Gray-box Learning of Serial Compositions of Mealy Machines

Andreas Abel and Jan Reineke

Department of Computer Science
Saarland University
Saarbrücken, Germany
{abel, reineke}@cs.uni-saarland.de

Abstract. We study the following *gray-box learning* problem: Given the serial composition of two Mealy machines A and B, where A is known and B is unknown, the goal is to learn a model of B using only output and equivalence queries on the composed machine.
We introduce an algorithm that solves this problem, using at most $|B|$ equivalence queries, independently of the size of A. We discuss its efficient implementation and evaluate the algorithm on existing benchmark sets as well as randomly-generated machines.

1 Introduction

Tools to analyze software or hardware systems, such as static analyzers or model checkers, require accurate system models as input. Third-party components, however, are rarely specified at the level of detail required by such tools.

One approach to automatically obtain formal models of systems is active learning. Here, one commonly assumes an oracle, or teacher, that admits two kinds of queries about the system: output queries return the result of the system for a specific input; equivalence queries check whether a conjectured model is consistent with the system to be learned and return a counterexample if not. Based on this setup, Angluin introduced the L^* algorithm [2] for learning deterministic finite automata. L^* has since been extended to other modeling formalisms, such as Mealy machines [19], register automata [11], or symbolic automata [16]. It is also at the heart of several model checking approaches, including [4, 8, 20].

As the system is treated as a black box, no information about the internal structure of the system can be taken into account by most existing learning algorithms. In practice, however, systems are often composed of sub-components, for some of which models might be available, but it is not possible to access the known and the unknown parts separately from the outside. Partial information about the inner workings of a system may be inferred from manuals or conjectured from similar, yet better documented systems. This scenario is depicted in Figure 1.

While it is in theory possible to learn a model of the entire system using existing black-box approaches, this is often not viable in practice because the state space is too large. A problem, which has received little attention in the

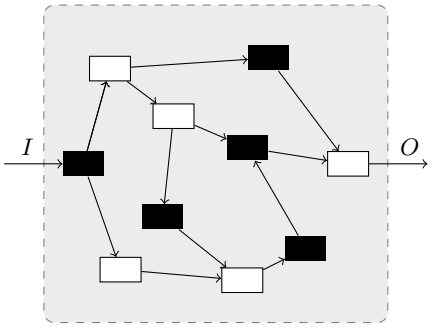

Fig. 1. Mealy machine network

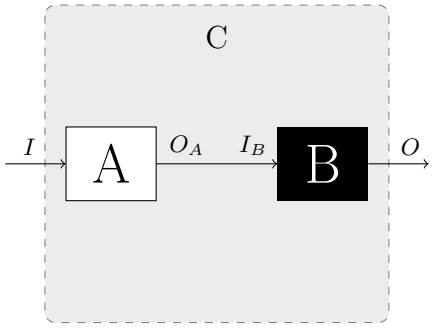

Fig. 2. Serial composition

literature so far, is how to use the available information about the system to focus the learning algorithm on those parts that are unknown. This problem could be termed *gray-box learning*.

In this paper, as a first step toward solving this problem, we study one specific instance: We assume that the system C is the serial composition of two Mealy machines A and B, and that we have a model for the left machine (A) and want to learn the right machine (B). We further assume that we can perform output and equivalence queries only on C as a whole. This scenario is shown in Figure 2.

While output queries can often be realized cheaply by measurements on the actual system, equivalence queries can usually only be approximated by a large number of such measurements. Our primary focus is thus to minimize the number of equivalence queries. We introduce an algorithm to exactly learn B in the context of A that performs at most $|B|$ equivalence queries, where $|B|$ denotes the number of states of B. We also discuss a more practical variant of this algorithm that requires a polynomial number of equivalence queries in the size of B.

We evaluate several variants of our approach on compositions of randomly-generated machines against an implementation of the classic L^* algorithm in LearnLib [13]. Furthermore, we also compare the performance of our approach with the tool BICA [17] on a set of standard benchmarks for minimizing incompletely specified Mealy machines. We show that our approach requires significantly fewer output and equivalence queries on most benchmarks.

2 Problem statement

In this section, we first formally define several concepts used throughout this paper. Then, we give a precise description of the problem that we address.

2.1 Basic notions

Definition 1 (Mealy Machine). *A Mealy machine M is a tuple (Q, I, O, δ, q_r), where $Q \neq \emptyset$ is a finite set of states, $I \neq \emptyset$ is a finite set of input symbols, $O \neq \emptyset$ is a finite set of output symbols, $\delta : Q \times I \to Q \times O$ is the transition function, and $q_r \in Q$ is the initial (reset) state.*

We extend δ to sequences in the usual way. We use ϵ to denote the empty sequence. Further, we use $M(x)$ to denote the output sequence of M when reading x, and $M_L(x)$ to denote last output of M when reading x.

