When UPPAAL-TRON computes the permissible sets of states of the model (See Deliverable D4.2) reachable after this event, this interval is taken into consideration, such that the state-set reflects all those states that can be reached within that interval. The user may further enlarge this by providing a lower and upper bound on the delay through the adaptor (including the observation uncertainty). The over-approximation applies both in the input and output direction.

Value over approximation in UPPAAL-TRON **:** The modelling pattern proposed to track continuous variables in Deliverable D4.6 (Section 2.2.2) can also be used to capture the uncertainty of a specific output value. The idea is that the model maintains separate variables for the upper and lower bound for the value in question, and have the adaptor update these when it (approximately) observes that the SUT has produced a new value.

Perspectives The Quasimodo work on UPPAAL-TRON has brought a multitude of approximation techniques. These make it possible to deal with imprecision and uncertainty that arise from discretization in the value and the time domain of (formal) specification models relative to the continuous reality.

3 ToLERo: TorX-Tested Lego Robots

Participants: Axel Belinfante, Hasan Sözer, Arjan Snippe, and Mariëlle Stoelinga (ESI/UT).

Context Model-based software testing is done by generating test-cases from a model that represents the desired behaviour of a System Under Test (SUT). Then the tool runs those tests and compares the results with the actual SUT. One of the issues here is the synchronisation of the communications between the tool and the SUT. In the testing tool TorX this is the task of the Adaptor. Although the job of the Adaptor is crucial, there is no systematic method of obtaining one.

This means that for each SUT one would need to develop a new Adaptor to be able to communicate with it. The goal of this work is to create a more generic Adaptor and thereby making it easier to support communication with a new SUT in the future.

Contribution In this BSc thesis project [23], we have systematically derived an adaptor for a LEGO bricks sorting machine. The machine takes as input black of white LEGO bricks, and sorts white bricks to the left and black bricks to the right. Such sorting machines reflect the working of more realistic applications, like luggage sorting systems in airports, or bottle sorting machines in recycle stations.

An adaptor has two main tasks. The first task for the Adaptor is to translate the messages from the driver into stimuli for the robot, and to translate the output from the robot into observations. Because we are going to test a Lego robot, this means we need to translate the stimuli into actions performed by one or more Lego Mindstorms motors. The observations will be a bit more difficult. The sensors provided by Lego Mindstorms do not always generate the expected results. Because we are performing a black-box test we will not be using the motors or sensors used by the SUT itself. Therefore we will be using a second NXT microcomputer. The second task for the Adaptor is to correctly pass all the messages from and to the SUT. This is important because the Driver needs to know if the sequence in which the messages arrive at the Adaptor are as good as possible. The problem here occurs when both the stimulus and the observation are sent at about the same time. In this case TorX might think the system is in a different state then the actual state. This might lead to errors that do not exist. The synchronisation problem occurs because the communication between the Adaptor and the Driver needs to be synchronised, while the communication between the Adaptor and the SUT can not be synchronised. Another problem that might occur is a buffer-overflow. Sometimes the SUT sends out a burst of observations at the same time, first of all, none of these observations may get lost at any point, all of these must be sent to the Driver as soon as possible.

The generic Adaptor also needs to fulfil these requirements. But it also needs to be as generic as possible, so it can be used for other SUTs as well.

Perspective Currently, the handling of quantitative system aspects (i.e. values of the light sensors) takes place in the SUT, but not in the model, which is purely qualitative.

In future work, we plan to tackle qualitative aspects also at a model level, dealing with the approximate aspects of this case study: This application is in essence approximate: it given the value of the light intensity, the machine determines if this concerns a white or black brick. Black and white are colours with very distinct light intensities, but this is not true for other colours: colours form by definition a spectrum, so it is not always possible to determine if a brick is e.g. orange or red. This requires slack, handline measurement errors, and all other issues related to approximate testing. Hence, the realised prototype is an excellent starting point to study these phenomena.

4 Automata Learning Using Abstraction

This work has been published as [2, 4, 3, 1].

Participants: Fides Aarts, Faranak Heidarian, Frits Vaandrager (ESI/RU),
Julien Schmaltz (Open University of The Netherlands, NL),
Petur Olsen (AAU),
Bengt Jonsson (University of Uppsala, Sw.).

Context Model-based testing (MBT) is a promising software testing technique for the automation of test generation and test execution. However, creating models is a complex, time-consuming, and error-prone task. Moreover, software systems evolve rapidly and models have to be updated or even new models have to be constructed. This cost of developing and maintaining models is a major obstacle to the wide adoption of MBT.

