Inference and Abstraction of the Biometric Passport*

Fides Aarts[1], Julien Schmaltz[1,2], and Frits Vaandrager[1]

[1] Institute for Computing and Information Sciences, Radboud University Nijmegen
{f.aarts,f.vaandrager}@cs.ru.nl
[2] School of Computer Science, Open University of The Netherlands
julien.schmaltz@ou.nl

Abstract. Model-based testing is a promising software testing technique for the automation of test generation and test execution. One obstacle to its adoption is the difficulty of developing models. Learning techniques provide tools to automatically derive automata-based models. Automation is obtained at the cost of time and unreadability of the models. We propose an abstraction technique to reduce the alphabet and large data sets. Our idea is to extract a priori knowledge about the teacher and use this knowledge to define equivalence classes. The latter are then used to define a new and reduced alphabet. The a priori knowledge can be obtained from informal documentation or requirements. We formally prove soundness of our approach. We demonstrate the practical feasibility of our technique by learning a model of the new biometric passport. Our automatically learned model is of comparable size and complexity of a previous model manually developed in the context of testing a passport implementation. Our model can be learned within one hour and slightly refines the previous model.

1 Introduction

Learning techniques, e.g., regular inference (also known as automata learning) [4], can be used to automatically create a model from an existing implementation. The regular inference algorithms provide sequences of inputs, called *membership queries*, to a system and observe the responses. In addition, *equivalence queries* check whether the procedure is completed. The practical application of learning techniques faces two issues: (1) the time to learn a model grows very fast with the size of the input alphabet and (2) automatically learned models are hard to read. Recently, a new abstraction technique has been proposed to reduce the size of the input alphabet and learn readable models [1, 2].

Model-based testing (MBT) is a promising software testing approach providing full automation of test-cases generation and test-cases execution. Test-cases are automatically derived from a specification model of the System Under Test

* Supported by the European Community's 7th Framework Programme No. 214755 (QUASIMODO).

(SUT). The MBT paradigm requires the existence of a formal model. Developing 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. Learning techniques could here play a role. In contrast to models, prototypes and partial implementations are always available during the development of software. One could learn a model from a reference implementation and use this model to test whether new implementations are still conforming to this reference model. Learning techniques could be used to derive the model at the first place.

In this paper, we apply the abstraction technique of Aarts et al. [1, 2] to learn a model of the new generation of biometric passports [10, 6]. 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 a previous application of this technique [1, 2] we validate our automatically derived model against a previous hand-made specification of the passport [13]. This specification was used to validate the Dutch implementation of the biometric passport using the ioco-theory for MBT [16]. We implemented our abstraction as a mapping module and connected it to the LearnLib library for regular inference [15]. After translating our automatically derived Mealy machine to a Labelled Transition System (LTS), we used the tool JTorX [5] to show that this learned model is ioco-conforming to the hand-made specification. Our model can be learned within one hour and is of comparable complexity and readability as the hand-made one. It took several hours to develop the latter.

Our main contribution is to demonstrate and validate the applicability of our abstraction technique for learning automata to a practical and realistic case-study. The main result is that the model learned is comparable in size and correct w.r.t. to a previously hand-made specification. The time needed for a computer to learn the model from an existing implementation is much less than the time needed by a human to develop it.

The rest of the paper is organized as follows. In the next section, we give an overview of our approach. In Section 3, we review the Mealy machine model, regular inference, and our abstraction technique. Section 4 gives a short overview of the biometric passport; the experiments and according results are reported in Section 5. Finally, Section 6 contains conclusions and directions for future work.

2 Overview

Our approach works as follows. The goal is to learn a model of a *SUT* - the biometric passport. For the learning process we use three components: a *Learner* (LearnLib), a *Teacher* (*SUT*), and an intermediate layer called *Abstraction mapping* that reduces the alphabet of the *SUT*, see Learning box in Figure 1. The abstraction mapping is created using a priori knowledge extracted from informal

specifications, observing the behavior of the *SUT*, interviews with experts, etc. Eventually, the learning algorithm generates a Mealy machine model of the *SUT*. If a reference model is available, we can validate the learned implementation to check whether it is correct with respect to the specification. In our approach, we use the testing relation **ioco** [16, 17], which is implemented in the JTorX tool [5]. The Mealy machine model has to be transformed to an Input-Output Transition System (IOTS) to allow comparison with the specification represented as a LTS, see Validation box in Figure 1. We use an abstracted version of the specification to conform to the alphabet defined in the IOTS. The abstract LTS is based on a formal model created by Mostowski et al. [13] to adopt model-based testing. Their model was fed to the testing tool TorXakis (based on TorX [18]) that automatically generates and executes test cases on-the-fly. By comparing the responses of the *SUT* to those specified in the model, a verdict can be made, see MBT box in Figure 1.

