Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 373265, 11 pages
http://dx.doi.org/10.1155/2013/373265

Research Article

Active Learning of Nondeterministic Finite State Machines

Warawoot Pacharoen,[1] Toshiaki Aoki,[2] Pattarasinee Bhattarakosol,[3] and Athasit Surarerks[1]

[1] *Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand*
[2] *School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan*
[3] *Department of Mathematics, Chulalongkorn University, Bangkok 10330, Thailand*

Correspondence should be addressed to Athasit Surarerks; athasit.s@chula.ac.th

Received 27 June 2013; Revised 12 October 2013; Accepted 12 October 2013

Academic Editor: Yang Tang

We consider the problem of learning nondeterministic finite state machines (NFSMs) from systems where their internal structures are implicit and nondeterministic. Recently, an algorithm for inferring observable NFSMs (ONFSMs), which are the potentially learnable subclass of NFSMs, has been proposed based on the hypothesis that the complete testing assumption is satisfied. According to this assumption, with an input sequence (query), the complete set of all possible output sequences is given by the so-called Teacher, so the number of times for asking the same query is not taken into account in the algorithm. In this paper, we propose L^*_{NM} , a refined ONFSM learning algorithm that considers the amount for repeating the same query as one parameter. Unlike the previous work, our approach does not require all possible output sequences in one answer. Instead, it tries to observe the possible output sequences by asking the same query many times to the Teacher. We have proved that L^*_{NM} can infer the corresponding ONFSMs of the unknown systems when the number of tries for the same query is adequate to guarantee the complete testing assumption. Moreover, the proof shows that our algorithm will eventually terminate no matter whether the assumption is fulfilled or not. We also present the theoretical time complexity analysis of L^*_{NM}. In addition, experimental results demonstrate the practical efficiency of our approach.

1. Introduction

Over the past decade the use of *automata learning techniques* has become widespread in the domain of formal verification (e.g., [1–4]). The learning techniques are usually employed for inferring a formal model such as *finite state automaton* (FA) or *finite state machine* (FSM) (also called *transducer*) of a system whose internal behavior is unknown. There are many methods that have been reported in the literature for automata learning (see, e.g., evolutionary based algorithms [5–7], SAT-solver based algorithm [8], and ant-colony optimization-based algorithm [9]). Among various techniques, Angluin's algorithm L^* [10] has received much attention in many studies.

Research into L^*, is essentially an active learning procedure (i.e., learning by queries) for inferring a minimal *deterministic finite state automaton* (DFA). As opposed to a passive learning approach (e.g., RPNI algorithm [11]), an active learning algorithm can choose and ask the expected queries to the so-called *Teacher* who is assumed to correctly answer them. There are two types of queries in L^*: *membership* and *equivalence* queries. A membership query is asked to investigate whether a given string is actually in the language of the target DFA. An equivalence query is asked to verify whether a hypothesized DFA is correct.

While the bulk of research in L^* has focused on *deterministic* models, nondeterminism is not unusual in certain systems that are composed of a number of components such as a communication system, a component-based system, and a service-oriented system. Such nondeterminism could arise from the asynchronous communication between different components, as well as from unpredictable activities such as interleaving between components. The use of a *nondeterministic finite state machine* (NFSM) is preferred because it can specify both an input/output structure and nondeterminism in a more neutral manner. Although it has been shown that learning the class of NFSMs and *non-deterministic finite automata* (NFAs) may be impossible [12], the case is not true for the whole class. For example, *residual finite state automata* (RFSAs) [13–15], *unambiguous finite automata* (UFAs) [16], and *parameterized finite state machines* (PFSMs) [17] are the subclasses that can be learned efficiently.

We consider here inference for the specific subclass of NFSMs, called *observable NFSMs* (ONFSMs), which have received much attention in a wide range of test generation methods (e.g., [18–20]). An ONFSM could produce different answers to a given input sequence, but its state is uniquely determined by the observation of an input sequence and the corresponding output sequence. This means that, with the same pair of input and output sequences, an ONFSM cannot change to more than one state.

The closest idea to our approach can be found in the work of El-Fakih et al. [21], in which an algorithm for inferring ONFSMs, namely, L_N, has been proposed based on the hypothesis that the *complete testing assumption* [18] (called *all-weather conditions assumption* in [21, 22]) is satisfied. According to this assumption, they assume that the complete set of all possible output sequences is given by the Teacher when applying an input sequence. For this reason, the number of times for repeating the same query is not taken into account in the complexity of their algorithm. However, it seems that how the Teacher constructs the complete set of all possible answers is questionable.

In this paper, we propose L^*_{NM}, a refined algorithm for ONFSM inference which considers the amount of applying the same input sequence (query) as one parameter. Unlike the previous work [21], our approach does not require all possible output sequences in one answer. In contrast, it tries to collect all possible output sequences by asking the same query many times, more precisely k times, to the Teacher. We have proved that L^*_{NM} can infer the corresponding ONFSMs of the unknown systems when the value of k is adequate to satisfy the complete testing assumption. Moreover, since sometimes the assumption may not hold (e.g., insufficient value of k), the termination of our algorithm is still guaranteed. We also present a more rigorous analysis of the worst-case time complexity of the proposed algorithm with respect to the cost of repeating the same query (k). In addition, we studied its practical efficiency using a suite of experiments. Based on the experimental results, we found that the proposed algorithm is applicable and scalable to infer the corresponding ONFSMs of the unknown systems. Moreover, our optimization can effectively reduce the number of queries when applying systems that are known to be equivalent to some (unknown) *partially specified* ONFSMs.

