Learning I/O Automata*

Fides Aarts and Frits Vaandrager

Institute for Computing and Information Sciences, Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands

Abstract. Links are established between three widely used modeling 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 behavior 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.

1 Introduction

Model-based system development is becoming an increasingly important driving force in the software and hardware industry. In this approach, models become the primary artifacts throughout the engineering lifecycle of computer-based systems. Requirements, behavior, functionality, construction and testing strategies of computer-based systems are all described in terms of models. Models are not only used to reason about a system, but also used to allow all stakeholders to participate in the development process and to communicate with each other, to generate implementations, and to facilitate reuse. The construction of models typically requires significant manual effort, implying that in practice often models are not available, or become outdated as the system evolves. Automated support for constructing behavioral models of implemented components would therefore be extremely useful.

The problem of inducing, learning or inferring grammars and automata has been studied for decades, but only in recent years *grammatical inference* a.k.a. *grammar induction* has emerged as an independent field with connections to many scientific disciplines, including bio-informatics, computational linguistics and pattern recognition [10]. Also recently, some important developments have taken place on the borderline of verification, testing and machine learning, see

* This research was supported by European Community's Seventh Framework Programme under grant agreement no 214755 (QUASIMODO).

e.g. [6, 16, 23], and researchers have shown that it is possible (at least in principle) to apply grammatical inference for learning models of software components. Grammatical inference techniques aim at building a grammar or automaton for an unknown language, given some data about this language. Within the setting of active learning, it is assumed that a *learner* interacts with a *teacher*. Inspired by work of Angluin [5] on the L^* algorithm, Niese [20] developed an adaptation of the L^* algorithm for active learning of deterministic Mealy machines. This algorithm has been further optimized in [23]. In the algorithm it is assumed that the teacher knows a deterministic Mealy machine $\mathcal{M}$. Initially, the learner only knows the action signature (the sets of input and output symbols I and O) and her task is to learn a Mealy machine that is equivalent to $\mathcal{M}$. The teacher will answer two types of questions — *output queries* ("what is the output generated in response to input $i \in I$?") and *equivalence queries* ("is an hypothesized machine $\mathcal{H}$ correct, i.e., equivalent to the machine $\mathcal{M}$?"). The learner always records the current state q of Mealy machine $\mathcal{M}$. In response to query i, the current state is updated to q' and answer o is returned to the learner. At any point the learner can "reset" the teacher, that is, change the current state back to the initial state of $\mathcal{M}$. The answer to an equivalence query $\mathcal{H}$ is either **yes** (in case $\mathcal{M} \approx \mathcal{H}$) or **no** (in case $\mathcal{M} \not\approx \mathcal{H}$). Furthermore, the teacher will give the learner a counterexample that proves that the learner's hypothesis is wrong with every negative equivalence query response, that is, an input sequence $u \in I^*$ such that $obs_{\mathcal{M}}(u) \neq obs_{\mathcal{H}}(u)$. This algorithm has been implemented in the LearnLib tool [23]. In practice, when a real implementation is used instead of an idealized teacher, the implementation cannot answer equivalence queries. Therefore, LearnLib "approximates" such queries by generating a long test sequence that is computed by standard methods such as state cover, transition cover, W-method, and the UIO method (see [15]). LearnLib has been applied successfully to learn computer telephony integrated (CTI) systems [11], and more recently to learn parts of the SIP and TCP protocols [1] and the new biometric passport [2].

Currently, LearnLib is able to automatically learn Mealy machines with up to 30.000 states. Nevertheless, a lot of further research will be required to make automata based learning tools suitable for routine use on industrial case studies. An important issue, clearly, is the development of abstraction techniques in order to be able to learn much larger state spaces (see [1], also for further references). Another issue is the extension of automata learning techniques to nondeterministic systems (see e.g. [29]). In this paper, we address a third issue that hinders the application of the LearnLib tool. In practice, the restriction of Mealy machines that each input corresponds to exactly one output is felt as being overly restrictive. Sometimes several inputs are required before a single output occurs, sometimes a single input triggers multiple outputs, etc.

The I/O automata of Lynch & Tuttle [18, 17] and Jonsson [13] constitute a popular modelling framework which does not suffer from the restriction that inputs and outputs have to alternate. Our aim is to to develop efficient algorithms for active learning of I/O automata. Hence we assume that the teacher knows an I/O automaton $\mathcal{A}$. We consider a setting in which the task of the learner is to

partially learn $\mathcal{A}$. More specifically, we assume that the learner initially knows an interface automaton $\mathcal{P}$ in the sense of De Alfaro and Henzinger [8], called the *learning purpose*, and that she has to learn the part of $\mathcal{A}$ whose behavior is compatible with $\mathcal{P}$. We think there are several good reasons to extend the framework of active learning with a notion of a learning purpose. In principle, the systems that we model using I/O automata will accept any input in any state. But in practice, a learner may not be able (e.g., not fast enough) to effectively provide any input in any state. Also, systems are often designed to be used in a certain manner, and their behavior may become unspecified and/or nondeterministic when they are used improperly. In such cases a learner may decide to interact with the system following the specified interaction protocol, for instance "after providing an input a user should wait for the system to become quiescent before she may provide a next input". A final motivation for using learning purposes is that often the state space of practical systems is very big and cannot be fully explored. By not providing certain inputs (in certain states), the learner may focus on interesting parts of the behavior that can be effectively learned.