Given two Mealy machines A and B, we can compose them to a serial Mealy machine C by using the output of A as the input for B. Formally:

Definition 2 ((Synchronous) Serial Composition of Mealy Machines). *Let $A = (Q_A, I_A, O_A, \delta_A, q_{r,A})$ and $B = (Q_B, I_B, O_B, \delta_B, q_{r,B})$ be two Mealy machines such that $O_A \subseteq I_B$. The serial composition of A and B is a Mealy machine $C = (Q, I, O, \delta, q_r)$, where $Q := Q_A \times Q_B$, $I := I_A$, $O := O_B$, $\delta((q_A, q_B), i) := ((q'_A, q'_B), o)$, where $(q'_A, o_A) := \delta_A(q_A, i)$, and $(q'_B, o) := \delta_B(q_B, o_A)$, and $q_r = (q_{r,A}, q_{r,B})$.*

Given a composition of two Mealy machines A and B, we define a machine B' to be right-equivalent to B in the context of A if the composition of A and B describes a machine that is equivalent to the composition of A and B'. Formally:

Definition 3 (Right-equivalence). *Let A, B, and B' be Mealy machines. Then, B' is* right-equivalent *to B in the context of A iff $\forall x \in I^* : B(A(x)) = B'(A(x))$.*

2.2 The gray-box learning problem

In this paper, we address the following problem. We assume that we have a serial composition C of two Mealy machines A and B. Further, we assume that we have a model of A, but B is unknown. While we do not have a model of C, we assume that we can determine the output of C on any input by an *output query*, and we can test whether a machine is equivalent to C by an *equivalence query*.

Using existing techniques, like Angluin's L^* algorithm [2], one could consider C to be a black box and learn a model of C. Such an approach would in the worst case employ a polynomial number of output and equivalence queries in the size of C, which can be up to $|A| \cdot |B|$.

Instead, our goal is to exploit the knowledge we have about A, and to learn a model of a minimum-size machine B', such that B' is right-equivalent to B in the context of A. In particular, as we consider equivalence queries to be more expensive than output queries, we want the number of equivalence queries to be polynomial in the number of states of B', independently of the size of A.

3 Preliminaries

Existing active learning approaches for Mealy machines (and related machine types) are usually based on a Myhill-Nerode-like equivalence relation that partitions the set of input words into classes such that the words that are in the same class cannot be distinguished with respect to different suffixes:

Definition 4 (Equivalence of input words). *Given a function $F : I^* \to O$, two words $x, y \in I^*$ are equivalent, $x \sim y$, iff $\forall z \in I^* : F(x \cdot z) = F(y \cdot z)$.*

F can be modeled by a Mealy machine iff this relation has finitely many equivalence classes. One can then construct a minimum-size Mealy machine whose states are the equivalence classes of this relation. Existing approaches compute the equivalence relation in a co-inductive fashion. In the beginning, they consider all words to be equivalent. Then, in each round this hypothesis is refined by identifying at least one new equivalence class, until the equivalence relation is fully determined.

If we consider the machine B in the serial composition with A, then it is possible that not all input sequences for B can be produced by A. Let $tr(A) = \{A(x) \mid x \in I^*\}$ be the set of output sequences that A can produce. For each output sequence $x \in tr(A)$ there might be multiple input sequences that produce this output. Let $A^{-1} : tr(A) \to I^*$ be a function such that $A^{-1}(x)$ returns one of these input sequences. In the following, it will not be important which of the possibly multiple sequences is actually returned.

We have that every right-equivalent Mealy machine B' for B in the context of A has to agree with the partial function $F_P : I_B^* \rightharpoonup O$ such that $\forall x \in tr(A) : F_P(x) = B_L(x)$. Note that while we do not have immediate access to B, we can use output queries on C to access B, as for all $x \in tr(A)$, $B_L(x) = C_L(A^{-1}(x))$.

Similarly to Definition 4, we define two words to be *right-compatible* in the context of A iff they cannot be distinguished with respect to different suffixes.

Definition 5 (Right-compatibility). *Two words $x, y \in I_B^*$ are* right-compatible *in the context of A, $x \sim_A y$, iff $\forall z \in I_B^* : (xz \notin tr(A) \vee yz \notin tr(A) \vee B_L(xz) = B_L(yz))$. Otherwise, x and y are incompatible, $x \nsim_A y$.*

However, right-compatibility is, unlike equivalence, not transitive. Thus it is not an equivalence relation, which means we cannot directly use the construction sketched above to build a minimum-size machine.

To see this, consider a Mealy machine A and two output symbols $a, b \in O_A$ with $\forall z \in I_B^* : az, bz \in tr(A) \wedge B_L(az) = 0 \wedge B_L(bz) = 1$ and $\forall z \in I_B^* : cz \notin tr(A)$. So B always outputs 0 if the first output of A was a, it always outputs 1 if the first output of A was b, and A never outputs c as the first output.

This means that $a \sim_A c$ and $b \sim_A c$, but $a \nsim_A b$. For this example, we can build a machine with three states that is right-equivalent to B. From the start state, a transition with c can go to any state. This also shows that there can be multiple machines with the minimum number of states that are right-equivalent to B.