This paper is organized as follows. The next section recalls the basic definitions and notions of NFSMs that will be used throughout this paper. Section 3 describes our proposed algorithm for inferring ONFSMs along with a simple optimization. Moreover, an algorithm analysis on the correctness and the worst-case time complexity is also presented in this section. The experimental results are shown in Section 4. Finally, discussions and conclusions of this study are presented in Section 5.

2. Background

This section briefly recalls the standard notations and related concepts that will be used later in this paper. From this point forward, the term "finite state machines" (FSMs) will be referred to as "Mealy machines," which represent outputs on their transitions.

Definition 1. A non-deterministic finite state machine (NFSM) M is a 5-tuple (Q, I, O, δ, q_0), where Q, I, and O are the nonempty finite sets of *states*, *input symbols*, and *output symbols*, respectively; $q_0 \in Q$ is the *initial state*; and $\delta : Q \times I \rightarrow 2^{Q\times O}$ is the *transition function*, where $2^{Q\times O}$ is the power set of $Q \times O$.

At any point in time, the machine M is currently at state $q \in Q$ and receives an input $i \in I$; it may change to state $q' \in Q$ and produce an output $o \in O$ if and only if $(q', o) \in \delta(q, i)$. For example, Figure 1 shows an NFSM M_0 that is nondeterministic in state q_1 under input b, formally $\delta(q_1, b) = \{(q_2, x), (q_3, y)\}$. This scenario means that if the machine M_0 in state q_1 receives an input b, there are two possible behaviours of M_0: either it changes to state q_2 and outputs x or it changes to state q_3 and outputs y. As usual, the function δ can be extended to take an input sequence; that is, $\delta : Q \times I^* \rightarrow 2^{Q\times O^*}$. For example, here $\delta(q_2, b \cdot a) = \{(q_1, x \cdot y)\}$.

Property 1 (initially connected). An NFSM is *initially connected* if every state $q \in Q$ can be reached from the initial state q_0, that is, for all $q \in Q$, $\exists\, \overline{x} \in I^*$ such that $q \in \delta_Q(q_0, \overline{x})$.

Property 2 (completely specified). An NFSM is *completely specified* if, for all of the states, it has transitions for every input. Formally, for all $q \in Q$, for all $i \in I$, $|\delta(q, i)| \geq 1$.

However, if the machine is not completely specified, called a *partially specified NFSM*, it can be transformed to a completely specified NFSM by adding either a *sink state* or *loop back transition*, with a designated *error symbol* for all inputs that do not occur in the original machine.

Property 3 (observable). An NFSM is *observable*, called ONFSM, if for every state $q \in Q$, input $i \in I$, and output $o \in O$, it has at most one transition leaving q with input i and output o, that is, $|\{q' \in Q \mid (q', o) \in \delta(q, i)(q', o) \in \delta(q, i)\}| \leq 1$.

This property ensures that, with the same input, the machine will never move to different states with the same output. This scenario aids us in determining the target state of the machine by observing only its output. However, as argued in [18, 20], the ONFSM (sometimes called the pseudononde-terministic FSM [23, 24]) is not a deterministic machine due to the fact that we cannot determine the output sequence for a given input sequence.

Property 4 (reduced). An NFSM is *reduced* if it is initially connected and no two states are equivalent. In other words,

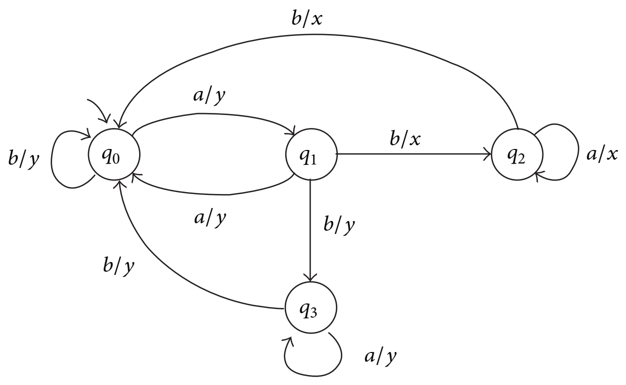

Figure 1: The (observable) non-deterministic finite state machine M_0.

there always exists an input sequence that can distinguish between any two states, that is, for all $q, q' \in Q$ and $\exists \overline{x} \in I^*$, $\delta_O(q, \overline{x}) \neq \delta_O(q', \overline{x})$.

Definition 2 (language). Given an NFSM $M = (Q, I, O, \delta, q_0)$, an associated language of M from a state $q \in Q$, denoted by $\mathscr{L}_M(q)$, is the set of input/output sequences allowed by M from q. More formally, $\mathscr{L}_M(q) = \{\overline{x}/\overline{y} \mid \overline{x} \in I^* \land \overline{y} \in \delta_O(q, \overline{x})\}$. We use $\mathscr{L}(M)$, called the *language* of M, to mean the set $\mathscr{L}_M(q_0)$.