Hence, the problem to build a state machine model of a system by providing inputs to it and observing the outputs resulting, often referred to as black box system identification, is both fundamental and of clear practical interest. A major challenge is to let computers perform this tasks in a rigorous manner for systems with large numbers of states. Tools that are able to infer state machine models automatically by systematically "pushing buttons" and recording the resulting behaviour will have numerous applications in different domains. For instance, they may help us to (a) understand and analyse the behaviour of legacy software, (b) to find security vulnerabilities in implementations of security protocols, and (c) can be used to automatically derive tests to check that a protocol behaves in accordance with a reference implementation.

Within active learning it is assumed that a *learner* infers an automaton through interaction with a *teacher*. The well-known L^* algorithm of Angluin [5], for instance, assumes that the teacher knows a FSM $\mathcal{M}$. The learner (initially) only knows the set of actions and her task is to learn a machine that is equivalent to $\mathcal{M}$. The teacher will answer two types of questions: *membership queries* ("is string w in the language accepted by $\mathcal{M}$") and *equivalence queries* ("is an hypothesised machine $\mathcal{H}$ correct, i.e., equivalent to the machine $\mathcal{M}$?"). In case of a no-answer, the teacher will also provide a counterexample that proves that the learner's hypothesis is wrong, that is, a distinguishing word from the language. After posing a finite number of queries, the algorithm will terminate with a final hypothesis $\mathcal{H}$ that is equivalent to $\mathcal{M}$.

Figure 1 illustrates how active learning can be used to obtain models of "reactive" systems. The core of the teacher now is a *SUT (System Under Test)*, a (physical) system to which we can apply inputs and whose outputs we may observe. The learner interacts directly with the SUT to infer a model. Since the SUT cannot respond to equivalence queries, the teacher is also equipped with a tool for model based testing (MBT). Given a hypothesised model, this tool "approximates" equivalence queries by generating a long test sequence using some model based testing algorithm. If the SUT passes this test, that is, the output that is generated by the SUT agrees with the output predicted by the model, then we assume that the model is correct. If the output of the SUT is different from the output of the model, this constitutes a

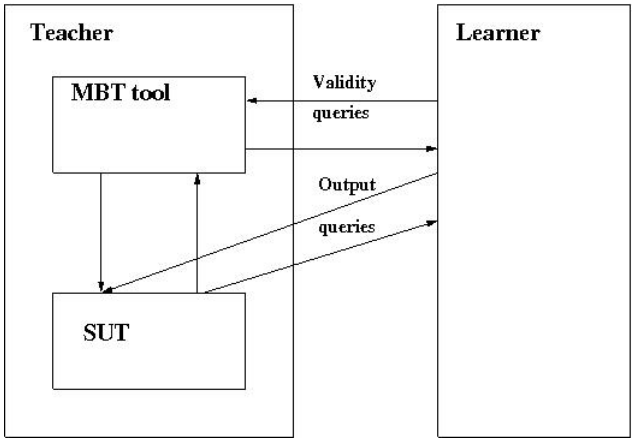

Figure 1: Active learning of reactive systems

counterexample that is forwarded to the learner. It is important to note that in this setting the learner, in general, is not perfect: due to incomplete coverage, it may occur that the implementation $\mathcal{M}$ passes the test for an hypothesis $\mathcal{H}$, even though $\mathcal{M}$ and $\mathcal{H}$ are not equivalent. Using the scheme of Figure 1, Niese [20] developed an adaptation for active learning of deterministic Mealy machines. This approach has been implemented in the LearnLib tool [22] and has, for instance, been applied successfully to learn computer telephony integrated (CTI) systems [14]. LearnLib, which is the winner of the 2010 Zulu competition, is currently able to learn state machines with up to 30.000 states.

During the last few years important developments have taken place on the borderline of verification, model-based testing and automata learning, see e.g. [7, 16, 22]. There are many reasons to expect that by combining ideas from these three areas it will become possible to learn models of realistic software components with state-spaces that are many orders of magnitude larger than what state-of-the-art tools can currently handle.

