

Regular Inference for State Machines using Domains with Equality Tests

Therese Berg¹, Bengt Jonsson¹, Harald Raffelt²

¹ Department of Computer Systems, Uppsala University, Sweden
`{thereseb, bengt}@it.uu.se`

² Chair of Programming Systems and Compiler Construction, University of
Dortmund, Germany
`harald.raffelt@cs.uni-dortmund.de`

Abstract. Existing algorithms for regular inference (aka automata learning) allows to infer a finite state machine by observing the output that the machine produces in response to a selected sequence of input strings. We generalize regular inference techniques to infer a class of state machines with an infinite state space. We consider Mealy machines extended with state variables that can assume values from a potentially unbounded domain. These values can be passed as parameters in input and output symbols, and can be used in tests for equality between state variables and/or message parameters. This is to our knowledge the first extension of regular inference to infinite-state systems. We intend to use these techniques to generate models of communication protocols from observations of their input-output behavior. Such protocols often have parameters that represent node addresses, connection identifiers, etc. that have a large domain, and on which test for equality is the only meaningful operation. Our extension consists of two phases. In the first phase we apply an existing inference technique for finite-state Mealy machines to generate a model for the case that the values are taken from a small data domain. In the second phase we transform this finite-state Mealy machine into an infinite-state Mealy machine by folding it into a compact symbolic form.

1 Introduction

Model-based techniques for verification and validation of reactive systems, such as model checking and model-based test generation [1] have witnessed drastic advances in the last decades. They depend on the availability of a formal model, specifying the intended behavior of a system or component, which ideally should be developed during specification and design. However, in practice often no such model is available, or becomes outdated as the system evolves over time, implying that a large effort in many model-based verification and test generation projects is spent on manually constructing a model from an implementation. It is therefore important to develop techniques for automating the task of generating models of existing implementations. A potential approach is to use program analysis to construct models from source code, as in software verification (e.g.,

[2, 3]). However, many system components, including peripheral hardware components, library modules, or third-party components do not allow analysis of source code. We will therefore focus on techniques for constructing models from observations of their external behavior.

The construction of models from observations of component behavior can be performed using regular inference (aka automata learning) techniques [4–10]. This class of techniques has recently started to get attention in the testing and verification community, e.g., for regression testing of telecommunication systems [11, 12], and for combining conformance testing and model checking [13, 14]. They describe how to construct a finite-state machine (or a regular language) from the answers to a finite sequence of *membership queries*, each of which observes the component’s output in response to a certain input string. Given “enough” membership queries, the constructed automaton will be a correct model of the system under test (SUT). Angluin [4] and others introduce *equivalence queries*, queries to whether a hypothesized automaton is a correct model of the SUT, they can be seen as idealizing some procedure for extensively verifying (e.g., by conformance testing) whether the learning procedure is completed. The reply to an equivalence query is either *yes* or a counterexample, an input string on which the constructed automaton and the SUT respond with different output.

We intend to use regular inference to construct models of communication protocol entities. Such entities typically communicate by messages that consist of a protocol data unit (PDU) type with a number of parameters, each of which ranges over a sometimes large domain. Standard regular inference can only construct models with a moderately large finite alphabet. In previous work [15], we presented an optimization of regular inference to cope with models where the domain over which parameters range is large but finite. But, in order to fully support the generation of models with data parameters, we must consider a general theory for inference of infinite-state state machines with input and output symbols from potentially infinite domains.

In this paper, we present the first extension of regular inference to infinite-state state machines. We consider Mealy machines where input and output symbols are constructed from a finite number of message types that can have parameters from a potentially infinite domain. These parameters can be stored in state variables of the machine for later use. The only allowed operation on parameter values is a test for equality. The motivation is to handle parameters that, e.g., are identifiers of connections, objects, etc. This class of systems is similar to, and slightly more expressive than, the class of “data-independent” systems, which was the subject of some of the first works on model checking of infinite-state systems [16, 17].

In standard regular inference control states and transitions are inferred, and counterexamples to a hypothesized automaton are only used to add more control states to the automaton. In this paper, we also infer state variables and operations on them, and counterexamples to a hypothesized model are used to extend the model with either more control states or more state variables.

In our approach, we first observe the behavior of the protocol when the parameters of input messages are from a small domain. Using the regular inference algorithm by Niese [18] (which adapts Angluin’s algorithm to Mealy machines), we generate a finite-state Mealy machine, which describes the behavior of the component on this small domain. We thereafter fold this finite-state Mealy machine into a smaller *symbolic* model.

Organization. The paper is organized as follows. In the next section, we review the Mealy machine model and in Section 3 we introduce the model for state machines using domains with equality tests. In Section 4 we review the inference algorithm for Mealy machines by Niese [18], and the adaptation required for our setting. In Section 5 we present our algorithm to map Mealy machines to Symbolic Mealy machines. Correctness and complexity of our algorithm is discussed in Section 6, and conclusions and future work are presented in Section 7.

Related Work. Regular inference techniques have been used for verification and test generation, e.g., to create models of environment constraints with respect to which a component should be verified [19], for regression testing to create a specification and a test suite [11, 12], to perform model checking without access to code or to formal models [14, 13], for program analysis [20], and for formal specification and verification [19]. Li, Groz, and Shahbaz [21, 22] extend regular inference to Mealy machines with a finite subset of input and output symbols from the possible infinite set of symbols. This work resembles the intermediate model, in our earlier work [15], used to construct a symbolic model. In this work we handle infinite sets of parameter values, and can also generate infinite-state models with both control states and state variables that range over potentially infinite domains.

2 Mealy Machines

A *Mealy machine* is a tuple $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$ where Σ_I is a nonempty set of *input symbols*, Σ_O is a finite nonempty set of *output symbols*, Q is a nonempty set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma_I \rightarrow Q$ is the *transition function*, and $\lambda : Q \times \Sigma_I \rightarrow \Sigma_O$ is the *output function*. Elements of Σ_I^* and Σ_O^* are (input and output, respectively) *strings* or *words*. Given $u, v \in \Sigma_I^*$, u is said to be a *prefix* of v if $v = uw$ for some $w \in \Sigma_I^*$.

An intuitive interpretation of a Mealy machine is as follows. At any point in time, the machine is in one state $q \in Q$. It is possible to give inputs to the machine, by supplying an input symbol a . The machine responds by producing an output string $\lambda(q, a)$ and transforming itself to the new state $\delta(q, a)$.

We extend the transition and output functions from input symbols to sequences of input symbols in the standard way, by defining:

$$\begin{aligned} \delta(q, \varepsilon) &= q & \lambda(q, \varepsilon) &= \varepsilon \\ \delta(q, ua) &= \delta(\delta(q, u), a) & \lambda(q, ua) &= \lambda(q, u)\lambda(\delta(q, u), a) \end{aligned}$$

The Mealy machines that we consider are *completely specified*, meaning that at every state the machine has a defined reaction to every input symbol in Σ_I , i.e., δ and λ are total. They are also *deterministic*, meaning that for each state q and input a exactly one next state $\delta(q, a)$ and output string $\lambda(q, a)$ is possible.

Let q and q' be two states of the same Mealy machine with input symbols Σ_I and output function λ , or two states in different machines. The states q and q' are *equivalent* if $\lambda(q, u) = \lambda(q', u)$ for each input string $u \in \Sigma_I^*$. That is, for each input string the machine starting in q will produce the same output string as the machine starting in q' . A Mealy machine \mathcal{M} is *minimized* if there are no pair of states q and q' of \mathcal{M} , where $q \neq q'$, that are equivalent. There are well-known algorithms for efficiently minimizing a given Mealy machine [23]. Given a Mealy machine \mathcal{M} with input alphabet Σ_I , initial state q_0 , and output function λ we define $\lambda_{\mathcal{M}}(u) = \lambda(q_0, u)$, for $u \in \Sigma_I^*$. Two Mealy machines \mathcal{M} and \mathcal{M}' with input alphabets Σ_I are *equivalent* if $\lambda_{\mathcal{M}} = \lambda_{\mathcal{M}'}$.

3 Symbolic Mealy Machines

In this section, we introduce Symbolic Mealy machines. They extend ordinary Mealy machines in that input and output symbols are messages with parameters, e.g., as in a typical communication protocol. We will specialize to the case when these parameters are from a large (in practice “infinite”) domain \mathcal{D} , on which the only permitted operation is test for equality. In general, we could have several such domains, but let us here assume that all parameters are from one domain.