Rather than developing and implementing an algorithm from scratch, we will use LearnLib. Our idea is to place a *transducer* in between the IOA 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 and Mealy machine learner together then implement an IOA learner. Note that this architecture is very similar to the architecture proposed in [1], where a transducer is used to relate the large parameter spaces of realistic communication protocols to small sets of actions that can effectively be handled by state-of-the-art automata learning tools.

As a spin-off of our research, we establish links between three widely used modeling frameworks for reactive systems: the ioco theory of Tretmans [26, 27], the interface automata of De Alfaro and Henzinger [8], and Mealy machines. In particular, we present behavior preserving maps between interface automata and Mealy machines, and we link the ioco preorder to alternating simulation.

The rest of this paper is structured as follows. Section 2 recalls interface automata and links alternating simulation to the ioco preorder. Section 3 addresses a basic question: what is the I/O behavior of an I/O automaton? Section 4 recalls Mealy machines and discusses translations between interface automata and Mealy machines. Section 5 describes our framework for learning I/O automata. In Section 6, we describe the implementation of our approach and its application to three case studies. Finally, Section 7 wraps up with some conclusions and suggestions for further research.

2 Interface Automata

An interface automaton models a reactive system that can interact with its environment. It is a simple type of state machine in which the transitions are associated with named actions. The actions are classified as either input or output.

The output actions are assumed to be under the control of the system whereas the input actions are under control of the environment. The interface automata that we study in this paper are a simplified version of the interface automata of De Alfaro and Henzinger [8] without internal actions. Within ioco theory [26, 27] interface automata are called labelled transition systems with inputs and outputs. Interface automata are similar to the I/O automata of Lynch & Tuttle [18, 17] and Jonsson [13]. The main difference is that in an I/O automaton input actions are required to be enabled in every state. In an interface automata certain input actions may be illegal in certain states: they are not enabled and we assume that the environment will not generate such inputs.

In this paper, an *interface automaton (IA)* is defined to be a tuple $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$, where I, O and Q are finite, nonempty sets of input actions, output actions, and states, respectively, with I and O disjoint, $q^0 \in Q$ the initial state, and $\rightarrow \subseteq Q \times (I \cup O) \times Q$ the transition relation. We write $q \xrightarrow{a} q'$ if $(q, a, q') \in \rightarrow$. An action a is *enabled* in state q, notation $q \xrightarrow{a}$, if $q \xrightarrow{a} q'$, for some state q'. We write $out_{\mathcal{A}}(q)$, or just $out(q)$ if $\mathcal{A}$ is clear from the context, for the set $\{a \in O \mid q \xrightarrow{a}\}$ of output actions enabled in state q. For $S \subseteq Q$ a set of states, we write $out_{\mathcal{A}}(S)$ for $\bigcup\{out_{\mathcal{A}}(q) \mid q \in S\}$. An *I/O automaton (IOA)* is an interface automaton in which each input action is enabled in each state, that is $q \xrightarrow{i}$, for all $q \in Q$ and all $i \in I$. A state q is called *quiescent* if it enables no output action. An interface automaton $\mathcal{A}$ is

- *input deterministic* if for each state $q \in Q$ and for each action $a \in I$ there is at most one outgoing transition of q with label a: $q \xrightarrow{a} q_1 \wedge q \xrightarrow{a} q_2 \Rightarrow q_1 = q_2$;
- *output deterministic* if for each state $q \in Q$ and for each action $a \in O$ there is at most one outgoing transition of q with label a: $q \xrightarrow{a} q_1 \wedge q \xrightarrow{a} q_2 \Rightarrow q_1 = q_2$;
- *deterministic* if it is both input and output deterministic;
- *output determined* if each state has at most one outgoing output transition:

$$q \xrightarrow{o_1} q_1 \wedge q \xrightarrow{o_2} q_2 \wedge \{o_1, o_2\} \subseteq O \Rightarrow o_1 = o_2 \wedge q_1 = q_2.$$

Figure 1 displays a simple example of a deterministic IOA that is also output determined. The initial state is marked with an extra circle, there is a single input action *in* and there are two output actions *out*1 and *out*2.

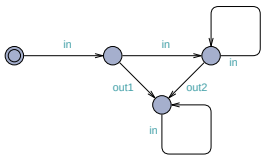

Fig. 1. A deterministic, output determined IOA.

Let $\mathcal{A}_1 = (I, O, Q_1, q_1^0, \rightarrow_1)$ and $\mathcal{A}_2 = (I, O, Q_2, q_2^0, \rightarrow_2)$ be interface automata with the same signature. Let $A = I \cup O$ and let $X, Y \subseteq A$. A binary relation $R \subseteq Q_1 \times Q_2$ is an *XY-simulation* from $\mathcal{A}_1$ to $\mathcal{A}_2$ if whenever $(q, r) \in R$ and $a \in A$ it holds that:

– if $q \xrightarrow{a}_1 q'$ and $a \in X$ then there exists a $r' \in Q_2$ s.t. $r \xrightarrow{a}_2 r'$ and $(q', r') \in R$.
– if $r \xrightarrow{a}_2 r'$ and $a \in Y$ then there exists a $q' \in Q_1$ s.t. $q \xrightarrow{a}_1 q'$ and $(q', r') \in R$.