Contribution In [4], we established links between three widely used modelling frameworks for reactive systems: the ioco theory of Tretmans, the interface automata of De Alfaro and Henzinger, and Mealy machines. It is shown that, by exploiting these links, any tool for active learning of Mealy machines can be used for learning I/O automata that are deterministic and output determined. The main idea is to place a transducer in between the I/O automata teacher and the Mealy machine learner, which translates concepts from the world of I/O automata to the world of Mealy machines, and vice versa. The transducer comes equipped with an interface automaton that allows us to focus the learning process on those parts of the behaviour that can effectively be tested and/or are of particular interest. The approach has been implemented on top of the LearnLib tool and has been applied successfully to three case studies.

Last year, we also proposed a new method for automatically learning models of large state machines [2]. Their idea is to place a so-called *mapper* $\mathcal{A}$ in between the SUT $\mathcal{M}$ and the learner, which transforms the interface of the SUT by an abstraction that maps (in a history

dependent manner) the large set of actions of the SUT into a small set of abstract actions. By combining the abstract machine $\mathcal{H}$ learnt in this way with information about the mapper $\mathcal{A}$, one can effectively learn a (symbolically represented) over-approximation of the behaviour of SUT $\mathcal{M}$. Roughly speaking, the learner is responsible for learning the global "control modes" in which the system can be, and the transitions between those modes, whereas the mapper records some relevant state variables (typically computed from the data parameters of previous input and output actions) and takes care of the data part of the SUT. The approach

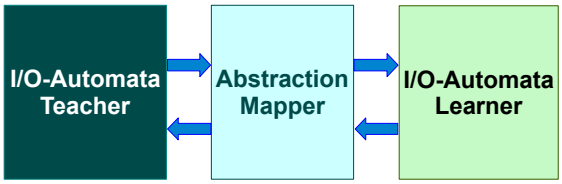

Figure 2: Active learning with an abstraction mapping.

of [2] has been inspired by ideas from predicate abstraction [17]. The feasibility of the approach was illustrated by learning models of (fragments of) realistic protocols such as SIP and TCP [2]. The learnt SIP model is an extended finite state machine with 29 states, 3741 transitions, and 17 state variables with various types (booleans, enumerated types, (long) integers, character strings,..). This corresponds to a state machine with an astronomical number of states and transitions, thus far fully out of reach of automata learning techniques. A major limitation of the approach of [2], however, is that the abstraction mapping has to be provided by the user, based on a priori knowledge of the SUT.

In [3], we apply the abstraction technique of [2] to learn a model of the new generation of biometric passports [15, 8]. The main idea of the abstraction technique is to extract a bit of *a priori* knowledge from documentation or interviews and use it to divide concrete parameter values into a small number of equivalence classes. This speeds up the learning process and reduces model size. In contrast to the previous application of this technique in [2], we validate our automatically derived model against a previous hand-made specification of the passport [19]. This specification was used to validate the Dutch implementation of the biometric passport using the ioco-theory for MBT [24]. We implemented our abstraction as a mapping module and connected it to the LearnLib library for regular inference [21]. After translating our automatically derived Mealy machine to a Labelled Transition System (LTS), we used the tool JTorX [6] to show that this learnt model is ioco-conforming to the hand-made specification. Our model can be learnt within one hour and is of comparable complexity and readability as the hand-made one. It took several hours to develop the latter.

In [1], we develop an algorithm that computes the mapper fully automatically for a restricted class of SUTs. Nondeterminism arises naturally when we apply abstraction: it may occur that the behaviour of a SUT is fully deterministic but that due to the mapper (which, for instance, abstracts from the precise value of certain input parameters), the system appears

to behave nondeterministically from the perspective of the learner. In [2] LearnLib is used as the basic learning tool, and therefore the abstraction of the SUT as defined by the mapper may not exhibit any nondeterminism: if it does then one has to refine the abstraction. This is exactly what has been done repeatedly during the manual construction of the abstraction mappings in the case studies of [2]. In [1], we formalise this procedure and describe the construction of the mapper in terms of a counterexample guided abstraction refinement (CEGAR) procedure, similar to the approach developed by Clarke et al. [10] in the context of model checking. Our algorithm applies to a restricted class of extended finite state machines in which one can test for equality of data parameters, but no operations on data are allowed. Using a prototype implementation of our algorithm, we have succeeded to learn – fully automatically – models of several realistic software components, including the biometric passport and the SIP protocol. The idea to use CEGAR for learning state machines has also been explored recently by Howar at al. [13], who developed and implemented a CEGAR procedure for the special case in which the abstraction is static and does not depend on the execution history. As we illustrate in [1], our approach is applicable to a much richer class of systems, which for instance includes the SIP protocol and the various components of the Alternating Bit Protocol.