4 Approach

Equivalence queries are typically assumed to be more expensive than output queries. Many existing active learning techniques therefore focus on keeping the number of required equivalence queries low.

At a high level, Angluin's L^* algorithm for instance, can be described as follows. In each round, the algorithm first performs a sequence of output queries in a systematic way, until there is exactly one machine of minimum size that is consistent with the results from all output queries performed so far. Only then, the

algorithm performs an equivalence query. If this query returns a counterexample, this implies that the correct machine must have at least one additional state. Thus, Angluin's algorithm performs at most n equivalence queries, where n is the size of the minimal correct machine.

Unlike in Angluin's setting, in general no unique machine of minimum size that is consistent with a set of observations exists. The basic idea behind our approach is to perform output queries until *all* machines of minimum size that are consistent with these queries are right-equivalent in the context of A. We then perform an equivalence query for one of these machines. If this query results in a counterexample, this counterexample witnesses that all of these machines are incorrect, and thus, the correct machine must have at least one additional state.

One challenge is to find a suitable sequence of output queries that is guaranteed to reduce the number of machines that are consistent with all queries performed so far. The basic idea is to iteratively construct all machines of minimum size that agree with all of the previous queries. We can then check whether each pair of these machines is right-equivalent. If they are not, we use a distinguishing sequence as a counterexample, without performing an equivalence query.

However, applying this approach naively would not be viable in many cases because there can be an exponential number of machines of the same size that are consistent with a set of observations, in particular in the beginning, when only a small number of queries have been performed. Thus, we identify a number of necessary conditions for candidate machines to be right-equivalent which can be efficiently determined on observation tables. Some of these conditions correspond to notions from Angluin's algorithm, such as consistency and closedness, while others, like input-completeness, are special to our particular setting.

In the rest of this section, we describe our proposed algorithm in detail and introduce the necessary theoretical concepts. In particular, we describe in detail which output queries our algorithm performs to systematically reduce the number of machines that are consistent with the observations made so far. In the following, we assume that the reader is familiar with Angluin's L^* algorithm [2].

4.1 Observation tables

The main data structure used in our approach is an *observation table*. The rows of the table are indexed by a set of prefixes, the columns by a set of suffixes, and the entries of the table store the last output symbol of an output query for the concatenation of the corresponding prefix and suffix. If this concatenation is not a possible output sequence of the left machine A, we do not perform an output query, but store $\perp$ in this cell instead. In contrast to most previous definitions, our observation tables *do not* consist of two explicitly distinguished parts.

Definition 6 (Observation Table). *An observation table $T = (S, E, Q)$ consists of a finite non-empty prefix-closed set of prefixes $S \subseteq tr(A)$, a finite suffix-closed set of suffixes $E \subseteq I_B^*$ (such that $I_B \subseteq E$, and $\epsilon \notin E$), and a function $Q : (S, E) \to O_B$ such that $Q(x, e) = C_L(A^{-1}(xe))$ iff $xe \in tr(A)$ and $Q(x, e) = \perp$ otherwise.*

For a set $R \subseteq S$ and $a \in I_B$, let $Succ_T(R, a) := \{xa \mid x \in R \wedge xa \in S\}$, i.e., $Succ_T(R, a)$ is the set of successor rows for elements of R that are in the table.

In the following, we will use the term *row* both for the prefixes and for the entries of a row, when it is clear what is meant from the context.

We call two rows compatible if all columns that are not $\perp$ in both rows are the same.

Definition 7 (Compatibility). *The rows for two prefixes $x, y \in S$ are compatible iff $\forall e \in E : Q(x, e) = \perp \vee Q(y, e) = \perp \vee Q(x, e) = Q(y, e)$.*

We call an observation table consistent if whenever two rows are compatible, their successors are also compatible.

Definition 8 (Consistency). *An observation table T is consistent iff for all prefixes $x, y \in S$ such that the rows for x and y are compatible, for all $a \in I_B$ all rows in $Succ_T(\{x, y\}, a)$ are compatible.*

If there is a suffix $e \in E$ that shows that the successors of x and y under an input a are not compatible, then ae is a suffix that shows that the rows for x and y are also not compatible. Thus, we can add ae to E to resolve this inconsistency.

We define a partition of the set of rows as follows.

Definition 9 (Partition). *A partition for observation table $T = (S, E, Q)$ is a partition $P = \{P_1, ..., P_k\}$ of S, such that*

- *for all $x, y \in P_i$: the rows for x and y are compatible,*
- *for each P_i, and for all $a \in I_B$, there is a P_j, such that: $Succ_T(P_i, a) \subseteq P_j$.*

Note that if $Succ_T(P_i, a) \neq \emptyset$ then there is only one such P_j since all classes of the partition are disjoint.

We will later show how we can use partitions to build candidate machines that are consistent with the observations made so far. The words in the same class of a partition will then lead to the same states in these candidate machines.