We write $\mathcal{A}_1 \leq_{XY} \mathcal{A}_2$ if there exists an XY-simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$ that contains (q_1^0, q_2^0). AA-simulations are commonly known as *bisimulations* and OI-simulations are known as *alternating simulations* [4]. De Alfaro and Henzinger [8] propose alternating simulations as the primary notion of refinement for IAs. In their approach, one IA refines another if it has weaker input assumptions and stronger output guarantees. We often write $\mathcal{A}_1 \leq_a \mathcal{A}_2$ instead of $\mathcal{A}_1 \leq_{OI} \mathcal{A}_2$ and $\mathcal{A}_1 \approx_b \mathcal{A}_2$ instead of $\mathcal{A}_1 \leq_{AA} \mathcal{A}_2$. There are several obvious inclusions between the different preorder, e.g. it follows that $\mathcal{A}_1 \leq_{AY} \mathcal{A}_2$ implies $\mathcal{A}_1 \leq_{XY} \mathcal{A}_2$.

Figure 2 shows an example of an alternating simulation between two IAs with inputs $\{in1, in2\}$ and outputs $\{out1, out2, d\}$.

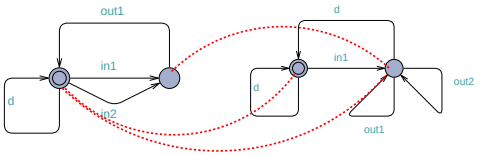

Fig. 2. Example of alternating simulation (from left to right IA).

Suppose that $\mathcal{A}_1 \leq_a \mathcal{A}_2$ and that R is the largest alternating simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. We define $\mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2)$, the *alternating simulation interface automaton* induced by $\mathcal{A}_1$ and $\mathcal{A}_2$, as the structure $(I, O, R, (q_1^0, q_2^0), \rightarrow)$ where

$$(q, r) \xrightarrow{a} (q', r') \Leftrightarrow q \xrightarrow{a}_1 q' \text{ and } r \xrightarrow{a}_2 r'.$$

Figure 3 shows the alternating simulation IA induced by the IAs of Figure 2. The following lemma follows easily from the definitions.

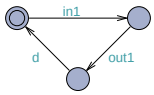

Fig. 3. IA induced by alternating simulation of Figure 2.

Lemma 1. *Suppose* $\mathcal{A}_1 \leq_a \mathcal{A}_2$. *Then* $\mathcal{A}_1 \leq_{OA} \mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2) \leq_{AI} \mathcal{A}_2$.

Larsen, Nyman and Wasowski [14] criticize interface automata and alternating simulations for being unable to express liveness properties and since they allow for trivial implementations: an IA $\mathcal{T}$ with a single state that accepts all inputs but never produces any output is a refinement of any IA over the same signature. In order to fix this problem, Larsen, Nyman and Wasowski [14] define model automata, an extension of interface automata with modality. In this paper, we propose a different solution, which is very simple and in the spirit of I/O automata and ioco theory: we make quiescence observable.

Let $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$ be an IA and let δ be a fresh symbol (not in $I \cup O$). Then the *δ-extension* of $\mathcal{A}$, notation $\mathcal{A}^\delta$, is the IA obtained by adding δ-loops to all the quiescent states of $\mathcal{A}$. Formally, $\mathcal{A}^\delta = (I, O_\delta, Q, q^0, \rightarrow')$ where $O_\delta = O \cup \{\delta\}$ and $\rightarrow' = \rightarrow \cup \{q \xrightarrow{\delta} q \mid q \in Q \text{ quiescent}\}$. For $\mathcal{A}_1$ and $\mathcal{A}_2$ IAs with the same signature, we define $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2 \Leftrightarrow \mathcal{A}_1^\delta \leq_a \mathcal{A}_2^\delta$.

Observe that in general $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1 \leq_a \mathcal{A}_2$, but $\mathcal{A}_1 \leq_a \mathcal{A}_2$ does not imply $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$: even though $\mathcal{T} \leq_a \mathcal{A}$, for any IA $\mathcal{A}$ with the same signature as our trivial IA $\mathcal{T}$, we do in general not have $\mathcal{T} \leq_{a\delta} \mathcal{A}$. If $\mathcal{A}^\delta$ enables a sequence of input actions leading to a state r from which an output is possible, then $\mathcal{T}^\delta$ must enable the same sequence of inputs leading to a related state q. But whereas q enables a δ-transition, r does not enable a matching δ-transition. In order to argue that $\leq_{a\delta}$ indeed is a reasonable notion of implementation, we will now show that — under certain determinacy assumptions — $\leq_{a\delta}$ coincides with the well-known ioco preorder of [26, 27].

We extend the transition relation to sequences by defining, for $\sigma \in (I \cup O)^*$, $\stackrel{\sigma}{\Rightarrow}$ to be the least relation that satisfies, for $q, q', q'' \in Q$ and $a \in I \cup O$,

$$q \stackrel{\epsilon}{\Rightarrow} q$$
$$q \stackrel{\sigma}{\Rightarrow} q' \wedge q' \xrightarrow{a} q'' \Rightarrow q \stackrel{\sigma a}{\Rightarrow} q''$$

Here ϵ denotes the empty sequence. We say that $\sigma \in (I \cup O)^*$ is a *trace* of $\mathcal{A}$ if $q^0 \stackrel{\sigma}{\Rightarrow} q$, for some state q, and write $Traces(\mathcal{A})$ for the set of traces of $\mathcal{A}$. We write $\mathcal{A}$ **after** σ for the set $\{q \in Q \mid q^0 \stackrel{\sigma}{\Rightarrow} q\}$ of states of $\mathcal{A}$ that can be reached with trace σ. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IA with the same signature. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are *input-output conforming*, notation $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$, if

$$\forall \sigma \in Traces(\mathcal{A}_2^\delta) : out(\mathcal{A}_1^\delta \textbf{ after } \sigma) \subseteq out(\mathcal{A}_2^\delta \textbf{ after } \sigma)$$

The results below link alternating simulation and the ioco preorder. These results generalize a similar, recent result of Veanes and Bjørner [28], which is stated in a setting of fully deterministic systems. We first state a small technical lemma.