Perspective Due to problems with non-deterministic behaviour of SUTs (for instance the passport), a topic for future research is to extend our approach to inference of non-deterministic systems. Such an extension will be essential, when doing more real-world case studies like this one. We expect that our CEGAR based approach can be further extended to systems that may apply simple or known operations on data, using technology for automatic detection likely invariants, such as Daikon [12]. Even though the class of systems to which our approach currently applies is limited, the fact that we are able to learn models of systems with data fully automatically is a major step towards a practically useful technology for automatic learning of models of software components.

5 Test Selection and the Preciseness of Learning

Participants: Jan Tretmans (ESI)
Frits Vaandrager (ESI/RU)
Published in [28].

5.1 Introduction

Model-based testing is a promising new technology that can contribute to increasing the efficiency and effectiveness of the testing process. In model-based testing, a model is the starting point for testing. This model shall express precisely and completely what the SUT, the System Under Test, should do, and should not do, and, consequently, it is a good basis for systematically generating test cases. This model is presumed to be correct and valid.

Obtaining or constructing a valid model, however, may be difficult if the system is complex, if specifiers and designers do not make models, if the system includes legacy or third-party components, or if documentation is missing or incomplete. This was also apparent in the Quasimodo case studies, such as in the case study about conformance testing of the Chess Wireless Sensor network node; see Deliverable D5.10: *Final report: Case Studies and Tool Integration*.

The availability of models is therefore a key issue that inhibits the further proliferation of model-based testing and of other forms of model-based and model-driven analysis and development [27]. Even if models are available, they are often incomplete and not fully valid. This means that a straightforward model-based testing process, consisting of the sequential steps of making a model, generating test cases from the model, executing these test cases, and assigning a verdict, is too naive and does not work. Such a process does not take into account that a model may not exist, is difficult to construct due to lack of information, or, if it exists, may be erroneous, misunderstood, incomplete, or invalid. This implies that a discrepancy between actual outcomes and expected ones during model-based testing does not necessarily point to a fault in the SUT, but may be due to problems in the model.

5.2 Model-Based Testing and Test-Based Modelling

Practical model-based testing is not only checking an SUT with respect to a model, but also checking and improving the model itself. Model-based testing serves as a technique to detect discrepancies between the SUT and the model, but without making any judgment about which one is wrong. Only subsequent analysis and diagnosis can show whether the model shall be adapted, or the SUT shall be repaired. What we see is a process of concurrent improvement of both the SUT and the model by iteratively comparing them using tests: the SUT is improved using tests generated from the model, and the model is refined using observations made during these tests. The benefit of model-based testing is that this comparison is fully automated. And when in the beginning there is no model at all, this modification process starts from scratch with an 'empty' model, building up and 'learning' the model completely from observations that are made by applying tests to the SUT.

On the one hand the not infrequent practice of using model-based testing to construct and refine models, and on the other hand recent theoretical developments in automata learning and grammatical inference, have led to an interesting area of research and development on the borderline of testing, verification, and machine learning, referred to as *model learning* or *test-based modelling*. Other terms are behaviour capturing, observation-based modelling, or just learning. The idea of model learning is to systematically perform experiments, or tests, on a (black-box) SUT, so that from the observations made during these tests a model can be constructed. It is a kind of black-box reverse engineering; see also Section 4 of Deliverable D4.4.

Model-based testing and learning are two sides of the same coin. Both use an SUT, a model, and tests, and aim at discovering discrepancies between behaviour described in the model and behaviour exhibited by the (black-box) SUT. Model-based testing starts with the model, and a discrepancy is in the first place considered a failure of the SUT, and an incentive to modify the SUT, after which it can be retested. Learning starts with an SUT, and a discrepancy is an incentive to adapt the hypothesized model, after which the next cycle of learning can start. In practical situations the difference often disappears, as on the one hand complete and valid models for model-based testing are often lacking for various reasons, and on the other hand many learning algorithms critically depend on a model-based testing step to check an hypothesized model. This intermingled use of model-based testing and test-based modelling leads to a cyclic, iterative, and 'agile' process of testing, learning, SUT modification, and model adaptation.

5.3 Preciseness of Learning and Test Selection

A number of learning approaches have their roots in the so-called L^* algorithm of Angluin [5]. One of these developments is an adaptation of L^* for Mealy Machines, which has been implemented in the LEARNLIB tool environment [26]. The goal of this approach is to learn a Mealy-Machine model from testing and observing a black-box SUT; see Section 4 of Deliverable D4.4.