In theoretical sense, the machines considered here are "transducers," not "acceptors." However, the relation between them can be seen as follows.

Remark 3. Given a non-deterministic finite automaton (NFA) $\mathscr{A} = (S, \Sigma, \Delta, s_0, F)$ where S is the nonempty finite set of *states*, $s_0 \in S$ is the *initial state*, $F \subseteq S$ is the set of *final states*, and $\Delta : S \times \Sigma \to 2^S$ is the *transition function*. As usual, one can construct a simple NFSM $M = (Q, I, O, \delta, q_0)$ that has only two outputs, for example, 0 or 1, such that if a string $\overline{x} \in \Sigma^+$ is not in the language of $\mathscr{A}$, then M produces the output ending with 0 when it has finished reading $\overline{x}$ as its input. Otherwise, that means $\overline{x}$ is in the language of $\mathscr{A}$.

From Remark 3, it follows that the computational power of NFSMs is equal to the classical NFAs.

Definition 4 (reduction). Given the two NFSMs $M_1 = (Q^1, I^1, O^1, \delta^1, q_0^1)$ and $M_2 = (Q^2, I^2, O^2, \delta^2, q_0^2)$, where $I^1 = I^2$ and NFSM M_1 is a reduction of NFSM M_2, denoted by $M_1 \preccurlyeq M_2$, if and only if $\mathscr{L}(M_1) \subseteq \mathscr{L}(M_2)$.

Definition 5 (equivalence). The *equivalence* relation between the two NFSMs M_1 and M_2 holds if and only if $M_1 \preccurlyeq M_2$ and $M_2 \preccurlyeq M_1$, that is, $\mathscr{L}(M_1) = \mathscr{L}(M_2)$.

3. Inference of ONFSMs

Even though Angluin's algorithm L^* can efficiently learn an unknown regular language U and produce a minimal DFA that accepts U in polynomial time, its adaptations to FSMs may not be efficient [25]. For example, the direct adaptations can be performed through model transformation techniques by mapping from inputs I and outputs O of the FSM to letters of a DFA's alphabet Σ, such that $\Sigma = I \cup O$ [26] or $\Sigma = I \times O$ [27]. However, these methods are confronted by complexity problems because the cost of L^* is polynomial, based on the size of Σ. Shahbaz and Groz [25] observed that, by slightly modifying the structure of the observation table and the way in which the counterexample is processed, their proposed method, namely, L_M^+, can learn deterministic FSMs, specifically Mealy machines, more effectively.

As usual in the setting of L^*-based algorithms, a learning algorithm, called *Learner*, needs to ask two types of questions to a *Minimally Adequate Teacher*, called *Teacher* for short, which is assumed to correctly answer the questions. The first type of question is called a *membership query* in L^*, which consists of a string σ from Σ^*. The Teacher replies either *true* if $\sigma \in U$ or *false* otherwise. Later, this concept is adapted to the *output query* [21, 25], which consists of a string σ from I^+. The difference is that, instead of true or false, the Teacher replies with the output string from O^+, which will be processed and recorded in an *observation table.*

The second type of question is called an *equivalence query*, which consists of a candidate DFA M, whose language the Learner believes to be identical to U (i.e., $\mathscr{L}(M) = U$) in the case of L^*, or a candidate Mealy machine M, whose language the Learner believes to be identical to the language of an unknown Mealy machine M_U (i.e., $\mathscr{L}(M) = \mathscr{L}(M_U)$) in the case of L_M^+). The answer is true if it is a correct conjecture; otherwise, the Teacher returns a counterexample, which is a string in the symmetric difference of $\mathscr{L}(M)$ and U in L^* or $\mathscr{L}(M)$ and $\mathscr{L}(M_U)$ in L_M^+.

In our setting, the algorithm asks each output query many times to collect all possible output sequences. Unlike [21], there is no need to modify the Teacher to answer each query with the complete set of all possible output sequences. Note that, after each query, the unknown machine must be returned to the initial state by a *reset* input.

In order to infer a black-box ONFSM, it is necessary to make a *complete testing assumption* [18] (also called fairness assumption [20] and all-weather conditions assumption [21, 22]).

Definition 6 (complete testing assumption). For a given black-box ONFSM, there is some unknown number k such that, if an input sequence is applied to the target ONFSM k times, then all possible responses are observed.

The idea behind the assumption is to bound how many outputs a target machine can produce given some input sequence. As a result, we need to ask the same output query k times for each input/output sequence to observe every possible output sequence from an unknown ONFSM (if the complete testing assumption holds for k).

Let $M = (Q, I, O, \delta, q_0)$ be an unknown ONFSM that is initially connected, completely specified, and reduced. A detailed description of an observation table and the procedure of L_{NM}^* will be described in this section.

3.1. Observation Table. At a higher level, the observation table is composed of two parts: an upper and a lower part. Each row in the upper part represents a candidate state of the unknown

Table 1: Example of an observation table.