We call a partition closed if for each class of the partition and each input symbol a, the observation table contains a successor row (under a) for at least one word of this class, if we know from the observations made so far that such a successor must exist. Our inference algorithm uses closedness as a way to determine which additional rows should be added to the table.

Definition 10 (Closedness for Partitions). *Let $P = \{P_1, ..., P_k\}$ be a partition for $T = (S, E, Q)$. P is closed if for all $P_i \in P$: if there is some $x \in P_i$ and some sequence $az \in E$ with $a \in I_B$ and $z \in I_B^*$ such that $Q(x, az) \neq \perp$, then there must be some $y \in P_i$ for which $Q(y, az) \neq \perp$, and $ya \in S$.*

Given an observation table T, let $\Pi(T, n)$ be the set of all partitions of size n. Let $\Pi_{min}(T)$ be the set of partitions of minimum size for an observation table T, i.e., $\Pi_{min}(T) = \Pi(T, m)$ where $m = min\{n \mid \Pi(T, n) \neq \emptyset\}$.

Definition 11 (Closedness). *An observation table $T = (S, E, Q)$ is closed if all minimum-size partitions $P \in \Pi_{min}(T)$ are closed.*

Definition 12 (Partial Closedness). *An observation table T is partially closed (p-closed) iff for all prefixes $x \in S$ and all sequences $az \in E$ such that $Q(x, az) \neq \perp$, there is a prefix $y \in S$ such that the rows for x and y are compatible, $Q(y, az) \neq \perp$ and $ya \in S$.*

If a table is not p-closed, then no partition can be closed.

Definition 13 (Agreement). *A Mealy machine M agrees with an observation table $T = (S, E, Q)$ if for all $x \in S$ and $e \in E$, $Q(x, e) = \perp \vee Q(x, e) = M_L(xe)$.*

For any closed partition $P = \{P_1, ..., P_k\}$ in $\Pi_{min}(T)$, we can build the following Mealy machine $M_P = (Q, I, O, \delta, q_r)$ with $k+1$ states: $Q := P \cup \{error\}$, $I := I_B$, $O := O_B \cup \perp$, $\delta(P_i, a) := (error, \perp)$ if $Succ_T(P_i, a) = \emptyset$, otherwise: $\delta(P_i, a) := (P_j, b)$ such that for some $x \in P_i$: $Q(x, a) = b \neq \perp$ and $Succ_T(P_i, a) \subseteq P_j$, and $q_r := P_i$ such that $\epsilon \in P_i$.

This machine enters a special error state if there is a class of the partition, for which the successor class is not defined.

In the following, we will use the notation $\pi_i(t)$ to denote the i-th component of a tuple t, e.g., $\pi_2(q_r, a) = a$.

Lemma 1. *Let P be a closed partition of an observation table $T = (S, E, Q)$, and $M_P = (Q, I, O, \delta, q_r)$ the Mealy machine constructed as described above. Then for all words $x \in S$, $x \in \pi_1(\delta^*(q_r, x))$.*

Theorem 1. *For a closed partition P of an observation table T, the machine M_P agrees with T.*

Definition 14. *Let $\gamma(M_P)$ be the set of machines with k states that can be obtained from M_P by removing the error state and replacing the transitions to the error state by transitions with arbitrary outputs and successor states.*

Theorem 2. *Let T be a closed observation table. Then every minimum-size machine M that agrees with T is isomorphic to an element of $\gamma(M_P)$ for some $P \in \Pi_{min}(T)$.*

Theorem 3. *If for a closed partition P the error state is not reachable in a composition of A with M_P, then all machines in $\gamma(M_P)$ are right-equivalent.* [1]

If the error state is reachable, we can use an input sequence that leads to the error state to extend the observation table.

Definition 15 (Input-Completeness). *An observation table $T = (S, E, Q)$ is input-complete if for all minimum-size partitions $P \in \Pi_{min}(T)$, the error state is not reachable in a composition of A with M_P.*

Definition 16 (Uniqueness). *An observation table $T = (S, E, Q)$ is unique if for all pairs of minimum-size partitions $P, P' \in \Pi_{min}(T)$, the machines M_P and $M_{P'}$ are right-equivalent in the context of A.*

[1] The proofs for the theorems in this section are available at `http://embedded.cs.uni-saarland.de/GrayBoxLearning/details.pdf`.

It follows that all machines of minimum-size size that agree with a consistent, closed, input-complete, and unique observation table are right-equivalent, and they can be obtained from the partitions.