Lemma 2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Let R be an alternating simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. Let $\sigma \in (I \cup O)^*$, $q_1 \in Q_1$ and $q_2 \in Q_2$ such that $q_1^0 \stackrel{\sigma}{\Rightarrow} q_1$ and $q_2^0 \stackrel{\sigma}{\Rightarrow} q_2$. Then $(q_1, q_2) \in R$.*

Theorem 1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input deterministic and $\mathcal{A}_2$ is output deterministic. Then $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$ implies $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$.*

Theorem 2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IAs with the same action signature such that $\mathcal{A}_1$ is input enabled and $\mathcal{A}_2$ is deterministic. Then $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$ implies $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$.*

Corollary 1. *Let $\mathcal{A}_1$ be an input deterministic IOA and let $\mathcal{A}_2$ be a deterministic IA with the same action signature. Then $\mathcal{A}_1$ **ioco** $\mathcal{A}_2$ iff $\mathcal{A}_1 \leq_{a\delta} \mathcal{A}_2$.*

Observe that all the determinacy conditions in the above results are essential: as soon as one assumption is left out the corresponding result no longer holds.

3 The I/O Behavior of I/O Automata

In order to be able to learn I/O automata, we need to decide which type of questions the learner may ask to the teacher. One obvious proposal would be to allow for membership queries of the form "Is sequence $u \in (I \cup O)^*$ a (quiescent) trace of the IOA?". However, there is strong evidence that this is an inefficient approach. In his PhD thesis [20], Niese compared two algorithms for learning Mealy machines. The first algorithm, an optimized version of Angluin's [5] L^* algorithm, allowed for membership queries "Is sequence $u \in (I \times O)^*$ a trace of the MM?". The second algorithm supported membership queries "What is the output generated by the MM in response to input sequence $u \in I^*$?". Niese showed that the second algorithm has a much better performance and requires less membership queries. We expect that for IOAs the situation is very similar.

Lynch & Tuttle [18, 17] and Jonsson [13] do not define a notion of input/output behavior for I/O automata, that is, given a stream of input values that is provided by the environment, the stream of output values that is computed by the I/O automaton. The main reason for this is that such a notion of behavior is not compositional. Instead, the behavior of an IOA is defined in terms of traces, sequences of input and output actions that may be observed during runs of the automaton. Henzinger [9] links determinism to predictability and calls a reactive system *deterministic* if, for every stream of input values that is provided by the environment, the stream of output values that is computed by the system is unique. The example IOA of Figure 1 is not deterministic in this sense since the input stream *in in* may either lead to the output stream *out*1 or to the output stream *out*2. One obvious way to proceed is to restrict the class of IOA that one calls deterministic, and and to study a notion of input/output behavior for this restricted class. This route is explored by Panangaden and Stark [21] in their study of "monotone" processes. We will explore a different route, in which the power of testers is slightly increased and the IOA of Figure 1 becomes again behavior deterministic.

If a system is predictable then one may expect that, for any history of input and output actions, the time required by the system to produce its next output (if any) is more or less known. Predictability is at the basis of the assumption in ioco theory that quiescence is observable: whenever a test is carried out, it is assumed that if a system does not produce an output within some fixed time T after the last input, it will never produce an output. By the same line of reasoning, one could assume that there exists a fixed time t such that the system never produces an output within time t after an input. Hence, if one assumes that the tester can compute faster than the IUT, then in principle the tester always has the choice to either wait for the next output of the IUT or to generate its next input before time t, that is, before occurrence of the output. Based on these considerations, we slightly increase the power of the testers: at any point we let the tester decide who is going to perform the next step, the IUT or the tester itself.

Formally, we introduce a fresh *delay* action Δ. By performing Δ, the environment gives an IOA the opportunity to perform its next output (if any). Let

$I_\Delta = I \cup \{\Delta\}$. The behavior of an environment can then be described by an *environment sequence* in $(I_\Delta)^*$, that is, a sequence of input actions interleaved with delay actions. Let $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$ be an IA and let $q, q' \in Q$, $e \in (I_\Delta)^*$ and $u \in (I \cup O_\delta)^*$. We write $q \overset{e/u}{\Rightarrow} q'$ to indicate that as a result of offering environment sequence e in state q, $\mathcal{A}_\delta$ may produce trace u and end up in state q'. Formally, $\overset{e/u}{\Rightarrow}$ is the least relation that satisfies $q \overset{\epsilon/\epsilon}{\Rightarrow} q$ and:

$$q \overset{e/u}{\Rightarrow} q' \wedge q' \overset{i}{\rightarrow} q'' \wedge i \in I \Rightarrow q \overset{e\ i/u\ i}{\Rightarrow} q''$$
$$q \overset{e/u}{\Rightarrow} q' \wedge q' \overset{o}{\rightarrow} q'' \wedge o \in O_\delta \Rightarrow q \overset{e\ \Delta/u\ o}{\Rightarrow} q''$$

For each environment sequence $e \in (I_\Delta)^*$, we define $obs_\mathcal{A}(e)$ to be the set of traces that may be observed when offering e to $\mathcal{A}_\delta$, that is, $obs_\mathcal{A}(e) = \{u \in (I \cup O_\delta)^* \mid \exists q \in Q : q^0 \overset{e/u}{\Rightarrow} q\}$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two IOAs with the same sets I and O of input and output actions, respectively, We write $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$, if $obs_{\mathcal{A}_1}(e) \subseteq obs_{\mathcal{A}_2}(e)$, for all environment sequences $e \in (I_\Delta)^*$. If $\mathcal{A}$ is a deterministic and output determined IOA then $obs_\mathcal{A}(e)$ contains exactly one element for each input sequence e. Thus, with this notion of observable behavior, a deterministic and output determined IOA is also behavior deterministic in the sense of Henzinger [9].