T_1	E	
	a	b
S		
ϵ	$\{y\}$	$\{y\}$
$S \cdot I/O$		
a/y	$\{y\}$	$\{x, y\}$
b/y	$\{y\}$	$\{y\}$

machine, while each row in the lower part represents the target state of a candidate state and an input. Formally, the structure of an observation table (denoted by (S, E, T)) of the algorithm L^*_{NM} consists of three parts: S, E, and T, where

(i) S is the non-empty finite set of prefix-closed input/output sequences $\overline{x}/\overline{y}$, where $\overline{x} \in I^*$, $\overline{y} \in O^*$, and S always contains the *empty sequence* ϵ.

(ii) E is the non-empty finite set of suffix-closed input sequences from I^+.

(iii) T is a finite function that maps $(S \cup S \cdot I/O) \times E$ to a set of output sequences from $2^{O^{|E|}}$.

Intuitively, an observation table can be visualized as a two-dimensional array with rows labelled by elements of S and $S \cdot I/O$ (i.e., $S \cup S \cdot I/O$) and columns labelled by elements of E. The entry corresponding to row s in $(S \cup S \cdot I/O)$ and column e in E equals $T(s, e)$, which contains the set of the output sequences from $\text{suff}^{|e|}(\delta_O(q_0, s \cdot e))$, where $\text{suff}^{|k|}(\mathcal{S})$ denotes the set of k-length suffixes of every sequence from a set $\mathcal{S}$. For example, let $\mathcal{S} = \{y \cdot x \cdot x, y \cdot x \cdot y, y \cdot y \cdot y\}$, $\text{suff}^{|1|}(\mathcal{S}) = \{x, y\}$ and $\text{suff}^{|2|}(\mathcal{S}) = \{x \cdot x, x \cdot y, y \cdot y\}$.

Definition 7 (row equivalence). Let s, t be two rows in the table (S, E, T), that is, $s, t \in S \cup S \cdot I/O$. Then, s and t are *row equivalent*, denoted by $s \cong_E t$, if and only if $T(s, e) = T(t, e)$ for all $e \in E$. Moreover, we used $[s]$ to denote the *equivalence class* of rows that are row equivalent to s.

For example, Table 1 is an example of the observation table used for learning the NFSM M_0 in Figure 1. From this table, row ϵ is equivalent to b/y (i.e., $\epsilon \cong_E b/y$) but is not equivalent to a/y (i.e., $\epsilon \ncong_E a/y$).

Definition 8 (closed observation table). An observation table is called *closed* if and only if, for each $t \in S \cdot I/O$, there exists an $s \in S$ such that $s \cong_E t$.

For example, Table 1 is not closed because $a/y \in S \cdot I/O$ but for all $s \in S, s \ncong_E a/y$. However, Table 2 is a closed observation table because, for each row t in $S \cdot I/O$, there exists a row s in S such that $s \cong_E t$.

From the closed observation table, we can construct an ONFSM conjecture as follows.

Definition 9 (ONFSM conjecture). Given a closed observation table (S, E, T), L^*_{NM} obtains an ONFSM conjecture $M = (Q, I, O, \delta, q_0)$, where

Table 2: Closed observation table.

T_1	E	
	a	b
S		
ϵ	$\{y\}$	$\{y\}$
a/y	$\{y\}$	$\{x, y\}$
$a/y \cdot b/x$	$\{x\}$	$\{x\}$
$S \cdot I/O$		
b/y	$\{y\}$	$\{y\}$
$a/y \cdot a/y$	$\{y\}$	$\{y\}$
$a/y \cdot b/y$	$\{y\}$	$\{y\}$
$a/y \cdot b/x \cdot a/x$	$\{x\}$	$\{x\}$
$a/y \cdot b/x \cdot b/x$	$\{y\}$	$\{y\}$

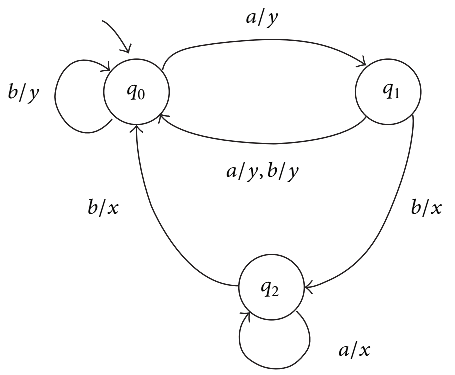

Figure 2: The ONFSM conjecture $M_0^{(1)}$ from Table 2.

(i) $q_0 = [\epsilon]$,

(ii) $Q = \{q_i | \ q_i = [s]$ for $1 \le i \le |S| - 1$, for all $s \in S \land s \neq \epsilon\}$,

(iii) $\delta(q, i) = \{(q', o) | \ q = [s], q' = [s \cdot i/o]$, for all $s \in S$, for all $i \in I$, for all $o \in T(s, i)\}$.

The conjecture $M_0^{(1)}$ (the superscript 1 means that it is the conjecture from the first learning iteration of the machine M_0) shown in Figure 2 is constructed from Table 2, which is a closed observation table according to Definition 9.