Let I and O be finite sets of (input and output) *actions*, each of which has a nonnegative arity. Let \mathcal{D} be a (finite or infinite) domain of data values. Let $\Sigma_I^{\mathcal{D}}$ be the set of *input symbols* of form $\alpha(d_1, \dots, d_n)$, where $\alpha \in I$ is an action of arity n , and $d_1, \dots, d_n \in \mathcal{D}$ are parameters from \mathcal{D} . The set of *output symbols* $\Sigma_O^{\mathcal{D}}$ is defined analogously.

We assume a set of *location variables*, ranged over by v, v_1, v_2, \dots , and a set of *formal parameters*, ranged over by p, p_1, p_2, \dots . A *symbolic value* is either a location variable or a formal parameter. We use y or z to range over symbolic values. A *parameterized input action* is a term of form $\alpha(p_1, \dots, p_n)$, where α is an input action of arity n , and p_1, \dots, p_n are formal parameters. A *parameterized output action* is a term of form $\beta(z_1, \dots, z_k)$, where $\beta \in O$ is an output action of arity k and each z_i is a symbolic value. We write \bar{d} for d_1, \dots, d_n , \bar{p} for p_1, \dots, p_n , \bar{v} for v_1, \dots, v_m , \bar{y} for y_1, \dots, y_m , and \bar{z} for z_1, \dots, z_k . A *guard* over the location variables \bar{v} and formal parameters \bar{p} is a conjunction of equalities and inequalities between the symbolic values in \bar{v}, \bar{p} . A *guarded action* is of the form

$$\alpha(\bar{p}); g/\bar{v} := \bar{y}; \beta(\bar{z})$$

where $\alpha(\bar{p})$ is a parameterized input action, g is a guard over location variables and the formal parameters \bar{p} , where $\bar{v} := \bar{y}$ is the multiple assignment statement $\langle v_1, \dots, v_m \rangle := \langle y_1, \dots, y_m \rangle$, meaning that y_i is assigned to v_i , in which each y_i is a symbolic value, and $\beta(\bar{z})$ is a parameterized output action. An example of a guarded action is $\alpha(p_1, p_2); true/v_1 := p_1; \beta(p_2)$, where the first input parameter

value is stored in a location variable, and the second input parameter value is output.

Let a *valuation function* ρ be a mapping from a (possibly empty) set $Dom(\rho)$ of location variables to data values in \mathcal{D} . We let ρ_0 denote the valuation function with an empty domain. We extend valuation functions to operate on vectors of location variables by defining $\rho(v_1, \dots, v_m)$ as $\rho(v_1), \dots, \rho(v_m)$. For an input symbol $\alpha(\bar{d})$, a valuation function ρ , and guard g such that $v_i \in Dom(\rho)$ for all location variables v_i occurring in g , we let $(\alpha(\bar{d}), \rho) \models g$ denote that $g[\bar{d}/\bar{p}, \rho(\bar{v})/\bar{v}]$ is true, i.e., that g is true when the formal parameters \bar{p} assume the data values \bar{d} and the location variables \bar{v} assume the data values $\rho(\bar{v})$.

Definition 1 (Symbolic Mealy machine). A Symbolic Mealy machine is a tuple $\mathcal{SM} = (I, O, L, l_0, \longrightarrow)$, where

- I is a finite set of input actions,
- O is a finite set of output actions,
- L is a finite set of locations, each of which has a nonnegative arity representing the number of location variables in that location,
- $l_0 \in L$ is the initial location with arity 0, and
- \longrightarrow is a finite set of transitions. Each transition is a tuple $\langle l, \alpha(\bar{p}), g, \bar{v} := \bar{y}, \beta(\bar{z}), l' \rangle$, where $l, l' \in L$ are locations and $\alpha(\bar{p}); g/\bar{v} := \bar{y}; \beta(\bar{z})$ is a guarded action respecting constraints given by arities of l and l' . \square

We write $l \xrightarrow{\alpha(\bar{p}); g/\bar{v} := \bar{y}; \beta(\bar{z})} l'$ to denote $\langle l, \alpha(\bar{p}), g, \bar{v} := \bar{y}, \beta(\bar{z}), l' \rangle \in \longrightarrow$. We require that Symbolic Mealy machines are completely specified and deterministic, i.e., for each reachable $\langle l, \rho \rangle$ and input symbol $\alpha(\bar{d})$, there is exactly one transition $\langle l, \alpha(\bar{p}), g, \bar{v} := \bar{y}, \beta(\bar{z}), l' \rangle$ from l such that $(\alpha(\bar{d}), \rho) \models g$.

Example An example of a Symbolic Mealy machine is shown in Figure 1. The Symbolic Mealy machine has one input action α with arity two, two output actions β and β' , both with arity one, one location l_0 with arity zero, and a second location l_1 with arity one.

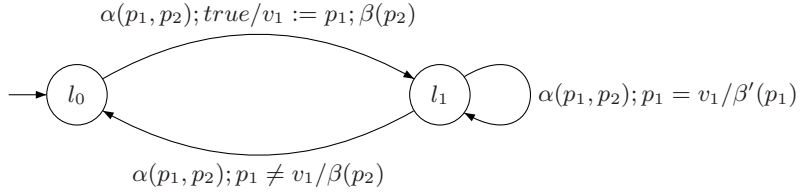


Fig. 1. Example of a Symbolic Mealy machine.

Let \mathcal{D} be a (finite or infinite) domain of data values. A Symbolic Mealy machine $\mathcal{SM} = (I, O, L, l_0, \longrightarrow)$ acting on the domain \mathcal{D} is the (possibly infinite-state) Mealy machine $\mathcal{SM}_{\mathcal{D}} = \langle \Sigma_I^{\mathcal{D}}, \Sigma_O^{\mathcal{D}}, Q, \langle l_0, \rho_0 \rangle, \delta, \lambda \rangle$, where

- $\Sigma_I^{\mathcal{D}}$ is the set of input symbols,
- $\Sigma_O^{\mathcal{D}}$ is the set of output symbols,
- Q is the set of pairs $\langle l, \rho \rangle$, where $l \in L$ and ρ is a valuation function which domain v_1, \dots, v_m is given by the arity m of location l ,
- $\langle l_0, \rho_0 \rangle$ is the initial state, and
- δ and λ are defined as follows. Whenever $l \xrightarrow{\alpha(\bar{p}); g/\bar{v}:=\bar{y}; \beta(\bar{z})} l'$ and $(\alpha(\bar{d}), \rho) \models g$ then
 - $\delta(\langle l, \rho \rangle, \alpha(\bar{d})) = \langle l', \rho' \rangle$, where $\rho'(v_i)$ is
 - * $\rho(v_j)$ if y_i is a location variable v_j ,
 - * d_j if y_i is a formal parameter p_j , and
 - $\lambda(\langle l, \rho \rangle, \alpha(\bar{d})) = \beta(d'_1, \dots, d'_k)$, where d'_i is
 - * $\rho(v_j)$ if z_i is a location variable v_j ,
 - * d_j if z_i is a formal parameter p_j .

Note that δ is well-defined since \mathcal{SM} is completely specified and deterministic.

If \mathcal{D} is finite, then $\mathcal{SM}_{\mathcal{D}}$ is a finite-state Mealy machine with at most $\sum_{l \in L} |\mathcal{D}|^{\text{arity}(l)}$ states, where $\text{arity}(l)$ is the arity of location l . Two Symbolic Mealy machines \mathcal{SM} and \mathcal{SM}' are *equivalent* if $\mathcal{SM}_{\mathcal{D}}$ is equivalent to $\mathcal{SM}'_{\mathcal{D}}$ for any data value domain \mathcal{D} .

Let \mathcal{M} be a Mealy machine with the set of input symbols $\Sigma_I^{\mathcal{D}}$, the set of states Q , initial state $q_0 \in Q$, and output function λ . For two data values d and d' in domain \mathcal{D} and a string $u \in (\Sigma_I^{\mathcal{D}})^*$, let $u[d \leftrightarrow d']$ denote the result of swapping the values d and d' in the string u . For each state $q \in Q$ define an equivalence relation on data values, denoted \simeq_q by $d \simeq_q d'$ if $\lambda(q, u[d \leftrightarrow d']) = \lambda(q, u)$. Intuitively, $d \simeq_q d'$ means that the data values d and d' are treated symmetrically from state q . We say that \mathcal{M} treats d and d' symmetrically if $d \simeq_{q_0} d'$, i.e., d and d' are treated symmetrically from the initial state. We say that \mathcal{M} is *symmetric* if $d \simeq_{q_0} d'$ for any two data values $d, d' \in \mathcal{D}$.