Even though our notion of observation is based on a stronger notion of testing than ioco theory, the resulting notion of preorder is the same.

Theorem 3. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be IOAs with the same inputs and outputs. Then $\mathcal{A}_1$* **ioco** *$\mathcal{A}_2$ iff $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$.*

4 From Interface Automata to Mealy Machines and Back

A *(nondeterministic) Mealy machine (MM)* is a tuple $\mathcal{M} = (I, O, Q, q^0, \rightarrow)$, where I, O and Q are finite, nonempty sets of inputs, outputs, 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 $q \overset{i/o}{\rightarrow} q'$ if $(q, i, o, q') \in \rightarrow$, and $q \overset{i/o}{\rightarrow}$ if there exists a q' such that $q \overset{i/o}{\rightarrow} q'$. Mealy machines are assumed to be *input enabled:* for each state q and input i, there exists an output o such that $q \overset{i/o}{\rightarrow}$. 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 $o \in O$: $q \overset{\epsilon/\epsilon}{\Rightarrow} q$ and $q \overset{u/s}{\Rightarrow} q' \wedge q' \overset{i/o}{\rightarrow} q'' \Rightarrow q \overset{u\ i/s\ o}{\Rightarrow} q''$. A state $q \in Q$ is called *reachable* if $q^0 \overset{u/s}{\Rightarrow} q$, for some u and s. A Mealy machine is *deterministic* iff given a state q and an input i there is exactly one output o and exactly one state q' such that $q \overset{i/o}{\rightarrow} q'$.

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 $u \in I^*$. 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 $u \in I^*$. If $\mathcal{M}$ is deterministic then $obs_{\mathcal{M}}(u)$ is a singleton set for each input sequence u. Thus a deterministic Mealy machine is also behavior deterministic in the sense of Henzinger [9].

We call an interface automaton *active* if each state enables an output action. Observe that for any interface automaton $\mathcal{A}$, the δ-extension $\mathcal{A}^\delta$ is active. Active interface automata can be translated to equivalent Mealy machines. We translate each input transition $q \xrightarrow{i} q'$ to a transition $q \xrightarrow{i/+} q'$ in the Mealy machine, where $+$ is a fresh output action denoting that the input is accepted. If input i is not enabled in state q, then we add a self-loop $q \xrightarrow{i/-} q$ to the Mealy machine. Here $-$ is a fresh output symbol denoting that the input is illegal. The fresh input action Δ ("delay") is used to probe for possible outputs: each output transition $q \xrightarrow{o} q'$ translates to a transition $q \xrightarrow{\Delta/o} q'$ in the Mealy machine.

Formally, for active IA $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$, the Mealy machine $T(\mathcal{A})$ is defined as the structure $(I_\Delta, O \cup \{+,-\}, Q, q^0, \rightarrow')$, where

$$i \in I \wedge q \xrightarrow{i} q' \Rightarrow q \xrightarrow{i/+}{}' q'$$
$$i \in I \wedge q \not\xrightarrow{i} \Rightarrow q \xrightarrow{i/-}{}' q$$
$$o \in O \wedge q \xrightarrow{o} q' \Rightarrow q \xrightarrow{\Delta/o}{}' q'$$

Figure 4 illustrates transformation T. We now define transformation R, the

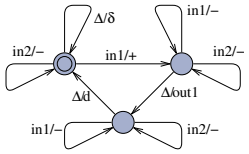

Fig. 4. Result of applying T to the δ-extension of the IA of Figure 3.

inverse of transformation T, which takes a Mealy machine and turns it into an IA. Let $\mathcal{M} = (I_\Delta, O \cup \{+,-\}, Q, q^0, \rightarrow)$ be a Mealy machine. Then $R(\mathcal{M})$ is the IA $(I, O, Q, q^0, \rightarrow')$, where

$$i \in I \wedge q \xrightarrow{i/+} q' \Rightarrow q \xrightarrow{i}{}' q' \quad \text{and} \quad o \in O \wedge q \xrightarrow{\Delta/o} q' \Rightarrow q \xrightarrow{o}{}' q'$$

If one takes any total IA $\mathcal{A}$ and applies first T and then R, one gets back $\mathcal{A}$.

Theorem 4. *Let $\mathcal{A}$ be a total IA. Then $\mathcal{A} = R(T(\mathcal{A}))$.*

Observe that if $\mathcal{A}$ is deterministic and output determined then $T(\mathcal{A})$ is deterministic, and if $\mathcal{M}$ is deterministic then $R(\mathcal{M})$ is deterministic and output

determined. In order to obtain a dual version of Theorem 4, we need to impose three additional conditions on $\mathcal{M}$. Let $\mathcal{M}$ be a Mealy machine whose inputs include Δ and whose outputs include $+$ and $-$. Then $\mathcal{M}$ is *separated* if an input in I always leads to an output $+$ or $-$, and input Δ always leads to an output in O: $q \xrightarrow{i/o} q' \;\Rightarrow\; (i = \Delta \Leftrightarrow o \in O)$. $\mathcal{M}$ is *consistent* if there exists no state q and input i for which both outputs $+$ and $-$ are possible: $\neg(q \xrightarrow{i/+} \wedge q \xrightarrow{i/-})$. $\mathcal{M}$ is *stable* if an output $-$ does not lead to a change of state: $q \xrightarrow{i/-} q' \;\Rightarrow\; q = q'$. Clearly, for any total IA $\mathcal{A}$, $T(\mathcal{A})$ is separated, consistent and stable. Note that deterministic Mealy machines are consistent. Using the conditions of separation, consistency and stability, it is easy to prove $\mathcal{M} = T(R(\mathcal{M}))$.