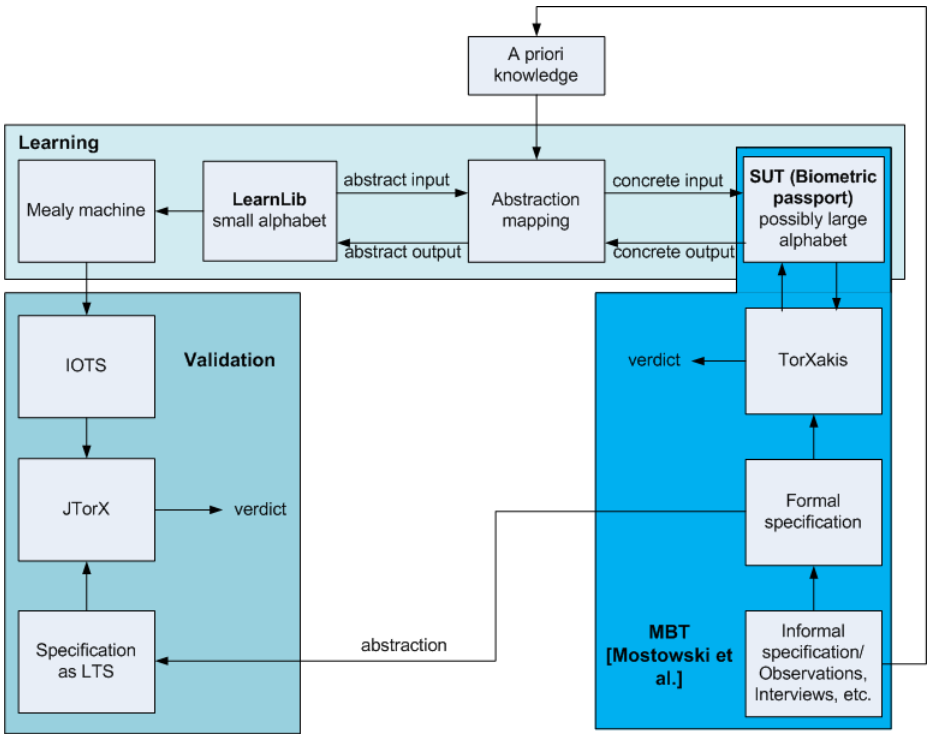

Fig. 1. Overview

3 Inference and Abstraction of Mealy Machines

In this section, we present basic principles of Mealy machines, how to infer them and to what extent abstraction techniques can be useful within learning.

3.1 Mealy Machines

A *(nondeterministic) Mealy machine (MM)* is a tuple $\mathcal{M} = \langle I, O, Q, q^0, \rightarrow \rangle$, where

- I, O and Q are finite, nonempty sets of input symbols, output symbols, and states, respectively,
- $q^0 \in Q$ is the initial state, and
- $\rightarrow \subseteq Q \times I \times O \times Q$ is the transition relation.

We write $q \xrightarrow{i/o} q'$ if $(q, i, o, q') \in \rightarrow$, and $q \xrightarrow{i/o}$ if there exists a q' such that $q \xrightarrow{i/o} q'$. Mealy machines are assumed to be *input enabled*: for each state q and input i, there exists an output o such that $q \xrightarrow{i/o}$. The transition relation is extended to sequences by defining $\overset{u/s}{\Rightarrow}$ to be the least relation that satisfies, for $q, q', q'' \in Q$, $u \in I^*$, $s \in O^*$, $i \in I$, and $o \in O$.

$$q \overset{\epsilon/\epsilon}{\Rightarrow} q$$
$$q \overset{u/s}{\Rightarrow} q' \wedge q' \xrightarrow{i/o} q'' \Rightarrow q \overset{u\ i/s\ o}{\Rightarrow} q''$$

A Mealy machine is *deterministic* if for each state q and input i there is exactly one output o and exactly one state q' such that $q \xrightarrow{i/o} q'$.

An intuitive interpretation of a Mealy machine is as follows. At any point in time, the machine is in some state $q \in Q$. It is possible to give inputs to the machine by supplying an input symbol $i \in I$. The machine then (nondeterministically) selects a transition $q \xrightarrow{i/o} q'$, produces output symbol o, and transforms itself to the new state q'.