Another approach to learning is based on the **ioco**-implementation relation [25, 29]. In this case there is not one unique model that can be learned, but there is a whole class of candidate models, ranging from a model isomorphic to the SUT, to the chaos model, which allows any behaviour, and thus is a valid **ioco**-model for any SUT. The isomorphic model describes very precisely the behaviour of the SUT, whereas the chaos model is very imprecise and does not provide any additional information. On the other hand, the isomorphic model will be very expensive, in terms of learning effort and cost, to derive, whereas the chaos model is very cheap. Many valid **ioco**-correct models exist 'between' the isomorphic and the chaos model.

It follows that **ioco**-learning can be expressed as an optimization problem searching for a model that optimally balances the cost of learning against obtained precision. An important question then is expressing the preciseness of the resulting, learned model. It turns out that this question is exactly the same as the question of test selection and coverage in model-based testing, and that it can be presented completely in the formal framework of model-based

testing [25].

Whereas LEARNLIB-style learning starts with an invalid, 'empty' model, and aims at finding an isomorphic model (in terms of Mealy Machines), **ioco**-style learning starts with the valid chaos model, which is subsequently refined and kept valid, until a sufficiently precise model is achieved, within the constraints of effort and cost. LEARNLIB-style learning, if it succeeds, gives a very precise, isomorphic model, but it might fail due to complexity. Learning in the **ioco**-approach promises better scalability because it always gives an answer, although it might be with less precision, but it is negotiable against cost of learning.

6 Learning Probabilistic System Models

Participants: Yingke Chen and Hua Mao, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen and Brian Nielsen (AAU).

Context The aim of this work is to learn compact and accurate probabilistic system model abstractions and approximations of a real system from observations of the system's interactions with its environment.

For complex systems that are only partially observable via their interactions with the user, it can be quite unrealistic to assume that an adequate deterministic model exists. For example, an embedded system in a mobile device may react differently to identical input sequences when operated in different physical environments. Data obtained by observing the behaviour of such a system could not be used in Angluin-style learning, where it is assumed that data is noise-free, which, in this case, means that each finite input sequence is uniquely labelled as either accepted or rejected by the system.

Contribution Probabilistic models provide the ability to construct accurate quantitative models of a system's observable (and possibly non-deterministic) behaviour. The data we require for learning only consists of previously observed system behaviours (e.g. obtained through active or passive testing). It is not assumed that the system for which a model is to be constructed is available for testing or interactive data generation in the learning process (however, our techniques might be refined by active learning techniques that take advantage of such interactive data acquisition).

In [9] we adapt and improve the Alergia algorithm for the task of learning (Labelled) Markov Chain system models for verification. We show theoretically and experimentally that the learnt models provide an accurate approximation of PLTL-expressible properties of the original, data-generating system.

Perspective Learning supports model construction at different levels of abstraction since it takes place relative to a set of observations that the learner can make of the system under test. These observations may be based on input and output actions, or may be based on states by evaluating a set of state predicates. By selecting different actions or predicates, different learnt models reflect different views of the underlying system. Thus, tailoring the observations to the properties of interest in the succeeding verification tasks will make verification easier and more efficient. Further, for such abstractions we find it more informative to preserve probabilistic information about system choices, rather than representing them as a purely deterministic or nondeterministic model.

References

[1] F. Aarts, F. Heidarian, P. Olsen, and F.W. Vaandrager. Automata learning through counterexample-guided abstraction refinement, May 2011. Submitted to ATVA 2011. Available through URL `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/CEGAR11/`.

[2] F. Aarts, B. Jonsson, and J. Uijen. Generating models of infinite-state communication protocols using regular inference with abstraction. In A. Petrenko, J.C. Maldonado, and A. Simao, editors, *22nd IFIP International Conference on Testing Software and Systems, Natal, Brazil, November 8-10, Proceedings*, volume 6435 of *Lecture Notes in Computer Science*, pages 188–204. Springer, 2010.

[3] F. Aarts, J. Schmaltz, and F.W. Vaandrager. Inference and abstraction of the biometric passport. In T. Margaria and B. 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 673–686. Springer, 2010.

[4] F. Aarts and F.W. Vaandrager. Learning i/o automata. In P. Gastin and F. Laroussinie, editors, *21st International Conference on Concurrency Theory (CONCUR), Paris, France, August 31st - September 3rd, 2010, Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2010.