Theorem 5. *Let $\mathcal{M}$ be a separated, consistent and stable Mealy machine with inputs I_Δ and outputs $O \cup \{+, -\}$. Then $\mathcal{M} = T(R(\mathcal{M}))$.*

5 Learning I/O Automata

In this section, we present our approach for active learning of I/O automata. We assume that the teacher knows a deterministic and output determined IOA $\mathcal{A} = (I, O, Q, q^0, \rightarrow)$. We consider a setting in which the task of the learner is to *partially* learn $\mathcal{A}$: the learner initially knows a deterministic interface automaton $\mathcal{P} = (I, O_\delta, P, p^0, \rightarrow')$, called the *learning purpose*, and has to learn the part of $\mathcal{A}$ whose behavior is compatible with $\mathcal{P}$. We require $\mathcal{A}^\delta \leq_a \mathcal{P}$.

The teacher and learner play the following game. The teacher records the current state of $\mathcal{A}$, which initially is q^0, and the learner records the current state of $\mathcal{P}$, which initially is p^0. Suppose that the teacher is in state q and the learner is in state p. The learner now can do four things: (1) If an input transition $p \xrightarrow{i}' p'$ is enabled then it may jump to p' and present input i to the teacher, which will then jump to the state q' such that $q \xrightarrow{i} q'$. (2) The learner may present a delay Δ to the teacher. If the teacher enables some output o, then it will jump to the unique state q' such that $q \xrightarrow{o} q'$ and return answer o to the learner. If no output action is enabled in q then the teacher returns δ. The learner then jumps to the unique state p' that can be reached by the answer o or δ that it has just received (by the assumption that $\mathcal{A}^\delta \leq_a \mathcal{P}$ we know this state exists). (3) The learner may return to its initial state and ask the teacher to do the same ("'reset"). (4) The learner may pose a preorder query ("is an hypothesized IA $\mathcal{H}$ correct?"). An hypothesis is a deterministic, output determined IA $\mathcal{H}$ such that $\mathcal{H}^\delta \leq_{AI} \mathcal{P}$. An hypothesis is correct if $\mathcal{A} \leq_{a\delta} \mathcal{H}$. If $\mathcal{H}$ is correct then the teacher returns the answer **yes**. If an hypothesis is not correct then, by Corollary 1, $\mathcal{H}^\delta$ has a trace σ such that the unique output o enabled by $\mathcal{A}^\delta$ after σ differs from the unique output enabled by $\mathcal{H}^\delta$ after σ. The teacher then returns the answer **no** together with counterexample $\sigma\ o$.

In order to appreciate our learning framework, consider the trivial learning purpose $\mathcal{P}_{triv}$ displayed in Figure 5 (left). Here notation $i : I$ means that we have an instance of the transition for each input $i \in I$. Notation $o : O$ is defined

Fig. 5. A trivial learning purpose (left) and a learning purpose with a nontrivial δ transition (right).

similarly. If $\mathcal{H}$ is an hypothesis, then by definition $\mathcal{H}^\delta \leq_{AI} \mathcal{P}_{triv}$. This just means that $\mathcal{H}$ is input enabled. If $\mathcal{H}$ is correct then $\mathcal{A} \leq_{a\delta} \mathcal{H}$. Since both $\mathcal{A}$ and $\mathcal{H}$ are deterministic, output determined IAs, this means that $\mathcal{A}$ and $\mathcal{H}$ are bisimilar! The following lemma provides some key insight in our approach in case of an arbitrary learning purpose. It implies that if hypothesis $\mathcal{H}$ is correct, $\mathcal{H}^\delta$ is bisimilar to $\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P})$.

Lemma 3. *Suppose $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are IAs, $\mathcal{A}_1$ is active and input deterministic, $\mathcal{A}_2$ is output determined, $\mathcal{A}_3$ is output deterministic, and $\mathcal{A}_1 \leq_a \mathcal{A}_3 \leq_{AI} \mathcal{A}_2$. Then $\mathcal{A}_3 \approx_b \mathsf{AS}(\mathcal{A}_1, \mathcal{A}_2)$.*

It is important that a learning purpose may contain nontrivial δ transitions. As an example, consider the IA of Figure 5 (right). This learning purpose expressing that after an input one has to wait until the system gets into a quiescent state before offering the next input. It is not possible to express this without δ's. But since in the end we want to learn IAs without δ's, we need an operation that eliminates all δ-transitions from an automaton. Let $\mathcal{A} = (I, O_\delta, Q, q^0, \rightarrow)$ be an IA. Let $\equiv$ be the smallest equivalence relation that contains $\xrightarrow{\delta}$. Then we define $\rho(\mathcal{A})$ to be the quotient IA $(I, O, Q/\equiv, q^0/\equiv, \rightarrow')$ where

$$q/\equiv \xrightarrow{a}{}' q'/\equiv \Leftrightarrow \exists r, r' : q \equiv r \wedge r \xrightarrow{a} r' \wedge r' \equiv q'$$

The following lemma implies that under certain conditions operation ρ preserves bisimulation equivalence.