For $q \in Q$ and $u \in I^*$, define $obs_{\mathcal{M}}(q, u)$ to be the set of output sequences that may be produced when offering input sequence u to $\mathcal{M}$, that is, $obs_{\mathcal{M}}(q, u) = \{s \in O^* \mid \exists \overline{q} : q \overset{u/s}{\Rightarrow} \overline{q}\}$. Two states $q, q' \in Q$ are *observation equivalent*, notation $q \approx q'$, if $obs_{\mathcal{M}}(q, u) = obs_{\mathcal{M}}(q', u)$, for all input strings $u \in I^*$. We write $obs_{\mathcal{M}}(u)$ as a shorthand for $obs_{\mathcal{M}}(q^0, u)$. Two Mealy machines $\mathcal{M}_1$ and $\mathcal{M}_2$ with the same sets of inputs I are *observation equivalent*, notation $\mathcal{M}_1 \approx \mathcal{M}_2$, if $obs_{\mathcal{M}_1}(u) = obs_{\mathcal{M}_2}(u)$, for all input strings $u \in I^*$. We say that $\mathcal{M}$ is *behavior deterministic* if $obs_{\mathcal{M}}(u)$ is a singleton set for each input sequence u. It is easy to see that a deterministic Mealy machine is also behavior deterministic.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two Mealy machines with the same sets of input symbols. A *bisimulation* between $\mathcal{M}_1$ and $\mathcal{M}_2$ is a relation $S \subseteq Q_1 \times Q_2$ satisfying:

$$q_1\ S\ q_2 \wedge q_1 \xrightarrow{i/o}_1 q_1' \Rightarrow \exists q_2' : q_2 \xrightarrow{i/o}_2 q_2' \wedge q_1'\ S\ q_2',$$
$$q_1\ S\ q_2 \wedge q_2 \xrightarrow{i/o}_2 q_2' \Rightarrow \exists q_1' : q_1 \xrightarrow{i/o}_2 q_1' \wedge q_1'\ S\ q_2'.$$

We say that Mealy machines $\mathcal{M}_1$ and $\mathcal{M}_2$ are *bisimilar*, notation $\mathcal{M}_1 \simeq \mathcal{M}_2$, if there exists a bisimulation relation between them that contains the pair (q_1^0, q_2^0). Since the union of bisimulations is again a bisimulation, there exists a largest bisimulation. We write $q_1 \simeq q_2$ if the pair $(q_1, q_2) \in Q_1 \times Q_2$ is contained in the largest bisimulation. The following lemma is well-known and easy to prove.

Lemma 1. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be Mealy machines with the same sets of inputs I and let $\mathcal{M}_2$ be deterministic. Then, for $q_1 \in Q_1$ and $q_2 \in Q_2$, $q_1 \simeq q_2$ iff $q_1 \approx q_2$.*

3.2 Inference of Mealy Machines

In this section, we present the setting for inference of Mealy machines. For this purpose we make use of an extension to Angluin's L^* algorithm [4] due to Niese [14]. There is a *Teacher*, who knows a behavior deterministic Mealy machine $\mathcal{M}$, and a *Learner*, who initially has no knowledge about $\mathcal{M}$, except for its sets I and O of input and output symbols. The *Learner* can ask two types of queries to the *Teacher*:

- A *membership query* consists in asking what the response is to an input string $u \in I^*$. The *Teacher* answers with an output string $s \in O^*$.[3]
- An *equivalence query* consists in asking whether a hypothesized machine $\mathcal{H}$ is correct, i.e., whether $\mathcal{H}$ is observation equivalent to $\mathcal{M}$. The *Teacher* will answer *yes* if $\mathcal{H}$ is correct or else supply a *counterexample*, which is a string $u \in I^*$ such that u produces a different output string for both automata, i.e., $obs_{\mathcal{M}}(u) \neq obs_{\mathcal{H}}(u)$

The typical behavior of a *Learner* is to start by asking a sequence of membership queries until a "stable" hypothesis $\mathcal{H}$ can be built from the answers. After that an equivalence query is made to find out whether $\mathcal{H}$ is equivalent to $\mathcal{M}$. If the result is successful, the *Learner* has succeeded. Otherwise the returned counterexample is used to perform subsequent membership queries until converging to a new hypothesized automaton, which is supplied in an equivalence query, etc.

3.3 Inference Using Abstraction

Existing implementations of inference algorithms only proved effective when applied to machines with small alphabets (sets of input and output symbols). Practical systems, however, typically do not have small alphabets. An example are communication protocols that interact with each other via messages consisting of an action type and a number of parameters, each of which can potentially take on a large number of values. As a result, the number of input and output symbols may be astronomical. In previous work, we developed a technique for

[3] Actually, the term *membership query* does not conform to this setting, because we do not check whether a certain string belongs to the language or not. In fact, the term *output query* would be more appropriate. However, because it is commonly used, we decided to keep the term *membership query* in the continuation of this paper.

using regular inference to infer models of large-state Mealy machines [1, 2]. The main idea is to transform the interface of the SUT by an abstraction mapping. We have adapted ideas from predicate abstraction [12, 8], which has been successful for extending finite-state model checking to larger and even infinite state spaces.