Algorithm 1: Main algorithm

```
Input: Machine A, OutputQuery OQ, EquivalenceQuery EQ
begin
    ObservationTable OT ← empty table
    addRow([ϵ])
    curSize ← 1
    while (true) do
        while (¬consistent ∨ ¬p-closed) do
            makeConsistent()                                  // consistency
            makePClosed()                                     // p-closedness
        set partitions ← ∅
        prevMachine ← ⊥
        while (true) do
            partition ← findNextPartition(partitions, curSize)
            if (partition = ⊥) then
                if (prevMachine = ⊥) then
                    curSize ← curSize+1
                    continue
                else
                    counterexample ← EQ(prevMachine)
                    if (counterexample = ⊥) then
                        removeErrorState(prevMachine)
                        return prevMachine
                    else
                        handleCounterexample(counterexample)
                        break
            if (¬isClosed(partition)) then
                closePartition()                              // closedness
                break
            machine ← getMachineForPartition(partition)
            errorPath ← getPathToErrorStateInComposition(A,machine)
            if (errorPath ≠ ⊥) then
                handleCounterexample(errorPath)       // input-completeness
                break
            if (prevMachine ≠ ⊥) then
                distInput ← checkRightEquivalence(A, machine, prevMachine)
                if (distInput ≠ ⊥) then
                    handleCounterexample(distInput)           // uniqueness
                    break
            partitions ← partitions ∪ {partition}
            prevMachine ← machine
```

4.2 Inference algorithm

At a high level, our algorithm works as shown in Algorithm 1. In each iteration of the main loop, we first make sure that the observation table is consistent and p-closed (by adding additional rows and columns if necessary). Then, we successively determine the partitions of minimum size for the observation table. Whenever we find a partition that is not closed, we add new rows to the table such that the partition becomes closed, and we continue with the next iteration of the main loop. If we find a closed partition, we check whether the error state is reachable in a composition of the corresponding machine with A. If we find a sequence that leads to the error state, this means that the table is not input-complete. Thus, we add this sequence (and its prefixes) to the observation table

and continue with the next iteration of the main loop. If we find more than one closed and input-complete partition in the same iteration of the main loop, we check whether the machines for these two partitions are right-equivalent in the context of A. If we find a distinguishing sequence, we extend the observation table accordingly, and continue with the next iteration of the main loop. If finally the table is consistent, closed, input-complete, and unique, we perform an equivalence query for the last machine we found (which is right-equivalent to all machines of minimum size that agree with the table). If the equivalence query is successful, we are done, otherwise, we get a counterexample that we add to the table.

5 Implementation

In this section, we describe how our algorithm can be implemented. We also propose some improvements that make the algorithm more usable in practice.

5.1 Computing the partitions

We reduce the problem of finding the partitions for a given size n, which is an NP-complete problem, to a boolean satisfiability (SAT) problem. Related reductions were used by [1] for minimizing incompletely-specified Mealy machines, and by [10] for finding DFAs that agree with a set of positive and negative input samples.

For space reasons, we will omit the details of our reduction approach. They are available at `embedded.cs.uni-saarland.de/GrayBoxLearning/details.pdf`.

5.2 Reachability of the error state

If the error state is reachable with an input a from a state in the composition of the hypothesis machine with the left machine A, this means for no prefix p in the observation table that leads to this state, the input pa is a possible output of the left machine, however, there is another possible output sequence that leads to the same state that has a corresponding successor. We can thus use this sequence as a counterexample.

A straightforward way to check the reachability would be to build the composition, and then to perform a breadth-first search on the composition. A necessary condition for the reachability of the error state in the composition is that the error state is reachable in the hypothesis machine. We have observed that in practice, if the error state is reachable in the hypothesis machine, then in many cases, it is also reachable in the composition. Thus, we use the following approach to find a corresponding sequence quickly: We first determine for each state of the hypothesis machine the distance of the shortest path to the error state. We then use this distance to guide the search in a modified breadth-first search in the composed machine.

5.3 Checking if two machines are right-equivalent

A straightforward way to check whether two hypothesis machines B and B' are equivalent in the context of A would be to compose both with A, and then check the two compositions for equivalence, for example using Hopcroft-Karp's near-linear algorithm. However, this can be computationally expensive when A is large compared to B and B', as it requires building the composition twice.

Therefore, we take the following alternative approach. We build a new machine D that outputs 1 iff the outputs of B and B' differ on (a prefix of) the corresponding input, and 0 otherwise. While the size of D can be quadratic in the size of B, we have observed that, after minimization, in practice the sizes are smaller or comparable to B. To check whether B and B' are right-equivalent we can then just check whether the composition of A with the minimized version of D can output 1, using the search algorithm described in the previous section.

5.4 Performing additional equivalence queries

While evaluating our approach on the benchmarks from Section 6.2, we came across several benchmarks, for which the number of right-equivalent machines is very large. We propose the following modification of our algorithm that uses some additional equivalence queries to achieve better performance in practice. For each value of *curSize*, we perform the n-th equivalence query after computing 2^n partitions, rather than first enumerating all right-equivalent machines.

The number of Mealy machines of size n with input alphabet I and output alphabet O that compute different functions is bounded by $(n \cdot |O|)^{n \cdot |I|}$. So the number of equivalence queries performed for machines of size i is at most $\log_2(i \cdot |O|)^{i \cdot |I|} = i \cdot |I| \cdot \log_2(i \cdot |O|)$. For $n = |B|$, the modified algorithm performs at most $\sum_{i=1}^{n} i \cdot |I| \cdot \log_2(i \cdot |O|) \in \mathcal{O}(n^3 \cdot |I| \cdot \log_2 |O|)$ many equivalence queries.