Theorem 10. *Let (S, E, T) be a closed observation table, and let M be the ONFSM conjecture that is constructed from (S, E, T). The conjecture M is consistent with the finite function T. Any other ONFSM that is consistent with T but inequivalent to M must have more states.*

Proof. Since the observation table (S, E, T) in the setting of L^*_{NM} preserves the prefix-closed and suffix-closed properties of S and T, respectively, the conjecture is proven to be consistent with the observation table that has been given by Niese [28]. Moreover, because the conjecture M is the reduced ONFSM by construction, any other ONFSM that is consistent with T but not equivalent to M must have at least one more state. □

3.2. The Algorithm. We now describe L^*_{NM}, which takes a set of input symbols I and the number of repeated queries k as input. Its pseudocode is given in Algorithm 1.

The algorithm starts by initializing an observation table (S, E, T) with $S = \{\epsilon\}$ and $E = I$. Then, it asks the output queries to fill the upper part of the table, that is, $T(\epsilon, e)$, for all $e \in E$ (line 1). Next, it uses the observed outputs from the upper part to construct the output queries to fill the lower part, namely, $S \cdot I/O$, of the table, that is, $T(\epsilon \cdot i/o, e)$, for all $i \in I$, for all $e \in E$, for all $o \in T(\epsilon, i)$ (line 2).

After initializing the table, L^*_{NM} repeatedly checks whether the current table is closed (line 4). If it is not closed, then there exists row $t \in S \cdot I/O$ such that $t \not\cong_E s$ for all $s \in S$. Then, L^*_{NM} finds and moves row t to S (line 5). Next, $t \cdot i/o$ is added to $S \cdot I/O$, and $T(t \cdot i/o, e)$ is determined by the output queries for all $i \in I$, $e \in E$, $o \in T(t, i)$ (line 6).

When the table is closed, L^*_{NM} makes an ONFSM conjecture M from the table according to Definition 8 and asks it to the Teacher (line 8). The Teacher replies either *yes*, acknowledging that the conjecture is correct, or with a counterexample. If the Teacher says *yes*, then L^*_{NM} terminates with the correct ONFSM M (line 16). Otherwise, the Teacher replies with a counterexample. The counterexample is analyzed as to whether it is false (line 10). If it is a false counterexample, then the procedure terminates; otherwise, it will be used for extending the table accordingly (lines 12-13). The method for processing a counterexample will be described in the next subsection. With the extended table, the algorithm repeats the checking loop (lines 4–6) again until the table is closed, followed by making a new conjecture.

Note that, according to the complete testing assumption, for each output query, L^*_{NM} must ask the same query k times (lines 1, 2, 6, and 13) to explore every possible output from the unknown system.

3.3. Counterexample. To the best of our knowledge, the crucial improvement in the methods for processing counterexamples of the original Angluin's algorithm L^* was proposed by Rivest and Schapire [29]. They observed that the handling of counterexamples as in L^* could lead to *inconsistency* in an observation table (S, E, T). Informally, the table is inconsistent if two (or more) rows in the upper part of the table that represent the same *potential state in the conjecture* have different target states when applied to some inputs. More precisely, $\exists s, t \in S$ and $\exists i \in I$, such that $s \cong_E t$ but $s \cdot i \not\cong_E t \cdot i$. This scenario implies that the rows s and t must be distinguished. Fortunately, Rivest and Schapire suggested that, by adding a distinguishing sequence from the counterexample to the set E, inconsistency will never occur. The reason is that the method will never directly add a new row to S, and consequently, the rows in S will remain inequivalent. Furthermore, this condition will always hold trivially. However, the method requires a relaxation on the prefix-closed and suffix-closed properties of the table. For more details and proofs of the method, interested readers can refer to the original paper [29].

In [25], Shahbaz and Groz modified the method for processing the counterexample based on Rivest and Schapire's idea. Their method starts by finding the longest prefix of the counterexample that has already been observed in the table, that is, $S \cup S \cdot I$. Then, the remaining string and all of its suffixes are added to E. Unlike the previous methods, the observation table preserves the prefix-closed and suffix-closed properties, and, therefore, the constructed conjecture is proved to be consistent with the table.

Our treatment of counterexamples is adapted straight from [25]. Let ce be the counterexample for the current conjecture. We find the longest prefix $u \in S \cup S \cdot I/O$ of ce such that $\text{ce} = u \cdot v$, and $v = \overline{x}/\overline{y}$ is the remaining input/output sequence of ce. Then, we add the input sequence of v, $\overline{x}$, and all of its suffixes to E.

We have observed that, when fixing a counterexample in this setting, the table preserves the prefix-closed and suffix-closed properties of S and E, respectively. Thus, the output conjecture is proved to be consistent with the observation table.

3.4. Correctness. As usual in an active learning procedure, our algorithm asks increasingly longer output queries to the Teacher to observe all of the possible states of an unknown machine, and the corresponding sets of output sequences are recorded in an observation table (S, E, T). According to the structure of the observation table, the set S contains uniquely potential states of the conjecture, and the set E contains the sequences that can be used to distinguish these states from each other. This scenario means that every row in S can be distinguished when applying some $e \in E$. In other words, any rows in the table (i.e., $S \cup S \cdot I/O$) that represent the same state must not be distinguished by any sequences in E.