Assume that we are given the task of inferring a (possibly large) Mealy machine $\mathcal{M}$ that describes the behavior of a SUT. In order to make the learning task feasible, we place a transducer in between the *Learner* and the SUT, which transforms large (parameter) domains of input and output strings of machine $\mathcal{M}$ into small ones. The combined behavior of the SUT and the transducer can be described by a Mealy machine $\mathcal{M}^{\mathcal{A}}$, which has smaller alphabets and (hopefully) also a smaller state space. This makes the task for the *Learner* simpler. All membership and equivalence queries generated by the *Learner* are translated by the transducer into realistic messages with possibly large (parameter) domains to accomplish the communication with the SUT, see the Learning part in Figure 1. Answers to queries are handled the opposite way around. To create an abstraction mapping for the transducer, we need to define when and how a concrete symbol is mapped to an abstract symbol. In general, this may require a set V of state variables to remember previous values and expressions that define when to update them. When not enough *a priori* knowledge about the behavior of $\mathcal{M}$ is available in the informal specification, an initial observation phase or interviews may be needed to create the abstraction mapping. Once we have inferred a Mealy machine equivalent to $\mathcal{M}^{\mathcal{A}}$, we have learned quite a bit about the behavior of the SUT, that is, about $\mathcal{M}$. In fact, in some cases (and this includes the biometric passport) we can even infer $\mathcal{M}$ from $\mathcal{M}^{\mathcal{A}}$ if we are willing to make certain structural assumptions (e.g. symmetries) about the behavior of the SUT.

Example. Consider a parameter Id that is used for a session establishment. We know that the SUT establishes a connection for the first generated Id value. All other newly generated values are treated in the same way, i.e. they are rejected. Accordingly, we can divide the values into two equivalence classes $FIRST$ and NEW. The learning algorithm generates one of these two abstract values, which are translated to a concrete value by the abstraction mapping. Moreover, we store the concrete value in a state variable *firstId* or *newId*. These variables can be used to map a concrete output value to an abstract one. Assume that according to the specification, the SUT should return the Id received in the previous input message. By comparing the generated concrete output value to the value in the state variable, we can assign the output symbol one of the abstract values: $EQUAL$ or NOT_EQUAL.

4 Biometric Passport

The biometric passport is an electronic passport provided with a computer chip and antenna to authenticate the identity of travelers. The data stored on the

passport are highly confidential, e.g. they might contain fingerprints or an iris scan of its owner, and are protected via several mechanisms to avoid and detect attacks. Examples of used protocols are Basic Access Control (BAC), Active Authentication (AA), and Extended Access Control (EAC) [6]. Official standards are documented in the International Civil Aviation Organisation's (ICAO) Doc 9303 [10].
In this paper, we take a look at the interaction of the following messages:

- *Reset* resets the system.
- *GetChallenge* followed by *CompleteBAC* forms a *BAC*, which establishes secure messaging with the passport by encrypting transmitted information.
- *FailBAC* constitutes an invalid *BAC*.
- *ReadFile(int file)* tries to access highly sensitive data specified in a certain file, which is represented as an integer value in the range from 256 up to (and including) 511.
- *AA* prevents cloning of passport chips.
- *CA* followed by *TA* forms an *EAC*, which uses mutual authentication and stronger encryption than *BAC* to control access to highly confidential data.
- *FailEAC* constitutes a valid *CA* and an invalid *TA*.

For each of these messages a value *OK* or *NOK* may be returned by the *SUT*. A global overview of the valid behavior is depicted in Figure 2, where a *BAC* consists of a *GetChallenge* followed by a *CompleteBAC* and an *EAC* constitutes a *CA* followed by a *TA*. The files 257 and 258 should be readable after a BAC. File 257 contains Machine Readable Zone (MRZ) data, i.e. name, date of birth, nationality, document number, etc. whereas file 258 contains a facial image. File 259 comprises biometric data like fingerprints or an iris scan, which are only readable after a *BAC* followed by an *EAC*. All other files should not be readable at any point in time.

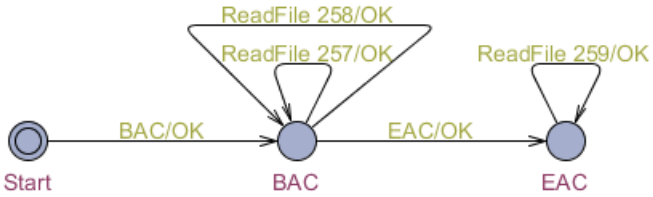

Fig. 2. Simplified model of the biometric passport

5 Experiments

We have implemented and applied our approach to infer a model of the biometric passport described in Section 4. In this section, we first describe our experimental setup, thereafter its application and a validation of our technique.