4 Inference of Symbolic Mealy Machines

In this section, we present our algorithm for inference of Symbolic Mealy machines. It is formulated in the same setting as Angluin's L^* algorithm [4], in which a so called *Learner*, who initially knows nothing about \mathcal{SM} , is trying to infer \mathcal{SM} by asking queries to a so called *Oracle*. The queries are of two kinds.

- A *membership query* consists in asking what the output is on a word $w \in (\Sigma_I^{\mathcal{E}})^*$ for some data value domain \mathcal{E} .
- An *equivalence query* consists in asking whether a hypothesized Symbolic Mealy machine \mathcal{H} is correct, i.e., whether \mathcal{H} is equivalent to \mathcal{SM} . The *Oracle* will answer *yes* if \mathcal{H} is correct, or else supply a *counterexample*, which is a word $u \in (\Sigma_I^{\mathcal{D}})^*$ such that $\lambda_{\mathcal{SM}_{\mathcal{D}}}(u) \neq \lambda_{\mathcal{H}_{\mathcal{D}}}(u)$, for some data value domain \mathcal{D} .

The typical behavior of a *Learner* is to start by asking a sequence of membership queries until she can build a “stable” hypothesis \mathcal{H} from the answers. After that she makes an equivalence query to find out whether \mathcal{H} is equivalent to \mathcal{SM} . If the result is successful, the *Learner* has succeeded, otherwise she uses the returned counterexample to perform subsequent membership queries until converging at a new hypothesized Symbolic Mealy machine, which is supplied in an equivalence query, etc.

We start by fixing a small, finite data value domain \mathcal{E} . In our algorithm for the *Learner*, we build a hypothesis \mathcal{H} in two phases: In the first phase we supply input from the domain \mathcal{E} , and infer a hypothesis \mathcal{M} for the Mealy machine $\mathcal{SM}_{\mathcal{E}}$ using an adaptation of Angluin’s algorithm due to Niese [18]. When building \mathcal{M} , we must treat all data values symmetrically in the following sense: for all performed membership queries $w \in (\Sigma_I^{\mathcal{E}})^*$, we also make a membership query for $w[d \leftrightarrow d']$ for any data values $d, d' \in \mathcal{E}$. This implies that we also supply membership queries for all permutations of data values in \mathcal{E} in w . This implies that the resulting hypothesis \mathcal{M} will be symmetric.

There is an optimal smallest size of \mathcal{E} which is still large enough to be able to exercise all tests for equalities between parameter values in \mathcal{SM} , and find the data values required to be stored in location variables of the constructed Symbolic Mealy machine in order for it to be deterministic, but we do not *a priori* know this size. Therefore, we start with a small domain \mathcal{E} that can be gradually extended when feedback from equivalence queries show that \mathcal{E} is too small.

To characterize the “optimal smallest size”, let \mathcal{M} be $\langle \Sigma_I^{\mathcal{E}}, \Sigma_O^{\mathcal{E}}, Q, q_0, \delta, \lambda \rangle$, where I, O are the input and output action alphabet, respectively. We say that a data value $d \in \mathcal{E}$ is *fresh* in a state $q \in Q$ if

- there exists an input string $u \in (\Sigma_I^{\mathcal{E}})^*$ which leads to $\delta(q_0, u) = q$ such that d does not occur as a data value in u , and
- for every input string $v \in (\Sigma_I^{\mathcal{E}})^*$ which leads to q and in which d but not some other $d' \in \mathcal{E}$ occurs as data value, it holds that $\delta(q_0, v[d'/d]) = q$.

The first item intuitively means that if a data value does not occur in an input string leading to a state then it cannot either be saved in a location corresponding to that state. The second item intuitively means that if a data value in a string can be replaced by another data value and the string still leads to the same state, then it has no special affect in the Mealy machine and is therefore not stored in a location corresponding to that state. We say that \mathcal{M} is *fresh* if for each state $q \in Q$, there are n data values in \mathcal{E} which are fresh in q , where n is the maximal arity of input actions in I . A sufficient condition on \mathcal{E} is given by the following Lemma 1.

Lemma 1. *Let $\mathcal{SM} = (I, O, L, l_0, \longrightarrow)$ be a Symbolic Mealy machine, \mathcal{E} be a domain of data values, m be the maximal arity of the locations in L , and n be the maximal arity of the input actions in I . Then if the size of \mathcal{E} is bigger than $m + \max(n, m)$, the Mealy machine $\mathcal{SM}_{\mathcal{E}}$ is fresh.*

Proof. Let $\mathcal{SM}_\mathcal{E}$ be $\langle \Sigma_I^\mathcal{E}, \Sigma_O^\mathcal{E}, Q, \langle l_0, \rho_0 \rangle, \delta, \lambda \rangle$ and let \mathcal{E} be bigger than $m + \max(n, m)$. Let \mathcal{M} be the minimal version of $\mathcal{SM}_\mathcal{E}$, which we get by partitioning Q with respect to an equivalence relation \cong , where two states are equivalent $q \cong q'$ iff they produce the same output $\lambda(q, u) = \lambda(q', u)$ for all input strings $u \in (\Sigma_I^\mathcal{E})^*$. The states of \mathcal{M} is the set Q/\cong of equivalence classes of Q . Let us call \mathcal{M} 's set of states $Q_\mathcal{M}$, its transition function $\delta_\mathcal{M}$, and its output function $\lambda_\mathcal{M}$. In \mathcal{M} the initial state is $[\langle l_0, \rho_0 \rangle]$, the transition function is $\delta([q], a) = [\delta(q, a)]$, and the output function is $\lambda([q], a) = \lambda(q, a)$. We will show that $\mathcal{SM}_\mathcal{E}$ is fresh, i.e. that in each $\langle l, \rho \rangle \in Q$ there are n data values in \mathcal{E} that satisfies the two conditions in order to be fresh.

First we will show that for each state $\langle l, \rho \rangle \in Q$ and each d_i there exists an input string $u_i \in (\Sigma_I^\mathcal{E})^*$ which leads to $\delta(\langle l_0, \rho_0 \rangle, u_i) = \langle l, \rho \rangle$ such that data value d_i does not occur in u_i , where $i = 1, 2, \dots, n$. It follows that the same holds for all states in $Q_\mathcal{M}$.

Let us assume that the size of \mathcal{E} is larger than $m + n$. Let $Range(\rho)$ be the range of ρ . We will show by induction on the distance from $\langle l_0, \rho_0 \rangle$ that if $\langle l, \rho \rangle$ is reachable and data value $d \in \mathcal{E}$ is not in $Range(\rho)$, then we can reach $\langle l, \rho \rangle$ with an input string in which d does not occur. Since there are at most m data values in $Range(\rho)$ there exists at least $n + 1$ such data values d in $\langle l, \rho \rangle$.

Assume that $\langle l', \rho' \rangle \in Q$ is at distance $k + 1$ from $\langle l_0, \rho_0 \rangle$. Then there exists a state $\langle l, \rho \rangle$ at distance k from $\langle l_0, \rho_0 \rangle$ such that $\langle l, \rho \rangle \xrightarrow{\alpha(\bar{d}^I)/\beta(\bar{d}^O)} \langle l', \rho' \rangle$ is a transition in $\mathcal{SM}_\mathcal{E}$. By induction hypothesis, for each data value d not in $Range(\rho)$ we can reach $\langle l, \rho \rangle$ with an input string in which d does not occur. Let $Range(\rho')$ be the range of ρ' . At most m data values are in $Range(\rho)$, and at most n data values are used in $\alpha(\bar{d}^I)$ on the transition to $\langle l', \rho' \rangle$, which leaves at least one data value in \mathcal{E} not used. Let us call this data value d , and let u be an input string of length k leading to $\langle l, \rho \rangle$ and having no occurrences of d in it. Then data value d does not occur in the input string $u\alpha(\bar{d}^I)$ leading to $\langle l', \rho' \rangle$, hence d cannot be in $Range(\rho')$. Since there are at most m data values in $Range(\rho')$ there are least $n + 1$ such data values d in each $\langle l', \rho' \rangle \in Q$.

For each $q \in Q_\mathcal{M}$ we define \simeq_q to be an equivalence relation on \mathcal{E} such that $e \simeq_q e'$ iff the behavior in q is completely symmetric with respect to e, e' , i.e., $\lambda_\mathcal{M}(q, w)[e' \leftrightarrow e] = \lambda_\mathcal{M}(q, w[e' \leftrightarrow e])$ for all $w \in (\Sigma_I^\mathcal{E})^*$, where $w[e' \leftrightarrow e]$ is the result of switching all occurrences of e' to e , and vice versa, in the string w . It follows that all data values not in $Range(\rho)$ are equivalent in state $[\langle l, \rho \rangle]$ due to symmetry.