[5] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[6] A. Belinfante. JTorX: A tool for on-line model-driven test derivation and execution. In *TACAS*, pages 266–270, 2010.

[7] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen. On the correspondence between conformance testing and regular inference. In M. 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.

[8] BSI. Advanced security mechanisms for machine readable travel documents - extended access control (eac) - version 1.11. Technical Report TR-03110, German Federal Office for Information Security (BSI), Bonn, Germany, 2008.

[9] Yingke Chen, Hua Mao, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning Probabilistic Automata for Model Checking. In *The Eighth International Conference on Quantitative Evaluation of SysTems (QEST 2011). Accepted.*, 2011.

[10] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[11] Alexandre David, Kim G. Larsen, Shuhao Li, Marius Mikucionis, and Brian Nielsen. Testing real-time systems under uncertainty. *LNCS (Submitted to Post-conference proceedings for FMCO'2010)*, 2011.

[12] M.D. Ernst, J.H. Perkins, P.J. Guo, S. McCamant, C. Pacheco, M.S. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, 2007.

[13] F. Howar, B. Steffen, and M. Merten. Automata Learning with Automated Alphabet Abstraction Refinement. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'11), January 23-25, 2011, Austin, Texas, USA*, 2011. To appear.

[14] H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In W.A. Hunt Jr. and F. 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.

[15] ICAO. Doc 9303 - machine readable travel documents - part 1-2. Technical report, International Civil Aviation Organization, 2006. Sixth edition.

[16] M. Leucker. Learning meets verification. In F.S. de Boer, M. M. Bonsangue, S. Graf, and W.P. de Roever, editors, *Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures*, volume 4709 of *Lecture Notes in Computer Science*, pages 127–151. Springer, 2006.

[17] C. Loiseaux, S. Graf, J. Sifakis, A. Boujjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.

[18] Marius Mikucionis, Kim G. Larsen, and Brian Nielsen. Uppaal-tron users manual, June 2009.

[19] W. Mostowski, E. Poll, J. Schmaltz, J. Tretmans, and R. Wichers Schreur. Model-based testing of electronic passports. In *FMICS '09: Proceedings of the 14th International Workshop on Formal Methods for Industrial Critical Systems*, pages 207–209, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] O. Niese. *An Integrated Approach to Testing Complex Systems*. PhD thesis, University of Dortmund, 2003.

[21] H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05: Proceedings of the 10th international workshop on Formal*

methods for industrial critical systems, pages 62–71, New York, NY, USA, 2005. ACM Press.

[22] H. Raffelt, B. Steffen, T. Berg, and T. Margaria. Learnlib: a framework for extrapolating behavioral models. *STTT*, 11(5):393–407, 2009.

[23] Arjan Snippe. Tolero: Torx-tested lego robots. Technical report, University of Twente, 2010.

[24] J. Tretmans. Test generation with inputs, outputs, and repetitive quiescence. *Software–Concepts and Tools*, 17:103–120, 1996.

[25] Jan Tretmans. Model based testing with labelled transition systems. In Robert Hierons, Jonathan Bowen, and Mark Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-78917-8-1.

[26] H. Raffelt, M. Merten, B. Steffen, and T. Margaria. Dynamic testing via automata learning. *Software Tools for Technology Transfer*, 11(4):307–324, 2009.

[27] J. Tretmans, editor. *Tangram: Model-Based Integration and Testing of Complex High-Tech Systems*, Eindhoven, The Netherlands, 2007. Embedded Systems Institute.

[28] J. Tretmans. Model-Based Testing and Some Steps towards Test-Based Modelling. In M. Bernardo and V. Issarny, editors, *SFM 2011*, volume 6659 of *Lecture Notes in Computer Science*, pages 297–326. Springer-Verlag, 2011.

[29] T.A.C. Willemse. Heuristics for IOCO-Based Test-Based Modelling. In L. Brim, B. Haverkort, M. Leucker, and J. v.d. Pol, editors, *Formal Methods Applications and Technology:* 11^{th} *Int. Workshop on Formal Methods for Industrial Critical Systems – FMICS 2006, and* 5^{th} *Int. Workshop on Parallel and Distributed Methods in Verification – PDMC 2006*, volume 4346 of *Lecture Notes in Computer Science*, pages 123–147. Springer-Verlag, 2007.