We used an authentic biometric passport as *SUT*. The data on the chip could be accessed via a smart card reader; JMRTD[1] served as API. We connected the *SUT* to an abstraction mapping, which performed a translation as described in Section 5.1. As *Learner*, we used the LearnLib library [15], developed at the Technical University Dortmund. This tool provides a Java implementation of the L* algorithm adapted by Niese. Because LearnLib views the *SUT* as a black box, equivalence queries can only be approximated by a large number of membership queries. In our experiments we used the W-Method by Chow [7] for equivalence approximation.

5.1 Abstraction Mapping

As described in Section 4, only the *ReadFile* message has a parameter called *file*, which can take on integer values in the range from 256 up to (and including) 511. Actually, each of these numbers has to be considered separately in the inference process, which would require a lot of time and memory space. By taking a closer look at the informal specification of the passport, we discovered that different files should be treated in the same way by the *SUT*. As one can see in Figure 2, files 257 and 258 should be readable after a BAC, 259 after a BAC followed by an EAC and the rest of the files should never be readable. Using this *a priori* knowledge about the passport, we can divide the values into three disjoint equivalence classes, which are:

- *ValidAfterBAC* refers to the files that can be read after a BAC, i.e. 257 and 258.
- *ValidAfterEAC* refers to the files that only can be read after a BAC followed by an EAC, i.e. 259.
- *NotValid* refers to the files that can never be read, i.e. all files except for 257, 258 and 259.

In the abstraction mapping an abstract value is translated to a concrete one by randomly choosing an element within the corresponding equivalence class. If the numbers are partitioned incorrectly, then there are two values in the same class that will produce a different response. This non-deterministic behavior will be detected by LearnLib, which will give an error message.

5.2 Results

The inference performed by LearnLib needed about one thousand membership queries and one equivalence query, and resulted in a model $\mathcal{H}^{\mathcal{A}}$ with five states and 55 transitions. Without our abstraction mapping, the Mealy machine would have had 1320 transitions, but also five states. The total learning time took less than one hour[4]. This is significantly shorter than deriving the model manually

[4] The experiments have been carried out on a PC with an Intel Pentium M 1.86GHz processor and 1GB of RAM.

from the informal specs, which took about 5 hours. All results are summarized in Table 1. For presentation purposes, we have depicted the model as follows: (1) we removed self-transitions with *NOK* as output. Because the model is input-enabled all missing entries refer to this kind of transition. (2) Transitions with same source location, output symbol and next location (but with different input symbols) are merged by concatenating the input symbols, separated by a bar (|). The resulting transition diagram has five locations and 19 transitions as shown in Figure 3.

Membership queries	1078[a]
Total input symbols used in membership queries	4158
Average membership query length	3,867
Equivalence queries	1
Total learning time	< 60 minutes

[a] This number does not include membership queries used for equivalence approximation.