In the case of a deterministic machine, when the same states are applied by any distinguishing sequences, the machine always responds with the same set of output sequences. However, this scenario is not always the case for a non-deterministic machine. The reason is that if the complete testing assumption does not hold, the Learner may observe a different set of output sequences when the state is applied more than once by the same input/output sequence. This situation could lead the Learner to infer an incorrect conjecture. Nevertheless, the learning procedure will always terminate, which will be proved as follows.

Proposition 11. *Suppose that $M_U = (Q, I, O, \delta, q_0)$ is an unknown ONFSM. Let q_i be a state in Q, and let $O_{q_i,a}$ be a set of possible outputs of a state q_i under an input a in I. Clearly, $O_{q_i,a} \subseteq O$. Let $L_{q_i,a}$ be the set of all combinations of outputs of the state q_i under the input a. Then, $L_{q_i,a} = 2^{O_{q_i,a}} \setminus \{\emptyset\}$ and $|L_{q_i,a}| = 2^{|O_{q_i,a}|} - 1$.*

Proposition 11 claims that the number of possible distinct output sets that can be observed and added to the observation table is finite.

Theorem 12. *Given an unknown ONFSM $M_U = (Q, I, O, \delta, q_0)$, L^*_{NM} will eventually provide a closed table (S, E, T) in each iteration, regardless of the complete testing assumption.*

Proof. Now assume that s is a row in S and t is a row in $S \cdot I/O$ and that they represent the same state q_i in Q. Therefore,

```
input: A set of input symbols $I$, the number of repeated queries $k$
output: ONFSM conjecture $M$
      // Construct the initial observation table $(S, E, T)$
(1)  set $S = \{\epsilon\}$, $E = I$, and update $T$ using output queries by asking each query $k$ times;
(2)  add $\epsilon \cdot i/o$ to $S \cdot I/O$ for all $i \in I, o \in T(\epsilon, i)$, and update $T$ using output queries by asking each query $k$ times;
(3)  repeat
       // Check whether the table is closed
(4)    while found $t \in S \cdot I/O$ such that $t \not\cong_E s$, for all $s \in S$ do
(5)      move $t$ to $S$;
(6)      add $t \cdot i/o$ to $S \cdot I/O$ for all $i \in I, o \in T(t, i)$, and update $T$ using output queries by asking each query $k$ times;
(7)    end
(8)    make the ONFSM conjecture $M$ from $(S, E, T)$;
(9)    if the Teacher replies with a counterexample ce then
(10)     if any prefix of ce has been recorded in T with a different value, then terminate with no solution;
(11)     else
(12)       find the longest $u \in S \cup S \cdot I/O$ such that $ce = u \cdot v$;
(13)       add the input sequence of $v$ and all of its suffixes to $E$, and update $T$ using output queries by asking each query $k$ times;
(14)     end
(15)   end
(16) until the Teacher replies "yes";
(17) return the conjecture $M$;
```

Algorithm 1: The algorithm L^*_{NM}.

$s \cong_E t$, which means that $T(s, e) = T(t, e)$ must hold for all e in E with respect to Definition 7. If the complete testing assumption holds, then we know that $\delta_O(q_i, a) = O_{q_i,a}$ for all $a \in I$ and for all $q_i \in Q$. Since e is I^+, we have $T(s, e) = T(t, e) = \delta_O(q_i, e)$. Thus, the table is closed.

In contrast, when the complete testing assumption does not hold, we know that $\delta_O(q_i, a) \subseteq O_{q_i,a}$. Thus, there could exist an e in E such that $T(s, e) \neq T(t, e)$; that is, the different subsets of $O_{q_i,a}$ have been observed as outputs for $T(s, e)$ and $T(t, e)$. This scenario leads the Learner to consider moving row t, which represents a spurious state, to S. Thus there are two possible cases as follows.

(i) If t is not a new row in S, then the table is now closed.

(ii) Otherwise, row t is moved to S, and the learning process can continue. In this case, the number of the remaining elements in $L_{q_i,a}$ must *decrease* by at least one for each iteration.

By Proposition 11, the set $L_{q_i,a}$ is finite. Thus, the maximum number of spurious states for a state q_i for all inputs is bounded by $\sum_{a \in I} |L_{q_i,a}|$, which is also finite. As a result, from (i) and (ii), the learning process eventually terminates with a closed table. □

Theorem 13. *Given an unknown ONFSM $M_U = (Q, I, O, \delta, q_0)$, let M be a corresponding conjecture that is constructed from a closed table (S, E, T) in each iteration. When L^*_{NM} terminates, if the complete testing assumption holds, then M is guaranteed to be isomorphic with M_U.*

Proof. Theorem 12 ensures that L^*_{NM} always provides a closed table in each iteration. Whenever the table is closed, the corresponding ONFSM conjecture is constructed based on Definition 9. Since, by Theorem 10, the conjecture M is consistent with the finite function T. For the case in which the complete testing assumption holds, according to the correctness of the Teacher's answer for the equivalence query, we either obtain a counterexample from the conjecture for extending the table, or the learning procedure terminates with a correct conjecture that is isomorphic to M_U. □

Note that, when the complete testing assumption does not hold, spurious states could be recorded as some rows in the table. Thus, the conjecture M, which is consistent with the table, could also have these spurious states. With respect to the correctness of the answer for the equivalence query, if M contains the spurious states, then L^*_{NM} terminates. In summary, our algorithm does not necessarily provide an ONFSM that is isomorphic to M_U in this case.