Let us assume that state $q \in Q_\mathcal{M}$ contains a state $\langle l, \rho \rangle \in Q$, and some input string $u \in (\Sigma_I^\mathcal{E})^*$ that leads to q does not contain occurrences of data value $d \in \mathcal{E}$. It was shown in the first part of this proof that such a d and u for state $\langle l, \rho \rangle$ can be found if $d \notin Range(\rho)$. Let us also assume that there is an input string $v \in (\Sigma_I^\mathcal{E})^*$ such that d occurs in v , another data value $d' \in \mathcal{E}$ does not occur in v , and it leads to a state $\langle l', \rho' \rangle \in Q$ in q , i.e., $[\langle l, \rho \rangle] = [\langle l', \rho' \rangle]$ in $Q_\mathcal{M}$. Next, we will show that whenever v leads to $\delta(\langle l_0, \rho_0 \rangle, v) = \langle l', \rho' \rangle$, then the string v in which d is replaced by d' leads to $\delta(\langle l_0, \rho_0 \rangle, v[d'/d]) = \langle l', \rho' \rangle$.

Assume that the size of \mathcal{E} is bigger than $2m$. We have that $d' \notin \text{Range}(\rho')$ since d' does not occur as data value in v . In state $\langle l', \rho' \rangle$ data value d' is in an equivalence class induced by $\simeq_{\langle l', \rho' \rangle}$ of at least size $|\mathcal{E}| - m$, since the equivalence class contains the data values \mathcal{E} except the at most m data values in $\text{Range}(\rho')$. In state $\langle l, \rho \rangle$ data value d is in an equivalence class of at least size $|\mathcal{E}| - m$ induced by \simeq_q . Since $[\langle l, \rho \rangle] = [\langle l', \rho' \rangle]$ and both d and d' is in the largest equivalence class of at least size $|\mathcal{E}| - m$ in state q the data values are in the same equivalence class $d \simeq_q d'$. Hence, d is not in $\text{Range}(\rho')$, and therefore $v[d'/d]$ leads to $\delta(\langle l_0, \rho_0 \rangle, v[d'/d]) = \langle l', \rho' \rangle$ in Q , and to q in $Q_{\mathcal{M}}$. \square

Returning to our inference algorithm, if the hypothesis \mathcal{M} generated by Niese's algorithm is not fresh, we do not make any equivalence query, but instead enlarge \mathcal{E} by one, and continue Niese's algorithm with the new enlarged \mathcal{E} . If \mathcal{M} is fresh, we transform \mathcal{M} into a Symbolic Mealy machine \mathcal{H} such that $\mathcal{H}_{\mathcal{E}}$ is equivalent to \mathcal{M} : this transformation is presented in Section 5. Thereafter \mathcal{H} is supplied in an equivalence query, to find out whether \mathcal{H} is equivalent to \mathcal{SM} . If the result is successful, the algorithm terminates. Otherwise, the counterexample returned needs to be analyzed, since it may contain values outside of \mathcal{E} . Let the counterexample be input string $u \in \Sigma_I^{\mathcal{D}}$, with data values from domain \mathcal{D} .

In the case that $|\mathcal{D}| \leq |\mathcal{E}|$, we apply any injective mapping from \mathcal{D} to \mathcal{E} , on the data values in the counterexample, and use the mapped counterexample in Niese's algorithm.

In the case that $|\mathcal{D}| > |\mathcal{E}|$, we try to find a mapping from \mathcal{D} to a subset $\mathcal{D}' \subseteq \mathcal{D}$ which is as small as possible, but such that the mapped counterexample is still a counterexample to \mathcal{H} . To find such a mapping, one can either use an exhaustive search or a heuristic search guided by \mathcal{H} . The search involves asking more membership queries. We then extend \mathcal{E} to the same size as \mathcal{D}' , and continue Niese's algorithm with the mapped counterexample.

5 Transforming Mealy Machines to Symbolic Mealy Machines

In this section, we present the transformation from a Mealy machine \mathcal{M} to a Symbolic Mealy machine \mathcal{SM} . Throughout this section, we will use the Symbolic Mealy machine in Figure 1 as a running example to illustrate the steps in our algorithm. We assume that we used the domain $\mathcal{E} = \{1, 2, 3\}$ of size 3, which is the smallest domain to make $\mathcal{SM}_{\mathcal{E}}$ fresh, and that we obtained the hypothesis Mealy machine shown in Figure 2.

In the second phase, we transform the Mealy machine \mathcal{M} into a Symbolic Mealy machine \mathcal{SM} , which must "simulate" \mathcal{M} in the sense that $\mathcal{SM}_{\mathcal{E}}$ is equivalent to \mathcal{M} . The transformation algorithm has four steps.

- In the first step, the algorithm figures out for each state of \mathcal{M} which data values must be remembered by a corresponding Symbolic Mealy machine in order to produce its future behavior. These data values are the basis for constructing the location variables required in the corresponding location in the

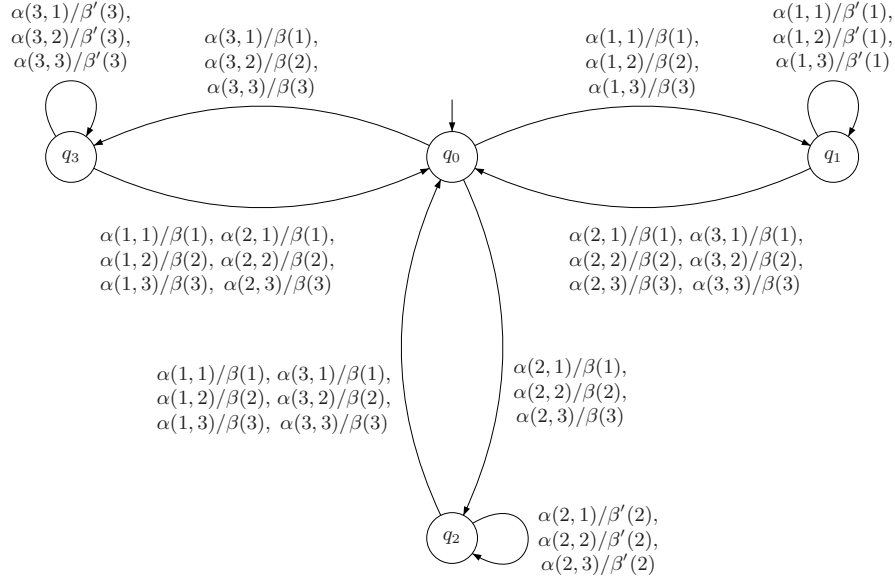


Fig. 2. A Mealy machine constructed with the inference algorithm applied to the example in Figure 1. All transitions that have the same start and target state are depicted with one edge.

Symbolic Mealy machine. In the running example of Figure 2, the algorithm figures out that, e.g., in state q_1 the data value 1 must be remembered.

- In the second step, we use the data values inferred in the first step to transform transitions of \mathcal{M} into a so called symbolic normal form, which is especially designed to capture exactly the equalities and inequalities between formal parameters and location variables. In the running example, the transition on $\alpha(1,2)/\beta'(1)$ from, e.g., location q_1 , will be transformed into $\alpha(v_1, p_1)/v_1 := v_1; \beta'(v_1)$.
- In the third step, we merge states of \mathcal{M} into locations of \mathcal{SM} , if the symbolic forms of their future behavior are the same, using an adaptation of a standard partition-refinement algorithm. In the running example, states q_1, q_2 , and q_3 will be merged into one location.
- In the fourth and final step, we transform transitions from symbolic normal form to the standard form used in Definition 1, and merge transitions when possible. The result of the first part of this step is shown in Figure 4.

Step One The first step in our algorithm is to compute a *state-value function* $V : Q \mapsto 2^{\mathcal{E}}$ from states in \mathcal{M} to sets of data values, which for each state gives the set of data values that \mathcal{M} must remember for its future behavior. We first observe that if the data value d is fresh in a state q , then obviously d should not be in $V(q)$. Furthermore, if a data value d' is not remembered by \mathcal{M} in q ,

then the future behavior from q remains unchanged if we swap the roles of d and d' (note that d and d' will occur in future input symbols). Let u be the input string which takes \mathcal{M} from its initial state to q and does not contain d . The input string u' , obtained by replacing all occurrences of d' by d in u will, by symmetry, take \mathcal{M} from its initial state to a state q' whose future behavior has swapped the roles of d and d' as compared with q . Since \mathcal{M} is minimized, q must be the same state as q' . Thus d' is also in fresh in q . In summary, $V(q)$ should contain all data values that are not fresh in q .