5.5 Handling counterexamples

Like in the original version of Angluin's L^* algorithm, we handle counterexamples by adding all prefixes of the counterexamples as rows to the table. Since, in general, the length of a counterexample can depend on $|C|$, the number of rows that are added (and hence the number of output queries that need to be performed to determine their entries) is not independent of $|A|$.

Rivest and Schapire [18] described an improved approach to handle counterexamples that needs to perform only a logarithmic number of membership queries (in the length of the counterexample). However, it is not possible to directly adapt this method to our setting, since it requires that there is always a suffix of the counterexample that is a distinguishing suffix for two compatible rows. It is future work to develop more advanced methods to deal with counterexamples in our setting.

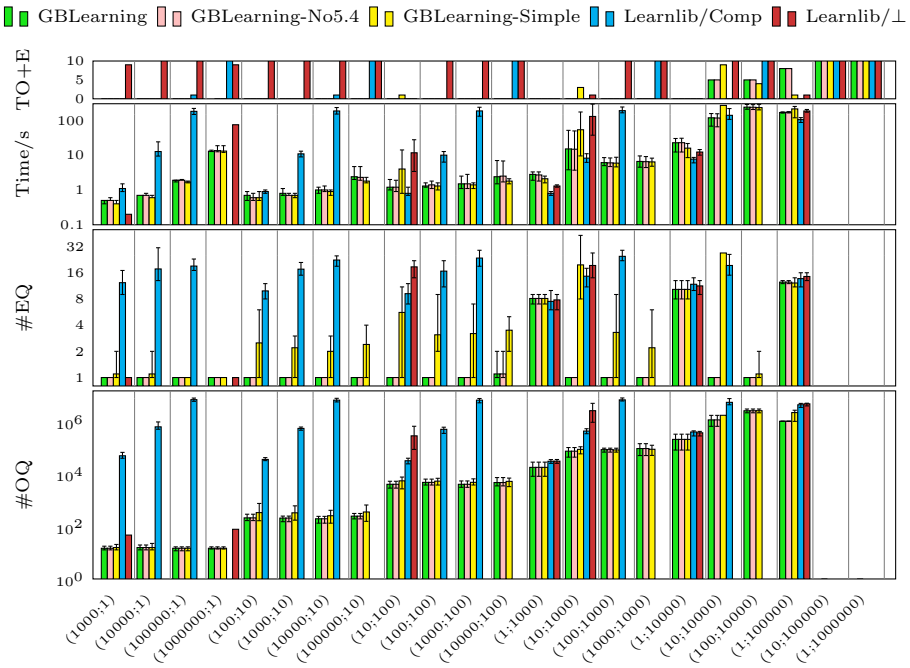

Fig. 3. Evaluation on randomly-generated machines

6 Evaluation

6.1 Randomly-generated machines

In this section, we compare several variants of our approach (that differ, in particular, with respect to the number of equivalence queries they perform) with the Mealy machine version of Angluin's L^* algorithm. We use a set of randomly generated compositional Mealy machines with between $1,000$ and $1,000,000$ states, and an input and output alphabet of size 4.

The results are shown in Figure 3. *GBLearning* ("Gray Box Learning") is an implementation of the approach described in the previous sections. *GBLearning-No5.4* is a variant of our approach that does not perform the additional equivalence queries described in Section 5.4. *GBLearning-Simple* is another variant of our approach that does neither check whether the error state is reachable, nor whether different machines that are consistent with the observation table are right-equivalent. Instead, it immediately performs an equivalence query upon finding a closed partition. Thus, the number of equivalence queries of this variant is *not* independent of the size of the right machine.

We compare these implementations with two variants of Angluin's L^* algorithm, as implemented in LearnLib [13] (*ExtensibleLStarMealy*). *Learnlib/Comp* treats the system as a black box, and learns the composition. Furthermore, we

modified Learnlib (*Learnlib/*$\perp$) such that it uses L^* on the right machine; impossible inputs are assumed to result in a special output symbol ($\perp$). Equivalence queries are performed by first composing the hypothesis for the right machine with the left machine. Note that this variant does not learn a minimum-size machine; in fact, the learned machine might even be larger than the composition.

The columns of Figure 3 show the sizes of the randomly-generated machines in the form $(|Q_A|;|Q_B|)$. The rows show the number of output queries (#OQ), equivalence queries (#EQ), and the execution time in seconds (averages, minima and maxima for the successful runs of 10 different randomly-generated machines of the same size). The row *TO+E* shows on how many of the 10 runs a timeout (5 minutes), or an error occurred. For *Learnlib/Comp* we observed one error, for *Learnlib/*$\perp$ three errors due to an exception ("incompatible output symbols"). All other entries in this row were timeouts. We used the jar-Release of LearnLib in version 0.9.1-ase2013-tutorial-r1. Both our tool and Learnlib use a query cache to avoid performing the same output query multiple times.