3.5. Complexity. We analyze a theoretical upper bound for the number of output queries asked by L^*_{NM}. Similar to the membership queries of L^* or the output queries of L^+_M, the maximum number of output queries also corresponds to the worst-case size of the observation table.

Let $|I|$ and $|O|$ be the sizes of the input set I and the output set O, respectively. Let n be the number of states of the ONFSM, and let m be the maximum length of any counterexamples that are provided by the Teacher for equivalence queries. The size of the table has at most $n + n|I||O|$ rows (n rows in the upper part + their successors) and $|I| + m(n-1)$ columns because E contains $|I|$ elements initially, and at most m suffixes of the maximum $n-1$ counterexamples are added. In addition, with respect to complete testing assumption, each query must be asked k times to observe every possible output. Thus, L^*_{NM} produces a correct conjecture by asking a maximum of $k(S \cup S \cdot I/O) \times E = O(kn|I|^2|O| + kmn^2|I||O|)$ output queries.

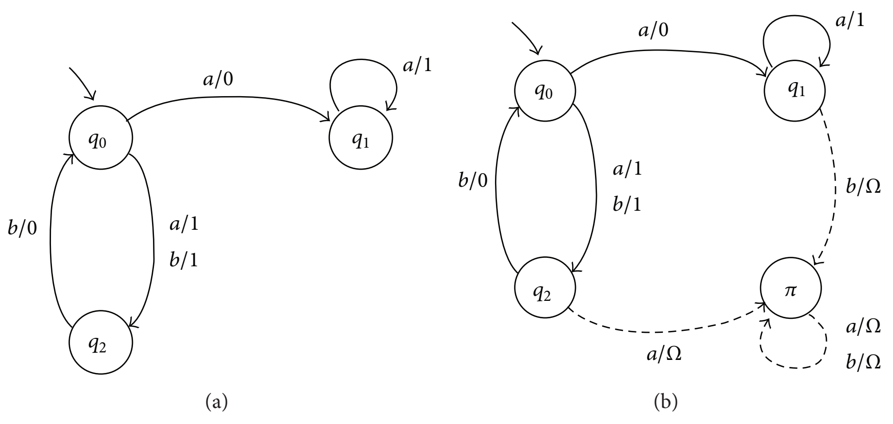

Figure 3: A partially specified NFSM (a) and the corresponding completely specified NFSM (b).

3.6. Optimization. As mentioned in [25], reactive systems can be naturally modeled as (non-deterministic) finite state machines. These models are very useful for checking some properties before implementing the system or testing whether the implementation conforms to the specification models. However, these models might come up with *partial* transition relations [30]. To apply our method, one necessary assumption is that the FSM models must be *completely specified*.

In this paper, any (non-deterministic) FSM can be transformed into a completely specified FSM by adding a sink state that loops itself for all inputs, that is, a state that has no outgoing transition to other states, and adding transitions for the missing inputs from any states in the original FSM to the sink state, with a designated error output symbol.

Consider an ONFSM example, shown in Figure 3, in which the input symbols are $\{a, b\}$ and the output symbols are $\{0, 1\}$. Here, an ONFSM on the left side is partially specified because it is missing input b of state q_1 and input a of state q_2. Thus, a sink state π is introduced, and new transitions are added between state q_1 under input b and state q_2 under input a to the sink state, as shown in Figure 3(b). In Figure 3 , an error output symbol is represented by Ω.

Thus, if we know that any sequence t will lead the machine to enter the sink state, then every sequence that has t as its prefix will also lead the machine to enter the sink state. We can then use this characteristic to reduce the number of output queries asked to the Teacher. Before asking each query, the Learner must first test whether it is an extension of an input/output sequence that has already been observed with an error output. If so, the Learner can then immediately record the result of the query as an error in the table.

Note that, when we obtain a correct conjecture, which is a completely specified ONFSM with a sink state, from L^*_{NM}, it can be transformed back to the original machine easily by removing the sink state and all of the transitions that lead to it.

3.7. Example. We illustrate the algorithm L^*_{NM} on the ONFSM M_0 given in Figure 1. The algorithm initializes (S, E, T) with $S = \{\epsilon\}$ and $E = I = \{a, b\}$. Moreover, we set $k = 10$ in this example. Then, it asks the output queries to fill the upper part of the table, that is, $T(\epsilon, a) = y$ and $T(\epsilon, b) = y$. Next, it uses the known outputs to construct the queries to fill the lower part of the table. The initial table is shown in Table 1.

Table 3: Processing the counterexample $a/y{\cdot}b/y{\cdot}a/y{\cdot}b/x$ for $M_0^{(1)}$.