Table 1. Learning statistics

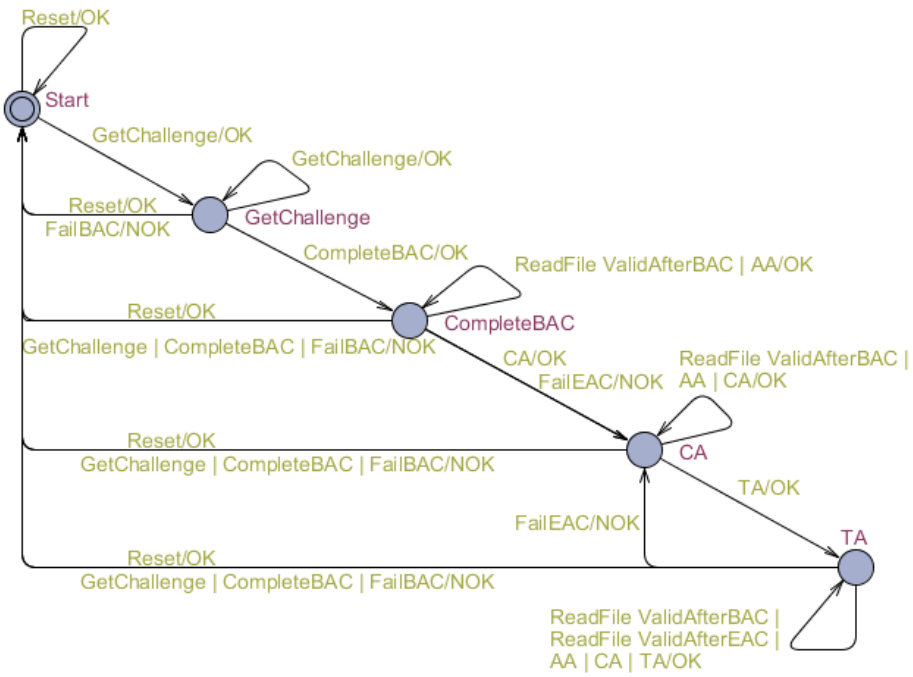

Fig. 3. Learned model $\mathcal{H}^{\mathcal{A}}$ of the biometric passport

The implementation of the biometric passport does not respond to a *Reset* input. For all other outputs the reaction time is dependent on the input symbol. If the waiting time for an output is too short, then an output symbol may be returned after a *timeout* has been assumed. In contrast, if the waiting time is too long, then the passport application crashes after certain inputs. As a solution, we changed the API of the *SUT*, so that it returns an *OK* symbol for each *Reset* input. By always returning an output symbol, we do not have to struggle with appropriate waiting times per input symbol. Instead, we wait until an output is received.

According to the passport specification, the implementation should be deterministic. However, surprisingly, the passport application sometimes exhibits non-deterministic behavior. LearnLib is restricted to infer behavior deterministic Mealy machines and cannot cope with non-deterministic behavior. Analyzing the external behavior of the system revealed that after a *GetChallenge*, *Complete-BAC*, *CA*, *TA* input sequence mostly an *OK* is returned, but in some rare cases it can also be a *NOK*. Together with Mostowski et al. we tried to examine the internal behavior of the application to understand where the non-determinism originates from. During their work this problem has also been encountered, but it has never been reported. Because a *TA* call includes numerous complex and long calculations, a problem can arise at several places. Moreover, external circumstances may influence the produced results like connection to or temperature of the smart card reader. In the end, we could not clearly determine the fault location and had to accept that the inference can fail once in a while.

5.3 The Behavior of the SUT

We assume that the behavior of the digital passport can be modeled in terms of a behavior deterministic Mealy machine $\mathcal{M}$. Clearly, due to the abstraction that we applied, the learned model $\mathcal{H}^{\mathcal{A}}$ is not equivalent to $\mathcal{M}$: even the alphabets are different. Let $\mathcal{M}^{\mathcal{A}}$ be the Mealy machine obtained from $\mathcal{M}$ by renaming each action *ReadFile*(*file*) in accordance with the abstraction mapping defined in Section 5.1. We assume that also $\mathcal{M}^{\mathcal{A}}$ is behavior deterministic. Since the SUT and the transducer together behave like $\mathcal{M}^{\mathcal{A}}$, the learned model $\mathcal{H}^{\mathcal{A}}$ should be equivalent to $\mathcal{M}^{\mathcal{A}}$. LearnLib implements several algorithms that can be used to "approximate" equivalence queries, that is, to establish that the hypothesized machine $\mathcal{H}^{\mathcal{A}}$ is observation equivalent to the model $\mathcal{M}^{\mathcal{A}}$ of the teacher. We have used the well-known W-method of [7] (see also [11]). This method assumes a known upper bound on the number of states n of $\mathcal{M}^{\mathcal{A}}$. Depending on n the W-method provides a test sequence of input symbols u with the property that $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$ are observation equivalent iff they produce the same output in response to u. But assuming that we have established equivalence of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, what have we learned about $\mathcal{M}$?

We reverse the abstraction mapping and construct a "concrete" model $\mathcal{H}$ of the passport as follows. We replace each *ValidAfterBAC* transition in $\mathcal{H}^{\mathcal{A}}$ by two transitions with the same source and target but with labels *ReadFile(257)*

and *ReadFile(258)*, respectively. Similarly, we replace each transition with label *ValidAfterEAC* by an identical transition with label *ReadFile(259)*. Finally, we replace each transition with label *NotValid* by 253 identical transitions with labels *ReadFile(256)*, *ReadFile(260)*, *ReadFile(261)*, ..., *ReadFile(511)*, respectively.

The following theorem states that if $\mathcal{M}$ treats equivalent input symbols in an equivalent way, observation equivalence of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$ implies observation equivalence of $\mathcal{M}$ and $\mathcal{H}$. So provided we are willing to make a structural assumption about the behavior of the biometric passport, our abstraction does not lead to any loss of information.

Theorem 1. *Call two input actions i_1, i_2 of Mealy machine $\mathcal{M}$* equivalent*, notation $i_1 \equiv i_2$, if they are mapped to the same abstract action. Suppose that in $\mathcal{M}$ equivalent inputs induce identical outputs and equivalent successor states:*

$$i_1 \equiv i_2 \wedge q \xrightarrow{i_1/o_1} q_1 \wedge q \xrightarrow{i_2/o_2} q_2 \Rightarrow o_1 = o_2 \wedge q_1 \approx q_2.$$

Then $\mathcal{M}^{\mathcal{A}} \approx \mathcal{H}^{\mathcal{A}}$ implies $\mathcal{M} \approx \mathcal{H}$.