We observe that *Learnlib/*$\perp$ was only successful when A had 10 or fewer states, or when B had just one state. It performed slightly better than *Learnlib/Comp* in only a few cases where $|Q_A| = 1$ or $|Q_B| = 1$. *Learnlib/Comp* was successful on almost all benchmarks with up to 100,000 states; however, it could not solve any benchmark with more states.

The implementations of our tool could also handle composed machines of larger sizes, in particular when B is relatively small. *GBLearning* and *GBLearning-No5.4* were successful on all benchmarks where B had up to 1,000 states, on several where B had 10,000 states, and on two where B had 100,000 states.

For those machines that our implementations and *Learnlib/Comp* could handle, the number of required output queries was much smaller for our implementations if $|Q_A| > 1$. In this case, there was no significant difference in the number of output queries between the different variants of our approach. Also, for $|Q_A| > 1$, the number of output queries depends mainly on $|Q_B|$ for all three variants.

For *GBLearning* and *GBLearning-No5.4*, the number of equivalence queries was mostly 1 or 2 even for relatively large unknown machines; however, randomly-generated machines might not be representative in this regard. *GBLearning-Simple* needed significantly more equivalence queries than these two variants for $|Q_B| > 1$, but significantly fewer than *Learnlib/Comp* for $|Q_A| > 10$.

6.2 Benchmarks for the minimization of incomplete Mealy machines

We now compare the same variants of our implementation with BICA [17]. BICA is a tool that uses a modification of Angluin's L^* algorithm for minimizing incompletely specified Mealy machines, a known NP-complete problem. If we choose as the left machine A of the composition a machine such that $tr(A)$ corresponds exactly to the specified inputs of the right machine, the minimization of incompletely specified Mealy machines can be considered to be a special case of our approach.

We use the same set of benchmarks that was used by the authors of BICA to evaluate their approach, but we excluded those benchmarks for which some output

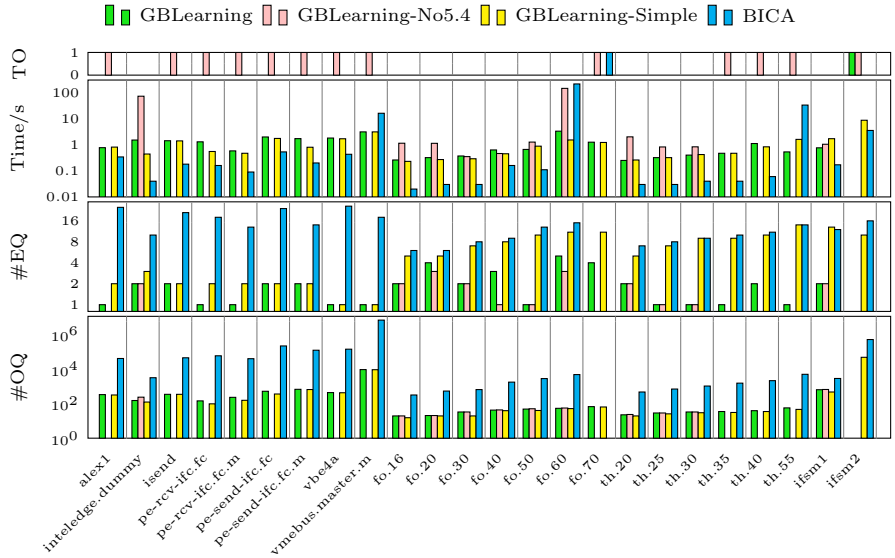

Fig. 4. Minimization benchmarks

bits are not specified, as this is currently not supported by our implementation. We use BICA in version 5.0.3. We added code to count the number of output and equivalence queries.

The results are shown in Figure 4. All variants of our approach require significantly fewer output queries than BICA; this might in part be due to the use of a query-cache in our implementations. Both *GBLearning* and *GBLearning-No5.4* require significantly fewer equivalence queries than BICA; however, *GBLearning-No5.4* did not terminate on several of them within a timeout of 5 minutes. *GBLearning-Simple* could solve all benchmarks; the number of equivalence queries was comparable to *GBLearning* for about one third of the benchmarks, on the remaining benchmarks it was comparable to BICA. However, the objective of the algorithm used by BICA was not mainly to minimize the number of queries, since output and in particular equivalence queries are very cheap in the present scenario, as the machines to be minimized are readily available, and equivalence queries can be performed by automata constructions. Instead, the focus of their approach was to minimize the execution time. In such a case, minimizing the number of equivalence queries as we do, by checking right-compatibility of many different candidates, is not beneficial in terms of runtime. In fact, BICA was faster on many of the benchmarks.

7 Related work

The concept of actively learning DFAs using membership and equivalence queries was introduced by Angluin in [2]. Angluin developed a polynomial-time learning

algorithm, called L^*, for fully-specified DFAs. Rivest and Schapire [18] later improved this algorithm and proposed a modification that does not require the system to have a reset state.