Lemma 4. *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, output determined IAs, $\mathcal{A}_1$ has outputs O, $\mathcal{A}_2$ has outputs O_δ, and both IAs share the same sets of inputs I. Suppose furthermore that $\mathcal{A}_2$ satisfies the following* triangle property*, for $i \in I$: $q \xrightarrow{\delta} q' \wedge q \xrightarrow{i} q'' \Rightarrow q' \xrightarrow{i} q''$. Then $\mathcal{A}_1^\delta \approx_b \mathcal{A}_2$ implies $\mathcal{A}_1 \approx_b \rho(\mathcal{A}_2)$.*

We always assume that the learning purpose $\mathcal{P}$ satisfies the triangle property. Under this assumption, it follows using the above lemma that, if hypothesis $\mathcal{H}$ is correct, $\mathcal{H}$ is bisimilar to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.

Rather than developing and implementing an algorithm from scratch, we use the LearnLib tool [23] to implement our learning approach. We place a *transducer* in between the IOA teacher and the Mealy machine learner, which records the current state p of the learning purpose $\mathcal{P}$ and translates concepts from the world of I/O automata to the world of Mealy machines, and vice versa, using the

translation functions defined in the previous section. Initially, the MM learner only knows a signature consisting of inputs I_Δ and outputs $O_\delta \cup \{+, -\}$. The behavior of the transducer is as follows:

- Whenever the transducer receives an output query $i \in I$ from the MM learner, it checks if i is enabled in the current state of $\mathcal{P}$. If the input is not enabled ("illegal") then the transducer returns an output $-$ to the MM learner. If the output is enabled then the transducer returns an output $+$ to the MM learner, updates state p to the unique state p' with an i-transition from p to p', and forwards i to the IOA teacher.
- Whenever the transducer receives an output query Δ this is forwarded directly to the IOA teacher. When it receives a response $o \in O_\delta$, the transducer updates state p accordingly, and forwards o to the MM learner.
- Whenever the transducer receives a "reset" from the MM learner, it resets its state to p^0, and forwards the "reset" to the IOA teacher.
- Whenever the transducer receives an equivalence query $\mathcal{H}$ from the MM learner, then it first checks whether $\rho(R(\mathcal{H})) \leq_{AI} \mathcal{P}$ (since both IAs are deterministic, this can be done in time linear in the size of their synchronous product). If $\rho(R(\mathcal{H}))$ does not conform to learning purpose $\mathcal{P}$, then an answer **no** is returned to the MM learner, together with a distinguishing trace in which all output symbols are replaced by Δ. If $\rho(R(\mathcal{H})) \leq_{AI} \mathcal{P}$ then the transducer forwards the preorder query $\rho(R(\mathcal{H}))$ to the IOA teacher. The transducer forwards a subsequent response of the IOA teacher to the MM learner, but with all output symbols replaced by Δ. If the response is **yes** then the transducer has successfully learned an IA $\rho(R(\mathcal{H}))$ that meets all the requirements.

Observe that when LearnLib is used, equivalence queries are always separated and stable. We claim that the algorithm always terminates and that the transducer indeed learns an IOA that is equivalent to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$. In order to see why this claim is true, a key observation is that the IOA teacher and transducer together behave like a teacher for Mealy machine $T(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.

Lemma 5. *The IOA teacher and transducer together behave like a teacher for Mealy machine $T(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.*

The main technical result of this article is that the MM learner and the transducer together will succeed in learning $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$, that is, the subautomaton of $\mathcal{A}$ induced by the learning purpose $\mathcal{P}$:

Theorem 6. *The composition of MM learner and transducer behaves like a learner for I/O automata, that is, execution will terminate after a finite number of queries, and upon termination the transducer has learned an IA that is bisimilar to $\rho(\mathsf{AS}(\mathcal{A}^\delta, \mathcal{P}))$.*

6 Experiments

We have implemented and applied our approach to infer three types of I/O automata. In this section, we first describe our experimental setup, thereafter its application to the three case studies.[1]

To serve as IOA teacher, we read in an I/O automaton specified in Aldebaran format [3]. We connect the IOA teacher to a transducer equipped with an interface automaton, which is also described in Aldebaran format. As MM learner in our framework, we use the LearnLib library [22].

In our first (trivial) case study we learned the IOA shown in Figure 1. Because the interaction with this automaton is not constrained, we used an interface automaton that accepts every input and output, see Figure 5. The inferred Mealy machine model can be transformed to the IOA by transformations R and ρ.

A model of the electronic passport [12, 7] has been inferred in a second experiment. We provided the IOA teacher with a model of the protocol taken from Mostowski et al. [19]. Analyzing the behavior of the automaton revealed that almost always the passport reacts like a Mealy machine: 13 out of 14 inputs generate an output symbol before a new input symbol can be transferred. Following this information, we defined an interface automaton in which inputs alternate with outputs or quiescence, see Figure 6 (left). Because no output is generated in response to a *Reset* input in the IOA, an output δ occurs within the Mealy machine that is learned. In fact, the inferred Mealy machine has one additional state, which can only be reached by a Δ/δ transition. After applying transformation R and ρ, we obtained the corresponding subautomaton of the IOA that was given to the teacher. With respect to learning performance, we observe that inferring an IOA requires more membership queries than learning the same behavior as a Mealy machine having i/o instead of $i/+$ and Δ/o transitions. Inferring an IOA of the electronic passport required 44294 membership queries, whereas learning the corresponding Mealy machine with i/o transitions merely needed 1079 queries. The difference can be explained by the fact that 80,72% of the membership queries asked to infer the passport IOA comprised unspecified input sequences. Because of the Mealy machine behavior of the IOA, no outputs are defined for most consecutive inputs. Moreover, membership queries were asked for the additional state.