The calculation of the state-value function in the example in Figure 2 yields that $V(q_0)$ is the empty set \emptyset , $V(q_1)$ is $\{1\}$, $V(q_2)$ is $\{2\}$, and $V(q_3)$ is $\{3\}$.

Step Two In the second step, we transform transition labels into a symbolic form.

Intuitively, the transition $q \xrightarrow{\alpha(\bar{d}^I)/\beta(\bar{d}^O)} q'$ will be transformed into a symbolic transition $q \xrightarrow{\alpha(\bar{v});g/\bar{v}:=\bar{y};\beta(\bar{z})} q'$ where g is as strong as possible. We will use a different representation for such a symbolic transition, which we call *symbolic normal form*, which does not use guards: instead each data value in \bar{d}^I and \bar{d}^O will be replaced by v_i if it occurs as the i th data value stored in state q , otherwise by an appropriate input parameter symbol p_j . Furthermore, different symbolic values are implicitly required to be different. A problem is that we cannot know in which “order” the data values in $V(q)$ will be mapped to location variables, so therefore the transformation will depend on a specific ordering \bar{d} of the data values in $V(q)$ and a specific ordering \bar{d}' of the data values in $V(q')$.

Let us define precisely the symbolic normal form. For each ordering \bar{d} of the data values in $V(q)$, for each vector \bar{d}^I of data values received in an input symbol, and for each data value d in $\bar{d} \cup \bar{d}^I$, define $SV_{\bar{d},\bar{d}^I}(d)$ as

- v_i if there exists $d_i \in \bar{d}$ such that $d = d_i$, or else
- p_k if j is the smallest index such that $d_j \in \bar{d}^I$ is $d_j = d$, and k is the number of unique data values $d_l \in \bar{d}^I$ with index $l \leq j$, such that d_l does not appear in \bar{d} ,

We extend $SV_{\bar{d},\bar{d}^I}$ to vectors of data values, by defining $SV_{\bar{d},\bar{d}^I}(d_1, \dots, d_n)$ as $SV_{\bar{d},\bar{d}^I}(d_1), \dots, SV_{\bar{d},\bar{d}^I}(d_n)$.

For each ordering \bar{d} of the data values in $V(q)$ and each ordering \bar{d}' of the data values in $V(q')$, the *symbolic normal form* of $q \xrightarrow{\alpha(\bar{d}^I)/\beta(\bar{d}^O)} q'$ is defined as

$$(q, \bar{d}) \xrightarrow{\alpha(SV_{\bar{d},\bar{d}^I}(\bar{d}^I))/\bar{v}:=\bar{y};\beta(SV_{\bar{d},\bar{d}^I}(\bar{d}^O))} (q', \bar{d}'),$$

where $\bar{v} := \bar{y}$ is an assignment in which y_i is

- $SV_{\bar{d},\bar{d}^I}(d_j)$ if $d'_i = d_j$, for some $d'_i \in \bar{d}'$ and $d_j \in \bar{d}$, or else
- $SV_{\bar{d},\bar{d}^I}(d_j^I)$ if $d'_i = d_j^I$, for some $d'_i \in \bar{d}'$ and $d_j^I \in \bar{d}^I$.

As an example, the symbolic normal form of $q_0 \xrightarrow{\alpha(1,2)/\beta(2)} q_1$, a transition in the Mealy machine in Figure 2, calculates to $(q_0, \square) \xrightarrow{\alpha(p_1, p_2)/v_1 := p_1; \beta(p_2)} (q_1, 1)$, where \square is the empty vector of data values.

Step Three In the third step, we merge states of \mathcal{M} if the symbolic forms of their future behaviors are equivalent. As explained in the description of Step two, the symbolic normal form of the behavior from a state q is defined only with respect to a given ordering \bar{d} of the data values in $V(q)$, meaning that for each state q we must fix some ordering of the stored data values. However, since some combinations of orderings allow to merge more states and obtain smaller machines than others, we shall not fix this ordering *a priori*. Instead, we create several copies of each q , one for each possible ordering of the data values in $V(q)$, and thereafter perform the merging starting with all these copies. Since at the end, we need only one pair of form (q, \bar{d}) for each q , we will prune copies of q that will create additional states, as long as at least one copy of each q remains.

Thus, our partitioning algorithm partitions pairs (q, \bar{d}) of states and data-value vectors into blocks $\mathcal{B}_1, \dots, \mathcal{B}_m$, representing potential locations in a Symbolic Mealy machine. Each block \mathcal{B}_i is a set of pairs (q, \bar{d}) , where $q \in Q$ and \bar{d} is some ordering of the data values $V(q)$. To break the symmetry between different orderings of data values, we pick for each block \mathcal{B}_i an arbitrary pair $(q, \bar{d}) \in \mathcal{B}_i$ which is called the *representative* for \mathcal{B}_i , to represent how the symbolic transitions from \mathcal{B}_i will look like. The goal of the partitioning is that each block should be consistent, as defined in the following definition.

Definition 2 (Block consistency for a block \mathcal{B}). *Let $(q, \bar{d}) \in \mathcal{B}$ be the representative for block \mathcal{B} . Block \mathcal{B} is consistent if whenever $(r, \bar{e}) \in \mathcal{B}$ there is a transition $(r, \bar{e}) \xrightarrow{\alpha(\bar{z})/\bar{v} := \bar{y}; \beta(\bar{z}')} (r', \bar{e}')$ (on symbolic normal form) iff there is a transition $(q, \bar{d}) \xrightarrow{\alpha(\bar{z})/\bar{v} := \bar{y}; \beta(\bar{z}')} (q', \bar{d}')$ (on symbolic normal form) with the same label, such that (r', \bar{e}') and (q', \bar{d}') are in the same block. \square*

We find a partitioning of pairs into consistent blocks by fix-point iteration, using a variation of the standard partition-refinement algorithm, as follows.

- Initially, for each i which is the size of $V(q)$ for some $q \in Q$, there is a block \mathcal{B}_i with all pairs (q, \bar{d}) such that \bar{d} has exactly i data values.
- Repeat the following step until convergence.
 - Pick a block \mathcal{B}_i and let (q, \bar{d}) be the representative for \mathcal{B}_i .
 - Split \mathcal{B}_i by letting a pair (r, \bar{e}) in \mathcal{B}_i remain in the block if for all $\alpha(\bar{z})$ there is a symbolic transition $(q, \bar{d}) \xrightarrow{\alpha(\bar{z})/\bar{v} := \bar{y}; \beta(\bar{z}')} (q', \bar{d}')$ from the representative (q, \bar{d}) iff there is a symbolic transition $(r, \bar{e}) \xrightarrow{\alpha(\bar{z})/\bar{v} := \bar{y}; \beta(\bar{z}')} (r', \bar{e}')$ from (r, \bar{e}) with the same label, such that (q', \bar{d}') and (r', \bar{e}') are in the same block. Let \mathcal{B}'_i be the set of all pairs (r, \bar{e}) that were originally in \mathcal{B}_i but did not pass this test.
 - Delete from \mathcal{B}'_i all pairs (r, \bar{e}) for which some other pair (r, \bar{e}') with the same state r remains in \mathcal{B}_i . It should be noted that this deletion of copies

is safe due to the strong symmetry between different orderings of data values \bar{e} in pairs of form (r, \bar{e}) .

- If \mathcal{B}'_i is thereafter non-empty, we let \mathcal{B}'_i be a new block, and choose an arbitrary member in \mathcal{B}'_i as its representative,
- The algorithm terminates when all blocks are consistent. Let $\{\mathcal{B}_1, \dots, \mathcal{B}_f\}$ denote the final set of blocks.

From the example in Figure 2 we create two consistent blocks, \mathcal{B}_0 and \mathcal{B}_1 , see Figure 3. Block \mathcal{B}_0 contains the pair (q_0, \square) , and block \mathcal{B}_1 contains the pairs $(q_1, 1)$, $(q_2, 2)$, and $(q_3, 3)$. Let us choose (q_0, \square) as the representative for block \mathcal{B}_0 , and $(q_1, 1)$ as the representative for block \mathcal{B}_1 . In the figure, from block \mathcal{B}_1 only the labels on the outgoing transitions from one of the pairs is drawn, since all pairs have the same labels on outgoing transitions.