Multiple studies [15, 6, 7, 17, 12] considered scenarios in which the teacher is unable to answer some output queries. In contrast to our setting, where the input language is a known regular language, these approaches assume that no information about the unspecified inputs is available a priori, so that whether a particular input is specified can only be determined by performing an output query. In this scenario, the best bound Leucker and Neider [15] could give for the number of required equivalence queries is $n^{\mathcal{O}(n)}$. Hsu and Lee [12] claimed that their approach is able learn a minimum-size model for an incompletely specified FSM in polynomial time. However, this approach is incorrect; it does in general not find a minimum-size machine [14].

The term "gray-box" has been used in relation with Angluin's algorithm before, but in different contexts. Babic et al. [3] describe an approach to learn an input-output relation for a program. They propose a symbolic version of L^* that is allowed to inspect the internal symbolic state of the program. Henkler et al. [9] consider real-time statecharts that have an additional interface for retrieving the current internal state. Elkind et al. introduce grey-box checking [5]. A grey-box system consists of completely-specified (white boxes) and unknown components (black boxes). The goal of grey-box checking is then to check whether the system satisfies a property, given e.g. by an LTL formula. The main problem studied by Elkind et al. is to learn a model of the entire system given the knowledge about the white boxes, which can then be used to model check the property. In contrast to our setting, they consider finite automata that synchronize on common letters in their alphabet, whereas we consider Mealy machines with explicit inputs and outputs. Furthermore, they only use output queries; equivalence queries are realized via a large number of output queries.

8 Discussion and future work

We have introduced an algorithm for gray-box learning of serial compositions of Mealy machines. Experimental results confirm that taking into account prior knowledge about a system to be learned often yields significant performance gains.

There are plenty of open problems left for future work: In this paper, we have considered the serial composition of two Mealy machines. In future work, we would like to extend our approach to arbitrary composition topologies.

While we can precisely bound the number of equivalence queries, we lack such knowledge about the number of output queries. More generally, we would like to better understand the computational complexity of the problem at hand.

In our experimental evaluation, we realized equivalence queries by automata-theoretic constructions, as we had precise knowledge of the system to be learned. In real application scenarios, such knowledge is not available. In those cases, it would be interesting to systematically perform measurements in a way that focuses on the unknown parts.

References

1. Abel, A., Reineke, J.: MeMin: SAT-based exact minimization of incompletely specified mealy machines. ICCAD '15, IEEE Press (2015)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and computation 75(2), 87–106 (1987)
3. Babic, D., Botincan, M., Song, D.: Symbolic grey-box learning of input-output relations. Tech. Rep. UCB/EECS-2012-59, University of California, Berkeley (2012)
4. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2003)
5. Elkind, E., Genest, B., Peled, D., Qu, H.: Grey-box checking. In: Formal Techniques for Networked and Distributed Systems (FORTE). pp. 420–435 (2006)
6. Grinchtein, O., Leucker, M.: Learning finite-state machines from inexperienced teachers. In: Grammatical Inference: Algorithms and Applications. Springer (2006)
7. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning, Lecture Notes in Computer Science, vol. 4130, pp. 483–497. Springer Berlin Heidelberg (2006)
8. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, vol. 2280, pp. 357–370. Springer Berlin Heidelberg (2002)
9. Henkler, S., et al.: Legacy component integration by the Fujaba real-time tool suite. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. pp. 267–270. ICSE '10, ACM, New York, NY, USA (2010)
10. Heule, M., Verwer, S.: Exact DFA identification using SAT solvers. In: Grammatical Inference: Theoretical Results and Applications, vol. 6339, pp. 66–79 (2010)
11. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring canonical register automata. In: Verification, Model Checking, and Abstract Interpretation, Lecture Notes in Computer Science, vol. 7148, pp. 251–266. Springer Berlin Heidelberg (2012)
12. Hsu, Y., Lee, D.: Machine learning for implanted malicious code detection with incompletely specified system implementations. In: IEEE International Conference on Network Protocols. pp. 31–36. Washington, DC, USA (2011)
13. Isberner, M., Howar, F., Steffen, B.: The open-source learnlib. In: Computer Aided Verification. pp. 487–495. Springer (2015)
14. Lee, D.: Personal communication (January 2015)
15. Leucker, M., Neider, D.: Learning minimal deterministic automata from inexperienced teachers. In: ISoLA. pp. 524–538 (2012)
16. Maler, O., Mens, I.E.: Learning regular languages over large alphabets. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2014)
17. Pena, J., Oliveira, A.: A new algorithm for exact reduction of incompletely specified finite state machines. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18(11), 1619–1632 (Nov 1999)
18. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Information and Computation 103(2), 299–347 (1993)
19. Shahbaz, M., Groz, R.: Inferring mealy machines. In: FM 2009: Formal Methods, pp. 207–222. Springer (2009)
20. Vardhan, A., Sen, K., Viswanathan, M., Agha, G.: Using language inference to verify omega-regular properties. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2005)