Proof. Suppose $\mathcal{M}^{\mathcal{A}} \approx \mathcal{H}^{\mathcal{A}}$. We must show that $\mathcal{M} \approx \mathcal{H}$. Since $\mathcal{H}^{\mathcal{A}}$ is deterministic, Lemma 1 implies that $\mathcal{M}^{\mathcal{A}} \simeq \mathcal{H}^{\mathcal{A}}$. Let S be the maximal bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$. It suffices to prove that S is a bisimulation between $\mathcal{M}$ and $\mathcal{H}$, since by Lemma 1 this implies $\mathcal{M} \approx \mathcal{H}$.

Since S relates the initial states of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, it also relates the initial states of $\mathcal{M}$ and $\mathcal{H}$.

Suppose that $(q, r) \in S$ and $q \xrightarrow{i/o}_{\mathcal{M}} q'$. Suppose that the abstraction function maps i to j. Then $q \xrightarrow{j/o}_{\mathcal{M}^{\mathcal{A}}} q'$. Since S is a bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, there exists a state r' such that $r \xrightarrow{j/o}_{\mathcal{H}^{\mathcal{A}}} r'$ and $(q', r') \in S$. By construction of $\mathcal{H}$, $r \xrightarrow{i/o}_{\mathcal{H}} r'$.

Now suppose that $(q, r) \in S$ and $r \xrightarrow{i/o}_{\mathcal{H}} r'$. Suppose the abstraction function maps i to j. Then, by construction of $\mathcal{H}$, $r \xrightarrow{j/o}_{\mathcal{H}^{\mathcal{A}}} r'$. Since S is a bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, there exists a state q' such that $q \xrightarrow{j/o}_{\mathcal{M}^{\mathcal{A}}} q'$ and $(q', r') \in S$. Therefore, by construction of $\mathcal{M}^{\mathcal{A}}$, there exists a concrete input label k such that the abstraction maps k to j and $q \xrightarrow{k/o}_{\mathcal{M}} q'$. Since $\mathcal{M}$ is input enabled, there exists an output label p and a state q'' such that $q \xrightarrow{i/p}_{\mathcal{M}} q''$. Oberve that $i \equiv k$. Hence, by the assumption of the theorem, $o = p$ and $q' \approx_{\mathcal{M}} q''$. By construction of $\mathcal{M}^{\mathcal{A}}$, this implies $q' \approx_{\mathcal{M}^{\mathcal{A}}} q''$ By Lemma 1, applied to $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, using the fact that $(q', r') \in S$, we obtain $(q'', r') \in S$.

5.4 Validation

To validate the learned model of the biometric passport, we compared it to a reference model taken from Mostowski et al. [13]. The specification is a LTS made

in Haskell[5] and has to be transformed to a different format to allow comparison with the inferred Mealy machine described in the *DOT*[6] language.

For the comparison, we used JTorX [5], a tool to test whether the **ioco** testing relation holds between a given specification and a given implementation. Intuitively, an implementation $i \in \mathcal{IOTS}(L_I, L_U)$ is input-output conforming to specification $s \in \mathcal{LTS}(L_I, L_U)$ if any experiment derived from s and executed on i leads to an output from i that is foreseen by s. For a formal definition, we refer to [17]. We have supplied JTorX with the specification and implementation as LTS - represented in Aldebaran[7] format. The learned Mealy machine has been transformed to a LTS by splitting each transition into two with the input symbol on the first transition and the output on the second one connected by an additional state. As a result, the input-enabledness of the Mealy machine gets lost. To convert the learned LTS to an IOTS, JTorX adds self-loop transitions to the according states. Furthermore, we removed the output *OK* for a *Reset* input, because it is unknown by the specification, see Section 5.2.

According to JTorX, the implementation is **ioco** conforming to the specification, but not vice versa. This is not surprising as the learned model is input-enabled while the specification is not. For example, the specification does not specify a *CompleteBAC* input in the initial state while the learned implementation does, see Figure 4 for a fragment of the specification. We only show the inputs in the initial state with according outputs, because the entire model contains too many states and transitions. As one can see, the automaton corresponds to a Mealy machine. Except for the *Reset* input, each input is followed by an output. If we would transform the specification to a Mealy machine, it would not be input-enabled. Because LearnLib infers an input-enabled Mealy machine of the implementation, it contains more behavior than described by the specification, which is allowed by the **ioco** testing relation.