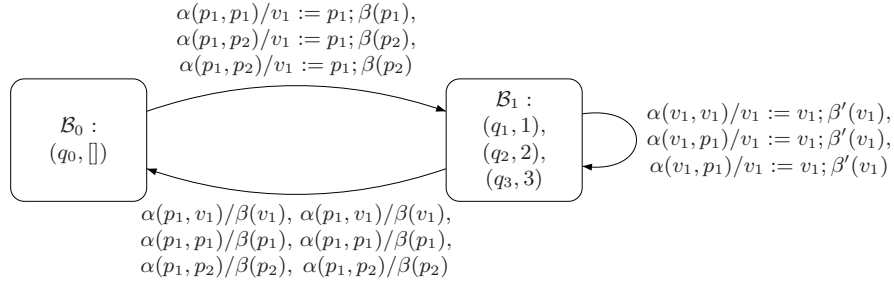


Fig. 3. The set of consistent blocks constructed from the example in Figure 2.

Step Four In the fourth step, we complete the transformation of the Mealy machine \mathcal{M} by creating a Symbolic Mealy machine \mathcal{SM} from the result of step three. This also involves transforming the symbolic normal form of transitions into the form used in Definition 1, and merging transitions. The locations of \mathcal{SM} correspond to the blocks resulting from the third step. When creating the transitions of \mathcal{SM} , we can select an arbitrary member (q, \bar{d}) in each block \mathcal{B}_i , and use only these selected members and the transitions between them to construct \mathcal{SM} . It is possible to select only one member since, by block consistency, all members are equivalent. For simplicity we let the block representative be the selected member.

Each transition between representatives must be transformed from symbolic normal form to the form used in Definition 1. As an example, consider the self-loop from the representative $(q_1, 1)$ in Figure 3 labeled by $\alpha(v_1, p_1)/v_1 := v_1; \beta'(v_1)$. It states that if an input symbol $\alpha(d_1, d_2)$ is received where d_1 is equal to the current value of the location variable v_1 , and d_2 is different from the value of any location variable, then the location variable is unchanged, and $\beta'(d_1)$ is generated. It should obtain the label $\alpha(p_1, p_2) : (p_1 = v_1 \wedge p_2 \neq v_1)/v_1 :=$

$v_1; \beta'(v_1)$, to conform with Definition 1. To define this transformation precisely, for a vector \bar{z} of symbolic values, let $T_{\bar{z}}$ be a mapping from symbolic values to symbolic values, defined as

- $T_{\bar{z}}(v_j) = v_j$ for any location variable v_j ,
- $T_{\bar{z}}(p_j) = p_i$ if p_j is a formal parameter which occurs in \bar{z} , and i is the smallest value such that p_j is the i th element z_i in \bar{z} .

$T_{\bar{z}}$ is extended to vectors of symbolic values, by defining $T_{\bar{z}}(y_1, \dots, y_n)$ as $T_{\bar{z}}(y_1), \dots, T_{\bar{z}}(y_n)$. For example $T_{p_1, p_1, v_1, p_2}(p_2, v_1, p_1, p_1) = p_4, v_1, p_1, p_1$.

For a vector \bar{z} of symbolic values and vector \bar{v} of location variables, define the guard $g_{\bar{z}, \bar{v}}$ over formal parameters \bar{p} and location variables \bar{v} as the conjunction of the following conjuncts:

- for each formal parameter p_j which is the i th element z_i in \bar{z} :
 - $p_i \neq v_k$ for all location variables v_k in \bar{v} ,
 - $p_i = p_k$ whenever p_j is both the i th and the k th element of \bar{z} , and k is the largest index such that p_j is the k th element of \bar{z} ,
 - $p_i \neq p_k$ whenever the k th element z_k in \bar{z} is a formal parameter different from p_j and $i < k$,
- for each location variable v_j which is the i th element z_i in \bar{z} :
 - $p_i = v_j$.

From the set of consistent blocks $\{\mathcal{B}_1, \dots, \mathcal{B}_f\}$ resulting from the third step, starting from $\mathcal{M} = \langle \Sigma_I^\mathcal{E}, \Sigma_O^\mathcal{E}, Q, q_0, \delta, \lambda \rangle$, we can now construct a Symbolic Mealy machine $\mathcal{SM} = (I, O, L, l_0, \longrightarrow)$, where

- I is the set of input actions,
- O is the set of output actions,
- L is the set of blocks $\{\mathcal{B}_1, \dots, \mathcal{B}_f\}$; the arity of each location $\mathcal{B}_i \in L$ is the size of \bar{d} , where (q, \bar{d}) is the representative of \mathcal{B}_i ,
- $l_0 \in L$ is the block among $\mathcal{B}_1, \dots, \mathcal{B}_f$ that contains the pair (q_0, \square) ,
- \longrightarrow contains for each symbolic normal form $(q, \bar{d}) \xrightarrow{\alpha(\bar{z})/\bar{v}' := \bar{y}; \beta(\bar{z}')} (q', \bar{d}')$ of a transition between representative (q, \bar{d}) of block \mathcal{B} and representative (q', \bar{d}') of \mathcal{B}' , the transition

$$\mathcal{B} \xrightarrow{\alpha(\bar{p}) ; g_{\bar{z}, \bar{v}} / \bar{v}' := T_{\bar{z}}(\bar{v}) ; \beta(T_{\bar{z}}(\bar{z}'))} \mathcal{B}' ,$$

where \bar{v} is the location variables of \mathcal{B} , and \bar{p} is the vector p_1, \dots, p_n of formal parameters where n is the size of \bar{z} .

In general, the resulting Symbolic Mealy machine in general has “too small” transitions, since each guard completely characterizes equalities and inequalities between the formal input parameter \bar{p} and the location variables. To get a final Symbolic Mealy machine, we merge transitions that differ only in their guards, by taking the disjunction of their guards, whenever the new guard is still a conjunction.

In the final step in our work with our example, we create a Symbolic Mealy machine from the set of consistent blocks shown in Figure 3. The initial construction, after transforming transitions from symbolic normal form is the Symbolic Mealy machine shown in Figure 4. We thereafter merge the two self loops in Figure 4 into $\alpha(p_1, p_2) : p_1 = v_1/v_1 := v_1; \beta'(v_1)$. On the other edges, the parameterized output actions can be replaced by $\beta(p_2)$ without altering the output behavior, then we merge transitions, obtaining the Symbolic Mealy machine in Figure 1.

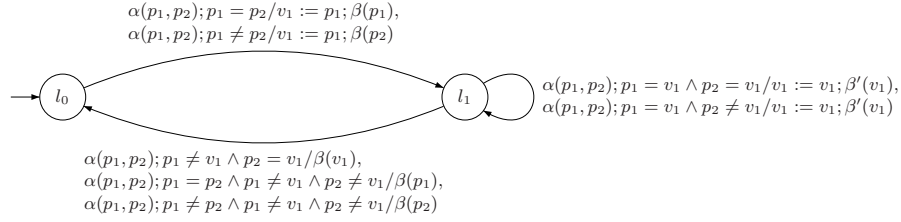


Fig. 4. Intermediate Symbolic Mealy machine constructed from the blocks in Figure 3.

6 Correctness and Complexity

The correctness of our transformation from a Mealy machine \mathcal{M} to a Symbolic Mealy machine \mathcal{SM} follows from the following two theorems.

Theorem 1. *Let \mathcal{E} be a finite domain such that $\mathcal{M} = \langle \Sigma_I^{\mathcal{E}}, \Sigma_O^{\mathcal{E}}, Q, q_0, \delta, \lambda \rangle$ is a fresh and finite-state (minimized) Mealy machine. If $\mathcal{SM} = (I, O, L, \longrightarrow, l_0)$ is the result of the transformation described in Section 5 from \mathcal{M} , then \mathcal{M} and $\mathcal{SM}_{\mathcal{E}}$ are equivalent (i.e., they produce the same output for all input strings $u \in (\Sigma_I^{\mathcal{E}})^*$).*

Proof. Let \mathcal{E} be as in the theorem. We will show by induction over the length of $u \in (\Sigma_I^{\mathcal{E}})^*$ that \mathcal{SM} outputs w on u iff \mathcal{M} outputs w on u , where $w \in (\Sigma_O^{\mathcal{E}})^*$.