In a third case study we applied our approach to learn a model of the Session Initiation Protocol (SIP) [25, 24]. The teacher is supplied with an IOA based on a Mealy machine generated using inference and abstraction techniques [1]. Analyzing the structure of the automaton showed that each input symbol is followed by one or more outputs. Furthermore, in the initial state only certain inputs are allowed. To concentrate the learning on this restricted behavior, we used the interface automaton shown in Figure 6 (right). Again, by applying

[1] All IOAs and interface automata used in the different case studies as well as the corresponding learned Mealy machines can be found at the URL `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/LearningIOAs/`.

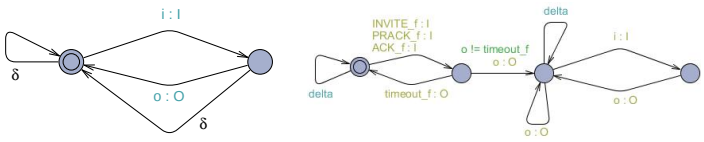

Fig. 6. IA in which each input is followed by at most one output (left) and IA in which initially only certain inputs are allowed and two consecutive inputs are not allowed (right).

transformation ρ and R, the inferred Mealy machine could be converted to the corresponding subautomaton of the IOA given to the teacher.

7 Conclusions and Future work

We have presented an approach for active learning of deterministic and output determined I/O automata. By eliminating the restriction from Mealy machines that inputs and outputs have to alternate, we have extended the class of models that can be learned. Our approach has been implemented on top of the LearnLib tool and has been applied successfully to three case studies. A new idea introduced in this paper is to use interface automata to focus the learning process to interesting/relevant parts of the behavior. Both in the passport and the SIP case study, the use of interface automata greatly reduced the number of queries. The efficiency of our learning approach can be improved by integrating this notion of interface automata within LearnLib: in this way it will be possible to further reduce the number of membership queries. Obvious topics for future research are to extend our approach to automata with nondeterminism and silent transitions, and to integrate our transducers with the ones used in [1] for data abstraction.

Acknowledgement Many thanks to Bengt Jonsson, Bernhard Steffen, Jan Tretmans and the anonymous referees for inspiring discussions and/or pointers to the literature, and to Falk Howar for his generous LearnLib support.

References

1. F. Aarts. *Inference and Abstraction of Communication Protocols.* Master thesis, Radboud University Nijmegen and Uppsala University, Nov. 2009.
2. F. Aarts, J. Schmaltz, and F. Vaandrager. Inference and abstraction of the biometric passport, May 2010.
3. ALDEBARAN manual. `http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html`.
4. R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR '98*, LNCS 1466, pages 163–178. Springer, 1998.
5. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

6. T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen. On the correspondence between conformance testing and regular inference. In *FASE 2005*, LNCS 3442, pages 175–189. Springer, 2005.
7. 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.
8. L. de Alfaro and T. Henzinger. Interface automata. In *ESEC/FSE-01*, *Software Engineering Notes 26*, pages 109–120, 2001. ACM Press.
9. T. Henzinger. Two challenges in embedded systems design: Predictability and robustness. *Philosophical Trans. of the Royal Society A*, 366:3727–3736, 2008.
10. C. d. Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, Apr. 2010.
11. H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *CAV 2003*, LNCS 2725, pages 315–327. Springer, 2003.
12. ICAO. Doc 9303 - machine readable travel documents - part 1-2. Technical report, International Civil Aviation Organization, 2006. Sixth edition.
13. B. Jonsson. Modular verification of asynchronous networks. In PODC'87, pages 152–166.
14. K. Larsen, U. Nyman, and A. Wasowski. Modal i/o automata for interface and product line theories. In *ESOP 2007*, LNCS 4421, pages 64–79. Springer, 2007.
15. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines — a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
16. M. Leucker. Learning meets verification. In *FMCO 2006*, LNCS 4709, pages 127–151. Springer, 2006.
17. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.
18. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In PODC'87, pages 137–151.
19. W. Mostowski, E. Poll, J. Schmaltz, J. Tretmans, and R. Wichers Schreur. Model-based testing of electronic passports. In *FMICS '09*, pages 207–209, 2009. Springer.
20. O. Niese. *An Integrated Approach to Testing Complex Systems*. PhD thesis, University of Dortmund, 2003.
21. P. Panangaden and E. Stark. Computations, residuals, and the power of indeterminancy. In *ICALP88*, LNCS 317 , pages 439–454. Springer, 1988.
22. H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05*, pages 62–71, 2005. ACM Press.
23. H. Raffelt, B. Steffen, T. Berg, and T. Margaria. Learnlib: a framework for extrapolating behavioral models. *STTT*, 11(5):393–407, 2009.
24. J. Rosenberg and H. Schulzrinne. Reliability of Provisional Responses in Session Initiation Protocol (SIP). RFC 3262 (Proposed Standard), June 2002.
25. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.
26. J. Tretmans. Test generation with inputs, outputs, and repetitive quiescence. *Software–Concepts and Tools*, 17:103–120, 1996.
27. J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, LNCS 4949, pages 1–38. Springer, 2008.
28. M. Veanes and N. Bjørner. Input-output model programs. In *ICTAC 2009*, LNCS 5684, pages 322–335. Springer, 2009.
29. T. Willemse. Heuristics for ioco-based test-based modelling. In *FMICS and PDMC 2006*, LNCS 4346, pages 132–147. Springer, 2007.