6 Conclusions and Future Work

Using regular inference and abstraction, we have managed to infer a model of the biometric passport that describes how the passport responds to certain input sequences. Although quite a number of papers have been written on regular inference of state machines, the number of real applications to reactive systems is still limited. The case study that we describe here is a small but real application. The new biometric passport is going to be used by millions of people, and it is vital that the confidential information stored on this passport is well-protected. Our

[5] http://www.haskell.org
[6] http://www.graphviz.org
[7] http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html

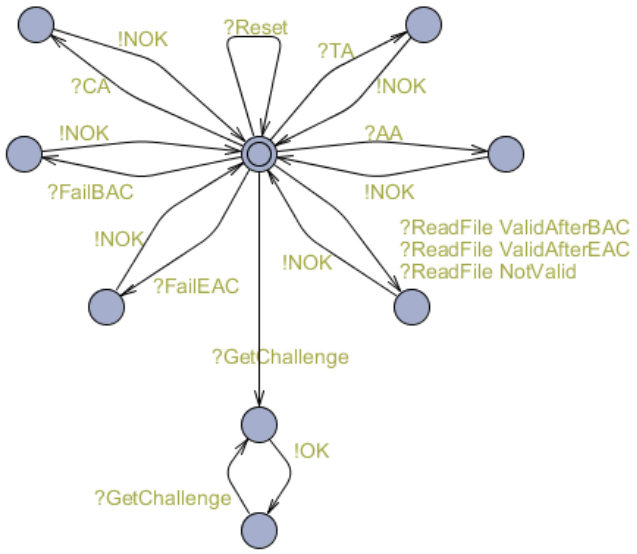

Fig. 4. Specification fragment of the biometric passport

model, which slightly refines the earlier model of [13], may serve as a reference model for testing different implementations of the biometric passport.

The data abstraction that we applied when learning the passport may seem rather obvious (and indeed is much simpler than the abstractions applied in [1, 2]), but is nevertheless crucial for the successful application of our learning framework. In order to prevent brute force attacks, the biometric passport only allows for about one input message per second. Without abstraction, the time needed to apply the framework (and in particular the approximation of equivalence queries via e.g. the W-method) would become prohibitively large. We have proven that under some reasonable assumptions about the behavior of the biometric passport, our abstraction does not lead to any loss of information.

The earlier model of [13] has been created manually in about 5 hours, whereas our model has been produced automatically in less than one hour. Our ambition is to further develop the learning framework, so that also for other applications it becomes feasible to mechanize and speed-up the time-consuming and error prone process of constructing reference models.

Due to the problems with the non-deterministic behavior of the passport, an obvious 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.

If inferring an input-enabled Mealy machine is too time-consuming and we are only interested in parts of the implementation, we may extend our abstraction mappings with an *interface automaton (IA)* as suggested by [3]. An interface automaton [9] is a labelled transition system with input and outputs, where

certain input actions may be illegal in certain states. When an input symbol or sequence generated by the learning algorithm is not allowed by the specified *IA*, this part of the implementation will not be inferred. By adding restrictions, we can focus on those parts of the implementation that are described by the specification.

Acknowledgement We are grateful to Falk Howar from the TU Dortmund for his generous LearnLib support, and Wojciech Mostowski for providing assistance with JMRTD.

References

1. F. Aarts. Inference and Abstraction of Communication Protocols. Master's thesis, Radboud University Nijmegen and Uppsala University, 2009.
2. F. Aarts, B. Jonsson, and J. Uijen. Generating Models of Infinite-State Communication Protocols using Regular Inference with Abstraction. In *Proceedings ICTSS 2010, 22^{nd} IFIP International Conference on Testing Software and Systems*, 2010.
3. F. Aarts and F. Vaandrager. Learning I/O Automata. In *Proceedings CONCUR 2010, 21^{th} International Conference on Concurrency Theory*, pages 71–85, 2010.
4. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
5. A. Belinfante. JTorX: A tool for on-line model-driven test derivation and execution. In *TACAS*, pages 266–270, 2010.
6. 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.
7. T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. on Software Engineering*, 4(3):178–187, May 1978. Special collection based on COMPSAC.
8. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.
9. L. de Alfaro and T.A. Henzinger. Interface automata. In V. Gruhn, editor, *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering (ESEC/FSE-01)*, volume 26 of *Software Engineering Notes*, pages 109–120, New York, September 2001. ACM Press.
10. ICAO. Doc 9303 - machine readable travel documents - part 1-2. Technical report, International Civil Aviation Organization, 2006. Sixth edition.
11. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines – a survey. *Proc. IEEE*, 84(8):1090–1126, 1996.
12. 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.
13. 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.

14. O. Niese. An integrated approach to testing complex systems. Technical report, Dortmund University, 2003. Doctoral thesis.
15. 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.
16. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.
17. J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, pages 1–38, 2008.
18. J. Tretmans and H. 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.