That \mathcal{M} outputs w on u means that there is a sequence of transitions $q_0 \xrightarrow{\alpha_0(\bar{d}_0^I)/\beta_0(\bar{d}_0^O)} q_1 \xrightarrow{\alpha_1(\bar{d}_1^I)/\beta_1(\bar{d}_1^O)} \dots \xrightarrow{\alpha_n(\bar{d}_n^I)/\beta_n(\bar{d}_n^O)} q_{n+1}$ in \mathcal{M} , where the string $\alpha_0(\bar{d}_0^I), \alpha_1(\bar{d}_1^I), \dots, \alpha_n(\bar{d}_n^I)$ is u and the string $\beta_0(\bar{d}_0^O), \beta_1(\bar{d}_1^O), \dots, \beta_n(\bar{d}_n^O)$ is w . In the transformation described in Section 5, pairs of form $\langle q_i, \bar{d}_i \rangle$, where $q_i \in Q$ and \bar{d}_i is a vector of the values in $V(q_i)$, are partitioned into a set of consistent blocks $\{\mathcal{B}_0, \dots, \mathcal{B}_f\}$. By induction we prove that for all $i \geq 0$, whenever $q_i \xrightarrow{\alpha_i(\bar{d}_i^I)/\beta_i(\bar{d}_i^O)} q_{i+1}$ and $\langle q_i, \bar{d}_i \rangle \in \mathcal{B}_i$ for some \bar{d}_i , there is a corresponding transition $\mathcal{B}_i \xrightarrow{\alpha_i(\bar{p}); g/\bar{v}_{i+1} := T_{\bar{z}}(\bar{p}); \beta_i(T_{\bar{z}}(\bar{z}))} \mathcal{B}_{i+1}$ in \mathcal{SM} such that $\langle q_{i+1}, \bar{d}_{i+1} \rangle \in \mathcal{B}_{i+1}$ for some order \bar{d}_{i+1} of the data values in $V(q_{i+1})$, and

- the guard is satisfied $(\alpha_i(\bar{d}_i^I), \bar{v}_i \mapsto \bar{d}_i) \models g$,
- \bar{z} is the symbolic version $SV_{\bar{d}_i, \bar{d}_i^I}(\bar{d}_i^I)$ of the data values in the input,
- \bar{z}' is the symbolic version $SV_{\bar{d}_i, \bar{d}_i^O}(\bar{d}_i^O)$ of the data values in the output,
- $\mathcal{B}_i, \mathcal{B}_{i+1} \in \{\mathcal{B}_0, \dots, \mathcal{B}_f\}$,
- and the vectors \bar{v}_i, \bar{v}_{i+1} are the location variables of \mathcal{B}_i and \mathcal{B}_{i+1} , respectively.

Base case: by construction of Symbolic Mealy machines, the pair $\langle q_0, [] \rangle$ is in the initial location \mathcal{B}_0 of \mathcal{SM} .

Induction step: this follows from observing that after the partitioning of pairs of states and data-value vectors into consistent blocks, there is at least one pair $\langle r, \bar{z} \rangle$ in a block for each $r \in Q$, specially a pair $\langle q_{i+1}, \bar{d}_{i+1} \rangle$ for q_{i+1} . We can therefore construct the symbolic normal form $\langle q_i, \bar{d}_i \rangle \xrightarrow{\alpha_i(\bar{z})/\bar{v}_{i+1}:=\bar{y};\beta(\bar{z}')} \langle q_{i+1}, \bar{d}_{i+1} \rangle$ of each transition $q_i \xrightarrow{\alpha_i(\bar{d}_i^I)/\beta_i(\bar{d}_i^O)} q_{i+1}$ in \mathcal{M} . By our construction of Symbolic Mealy machines, there is in \mathcal{SM} a transition $\mathcal{B}_i \xrightarrow{\alpha_i(\bar{v});g_{\bar{z},\bar{v}_i}/\bar{v}_{i+1}:=T_{\bar{z}}(\bar{y});\beta_i(T_{\bar{z}}(\bar{z}'))} \mathcal{B}_{i+1}$ that corresponds to the symbolic transition $\langle q_i, \bar{d}_i \rangle \xrightarrow{\alpha_i(\bar{z})/\bar{v}_{i+1}:=\bar{y};\beta_i(\bar{z}')} \langle q_{i+1}, \bar{d}_{i+1} \rangle$, and $g_{\bar{z},\bar{v}_i}$ is a disjunct in g . \square

Theorem 2. *Let \mathcal{SM} and \mathcal{SM}' be Symbolic Mealy machines with the same set of input actions I and output actions O . Let m be the maximal arity of the locations in \mathcal{SM} and \mathcal{SM}' , and let n be the maximal arity of the input actions in I . If the size of \mathcal{E} is bigger than $m + \max(n, m)$, and if $\mathcal{SM}_{\mathcal{E}}$ and $\mathcal{SM}'_{\mathcal{E}}$ are equivalent, then for any data-value domain \mathcal{D} the Mealy machines $\mathcal{SM}_{\mathcal{D}}$ and $\mathcal{SM}'_{\mathcal{D}}$ are equivalent.*

Proof. If \mathcal{D} is at most as big as \mathcal{E} , we can injectively map the data values in \mathcal{D} to a subset $\mathcal{E}' \subseteq \mathcal{E}$. We can do this since the only test on the parameters are tests for equality between input parameters in Symbolic Mealy machines. Then it is easy to see that $\mathcal{SM}_{\mathcal{E}'}$ and $\mathcal{SM}'_{\mathcal{E}'}$ are equivalent.

Let us prove the case when \mathcal{D} is bigger than \mathcal{E} . We will define a mapping from \mathcal{D} to \mathcal{E} for each position in any input string $u_{\mathcal{D}} \in (\Sigma_I^{\mathcal{D}})^*$. Let $\mathcal{SM}_{\mathcal{E}}$ be the Mealy machine $\langle \Sigma_I^{\mathcal{E}}, \Sigma_O^{\mathcal{E}}, Q^{\mathcal{E}}, q_0^{\mathcal{E}}, \delta^{\mathcal{E}}, \lambda^{\mathcal{E}} \rangle$, and let $\mathcal{SM}_{\mathcal{D}}$ be the Mealy machine $\langle \Sigma_I^{\mathcal{D}}, \Sigma_O^{\mathcal{D}}, Q^{\mathcal{D}}, q_0^{\mathcal{D}}, \delta^{\mathcal{D}}, \lambda^{\mathcal{D}} \rangle$. Let $u_{\mathcal{D}} = \alpha_1(\bar{d}^1), \alpha_2(\bar{d}^2), \dots, \alpha_k(\bar{d}^k)$ be a string in $(\Sigma_I^{\mathcal{D}})^*$ of length k , and let $u_{\mathcal{D}}^i$ denote the prefix $\alpha_1(\bar{d}^1), \alpha_2(\bar{d}^2), \dots, \alpha_i(\bar{d}^i)$ of $u_{\mathcal{D}}$ of length i , where $i \leq k$. Let $V_{\mathcal{E}}$ be the state value function of $\mathcal{SM}_{\mathcal{E}}$, $V'_{\mathcal{E}}$ of $\mathcal{SM}'_{\mathcal{E}}$, and $V_{\mathcal{D}}$ of $\mathcal{SM}_{\mathcal{D}}$, respectively. For each position i in $u_{\mathcal{D}}$ let $\gamma_i : \mathcal{D} \mapsto \mathcal{E}$ be a mapping from \mathcal{D} to \mathcal{E} . We define γ_i inductively for $i = 0, 1, \dots, k$. Assume γ_i is defined, we extend γ_i to vectors of data values, by defining $\gamma_i(\bar{d}^i)$ as $\gamma_i(d_1^i), \dots, \gamma_i(d_j^i)$, where j is the length of \bar{d}^i . The domain of mapping γ_{i+1} is $V_{\mathcal{D}}(\delta^{\mathcal{D}}(q_0^{\mathcal{D}}, u_{\mathcal{D}}^i)) \cup \bar{d}^{i+1}$. The values in the domain will be injectively mapped to \mathcal{E} . The mapping γ_{i+1} is defined as follows.

- If $\gamma_i(d) \in V_{\mathcal{E}}(\delta^{\mathcal{E}}(q_0^{\mathcal{E}}, \alpha_1(\gamma_1(\bar{d}^1)), \dots, \alpha_i(\gamma_i(\bar{d}^i))))$ then $\gamma_{i+1}(d)$ is defined as $\gamma_i(d)$. This means, if a data value $d \in \mathcal{D}$ has been mapped to a data value

$e \in \mathcal{E}$, which is stored in a location variable, then d must in the next step be mapped to the same data value e .

- If $\gamma_i(d) \notin V_{\mathcal{E}}(\delta^{\mathcal{E}}(q_0^{\mathcal{E}}, \alpha_1(\gamma_1(\bar{d}^1)), \dots, \alpha_i(\gamma_i(\bar{d}^i))))$ then $\gamma_{i+1}(d)$ is mapped on some $e \in (\mathcal{E} \setminus V_{\mathcal{E}}(\delta^{\mathcal{E}}(q_0^{\mathcal{E}}, \alpha_1(\gamma_1(\bar{d}^1)), \dots, \alpha_i(\gamma_i(\bar{d}^i))))$. There are enough values in \mathcal{E} to find such a data value e since $\mathcal{SM}_{\mathcal{E}}$ is fresh.

Due to Lemma 1, $\mathcal{SM}'_{\mathcal{E}}$ and $\mathcal{SM}_{\mathcal{E}}$ are fresh. This in turn imply that their state value functions, $V_{\mathcal{E}}$ and $V'_{\mathcal{E}}$, can be calculated. Since $\mathcal{SM}'_{\mathcal{E}}$ is equivalent to $\mathcal{SM}_{\mathcal{E}}$, their state value functions $V_{\mathcal{E}}$ and $V'_{\mathcal{E}}$ are equivalent. We apply the inverse mapping γ_i^{-1} of γ_i for each position i in $\alpha_1(\gamma_1(\bar{d}^1)), \dots, \alpha_k(\gamma_k(\bar{d}^k))$ to get the corresponding input string for $\mathcal{SM}'_{\mathcal{D}}$. The corresponding string is (of course) $\alpha_1(\gamma_1^{-1}(\gamma_1(\bar{d}^1))), \dots, \alpha_k(\gamma_k^{-1}(\gamma_k(\bar{d}^k))) = \alpha_1(\bar{d}^1), \dots, \alpha_k(\bar{d}^k) = u_{\mathcal{D}}$.

Let the output $\lambda^{\mathcal{E}}(q_0^{\mathcal{E}}, \alpha_1(\gamma_1(\bar{d}^1)), \dots, \alpha_k(\gamma_k(\bar{d}^k)))$ in $\mathcal{SM}_{\mathcal{E}}$ be the output string $\beta_1(\bar{e}^1), \dots, \beta_k(\bar{e}^k)$. Then the output $\lambda^{\mathcal{D}}(q_0^{\mathcal{D}}, u_{\mathcal{D}})$ in $\mathcal{SM}_{\mathcal{D}}$ is the output string $\beta_1(\gamma_1^{-1}(\bar{e}^1)), \dots, \beta_k(\gamma_k^{-1}(\bar{e}^k))$. The output for $\mathcal{SM}'_{\mathcal{D}}$ is also the string $\beta_1(\gamma_1^{-1}(\bar{e}^1)), \dots, \beta_k(\gamma_k^{-1}(\bar{e}^k))$. I.e., running the string $u_{\mathcal{D}}$ in $\mathcal{SM}_{\mathcal{D}}$ and $\mathcal{SM}'_{\mathcal{D}}$ produces the same output string. \square

The two preceding theorems imply that if \mathcal{SM} is the Symbolic Mealy machine which we attempt to learn, and \mathcal{SM}' is the Symbolic Mealy machine we construct, then $\mathcal{SM}_{\mathcal{E}}$ is equivalent to $\mathcal{SM}'_{\mathcal{E}}$ for any domain \mathcal{E} .

An upper bound on the number of membership queries can be obtained from the corresponding bound on the number of membership queries needed to infer a Mealy machine $\mathcal{M} = \langle \Sigma_I^{\mathcal{E}}, \Sigma_O^{\mathcal{E}}, Q, q_0, \delta, \lambda \rangle$, which is $\mathcal{O}(|\Sigma_I^{\mathcal{E}}| \times |Q| \times \max(|\Sigma_I^{\mathcal{E}}|, |Q|) \times C)$, where

- $|\Sigma_I^{\mathcal{E}}| = \sum_{\alpha \in I} |\mathcal{E}|^{\text{actarity}(\alpha)}$, where $\text{actarity}(\alpha)$ is the arity of α ,
- $|Q| = \sum_{l \in L} |\mathcal{E}|^{\text{arity}(l)}$,
- C is the length of the longest counterexample returned in equivalence queries.

where $|\mathcal{E}|$ can be chosen as $m + \max(n, m) + 1$ (in the notation of Lemma 1) to make sure that $\mathcal{SM}_{\mathcal{E}}$ is fresh. The maximum number of equivalence queries required by the algorithm is $|Q| + |\mathcal{E}|$. To infer our example in Figure 2 we performed 333 membership queries and 1 equivalence query.

A way to reduce the number of membership queries required to infer the Mealy machine is to use a symmetry filter which deduces answers to membership queries from already answered membership queries [12]. The symmetry filter will filter out membership queries which have the same differences and equalities between parameter values as an already answered membership query.

7 Conclusions and Future Work

We have extended regular inference to a class of state machines with infinite state-spaces and infinite alphabets. Our motivation is to develop techniques for

inferring models of entities in communication protocols by observing test executions. It would be interesting to try to extend our approach to data domains with more complex operations, such as counters, time-stamps, etc.

Our two-phase approach implies that the intermediate finite-state Mealy machine may get rather large, in comparison with the final Symbolic Mealy machine. This problem might be mitigated by developing an algorithm where the generation of the intermediate machine and the compacting transformation are performed “in parallel”. Next on our agenda is to apply the results of this paper in a case study on realistic communication protocols.

Acknowledgement We thank B. Steffen and J. Parrow for helpful discussions.

References

1. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A., eds.: Model-Based Testing of Reactive Systems. Volume 3472 of Lecture Notes in Computer Science. Springer Verlag (2004)
2. Ball, T., Rajamani, S.: The SLAM project: Debugging system software via static analysis. In: Proc. 29th ACM Symp. on Principles of Programming Languages. (2002) 1–3
3. Henzinger, T., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. 29th ACM Symp. on Principles of Programming Languages. (2002) 58–70
4. Angluin, D.: Learning regular sets from queries and counterexamples. *Information and Computation* **75** (1987) 87–106
5. Balcázar, J., Daz, J., Gavaldá, R.: Algorithms for learning finite automata from queries: A unified view. In: *Advances in Algorithms, Languages, and Complexity*. Kluwer (1997) 53–72
6. Dupont, P.: Incremental regular inference. In Miclet, L., de la Higuera, C., eds.: *ICGI*. Volume 1147 of *Lecture Notes in Computer Science.*, Springer (1996) 222–237
7. Gold, E.M.: Language identification in the limit. *Information and Control* **10** (1967) 447–474
8. Kearns, M., Vazirani, U.: *An Introduction to Computational Learning Theory*. MIT Press (1994)
9. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences. *Information and Computation* **103** (1993) 299–347
10. Trakhtenbrot, B., Barzdin, J.: *Finite automata: behaviour and synthesis*. North-Holland (1973)
11. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In Kutsche, R.D., Weber, H., eds.: *Proc. FASE '02, 5th Int. Conf. on Fundamental Approaches to Software Engineering*. Volume 2306 of *Lecture Notes in Computer Science.*, Springer Verlag (2002) 80–95
12. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: *Proc. 15th Int. Conf. on Computer Aided Verification*. (2003)
13. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. In Wu, J., Chanson, S.T., Gao, Q., eds.: *Formal Methods for Protocol Engineering and Distributed Systems, FORTE/PSTV*, Beijing, China, Kluwer (1999) 225–240

14. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In Katoen, J.P., Stevens, P., eds.: Proc. TACAS '02, 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 2280 of Lecture Notes in Computer Science., Springer Verlag (2002) 357–370
15. Berg, T., Jonsson, B., Raffelt, H.: Regular inference for state machines with parameters. In Baresi, L., Heckel, R., eds.: FASE. Volume 3922 of Lecture Notes in Computer Science., Springer (2006) 107–121
16. Wolper, P.: Expressing interesting properties of programs in propositional temporal logic (extended abstract). In: Proc. 13th ACM Symp. on Principles of Programming Languages. (1986) 184–193
17. Jonsson, B., Parrow, J.: Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation* **107** (1993) 272–302
18. Niese, O.: An integrated approach to testing complex systems. Technical report, Dortmund University (2003) Doctoral thesis.
19. Cobleigh, J., Giannakopoulou, D., Pasareanu, C.: Learning assumptions for compositional verification. In: Proc. TACAS '03, 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 2619 of Lecture Notes in Computer Science., Springer Verlag (2003) 331–346
20. Ammons, G., Bodik, R., Larus, J.: Mining specifications. In: Proc. 29th ACM Symp. on Principles of Programming Languages. (2002) 4–16
21. Li, K., Groz, R., Shahbaz, M.: Integration testing of distributed components based on learning parameterized I/O models. In Najm, E., Pradat-Peyre, J.F., Donzeau-Gouge, V., eds.: FORTE. Volume 4229 of Lecture Notes in Computer Science. (2006) 436–450
22. Shahbaz, M., Li, K., Groz, R.: Learning and integration of parameterized components through testing. In Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W., eds.: TestCom/FATES. Volume 4581 of Lecture Notes in Computer Science., Springer (2007) 319–334
23. Kohavi, Z.: Switching and Finite Automata Theory: Computer Science Series. McGraw-Hill Higher Education (1990)