(a) Adding the suffixes of $a \cdot b$ to E

T_2	E		
	a	b	$a \cdot b$
S			
ϵ	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
a/y	$\{y\}$	$\{x, y\}$	$\{y \cdot y\}$
$a/y \cdot b/x$	$\{x\}$	$\{x\}$	$\{x \cdot x\}$
$S \cdot I/O$			
b/y	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
$a/y \cdot a/y$	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
$a/y \cdot b/y$	$\{y\}$	$\{y\}$	$\{y \cdot y\}$
$a/y \cdot b/x \cdot a/x$	$\{x\}$	$\{x\}$	$\{x \cdot x\}$
$a/y \cdot b/x \cdot b/x$	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$

(b) Moving the rows $a/y \cdot b/y$ to S

T_2	E		
	a	b	$a \cdot b$
S			
ϵ	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
a/y	$\{y\}$	$\{x, y\}$	$\{y \cdot y\}$
$a/y \cdot b/x$	$\{x\}$	$\{x\}$	$\{x \cdot x\}$
$a/y \cdot b/y$	$\{y\}$	$\{y\}$	$\{y \cdot y\}$
$S \cdot I/O$			
b/y	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
$a/y \cdot a/y$	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
$a/y \cdot b/x \cdot a/x$	$\{x\}$	$\{x\}$	$\{x \cdot x\}$
$a/y \cdot b/x \cdot b/x$	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$
$a/y \cdot b/y \cdot a/y$	$\{y\}$	$\{y\}$	$\{y \cdot y\}$
$a/y \cdot b/y \cdot b/y$	$\{y\}$	$\{y\}$	$\{y \cdot x, y \cdot y\}$

When the initial table is filled, L^*_{NM} repeatedly tests whether the table is closed. Table 1 is not closed because the row a/y in $S\cdot I/O$ is not equivalent to any row in S. Therefore, the algorithm moves the row a/y to S and extends the table by adding $a/y\cdot a/y, a/y\cdot b/x$ and $a/y\cdot b/y$ to $S\cdot I/O$. Then, the queries are constructed for the missing elements of the observation table.

The new table is closed, as shown in Table 2; consequently, L^*_{NM} makes a conjecture $M_0^{(1)}$ from the table, which is shown in Figure 2. Because the conjecture $M_0^{(1)}$ is not correct, the Teacher replies with a counterexample *ce*. In this case, we assume that the counterexample *ce* is $a/y\cdot b/y\cdot a/y\cdot b/x$ (since $a/y\cdot b/y\cdot a/y\cdot b/x \in \mathscr{L}(M_0^{(1)})$, but $a/y\cdot b/y\cdot a/y\cdot b/x \notin \mathscr{L}(M_0))$.

According to the method for processing the counterexample, L^*_{NM} adds $a\cdot b$, which is the remaining input sequence of *ce*, and all of its suffixes, that is, b and $a\cdot b$, to E, as shown in Table 3(a). This table is not closed because the row $a/y\cdot b/y$ is not equivalent to any rows in S. Thus, the row $a/y\cdot b/y$ is moved to S, and the table is extended accordingly. The resulting table after filling in the missing elements by asking output queries is Table 3(b).

Next, L^*_{NM} checks whether Table 3(b) is closed. This table is closed, so L^*_{NM} constructs a new conjecture that is isomorphic to M_0. Thus, the Teacher replies *yes* to this conjecture, and L^*_{NM} terminates with the correct conjecture as its output. The total number of output queries asked by the algorithm during this run is 300.

4. Experiments

We have performed a suite of experiments to demonstrate the applicability and scalability of our algorithm in practice. This suite is composed of (i) nine samples of (partially and completely specified) ONFSMs, either inspired by different papers [20, 31] or specifically designed, and (ii) random (partially specified) ONFSMs with arbitrary sizes for the number of states. Furthermore, we have implemented our algorithm in Java, together with our proposed optimization.

We have also simulated the Teacher to answer the equivalence query by using the model checker *Labelled Transition System Analyser* (LTSA) [32]. To apply the LTSA, we first transform the ONFSM to the corresponding *Labelled Transition System* (LTS). The transformation technique is straightforwardly modified from [33]. Then, the model checking tool checks the *trace equivalence relation* between two corresponding LTSs of the learned ONFSM and the target ONFSM.

As mentioned in [34], the performance of the Teacher in answering an equivalence query depends on the method that is used to realize it. Thus, the time spent by the equivalence query is disregarded from the measurement. To evaluate the execution time of the algorithm, we measured the total execution time except for the time utilized for the equivalence queries.

The experiments were conducted using a Windows 7 system with an Intel Core i5, 2.67 GHz and 4 GB of memory, and LTSA version 3.0. In addition, the Learner and the Teacher were running on the same machine.

Table 4: Runs from 9 sample machines.

Machines	No. of states	k	Output queries	EQ	Avg. time (ms)
M1	3	1	14	1	7.2
M2	3	8	144	1	103.1
M3	4	7	140	1	65.9
M4	4	10	260	1	363
M5	4	7	154	1	95.8
M6	6	11	462	2	251.9
M7	6	6	336	3	176.1
M8	7	10	510	2	307.6
M9	8	10	570	2	329.4

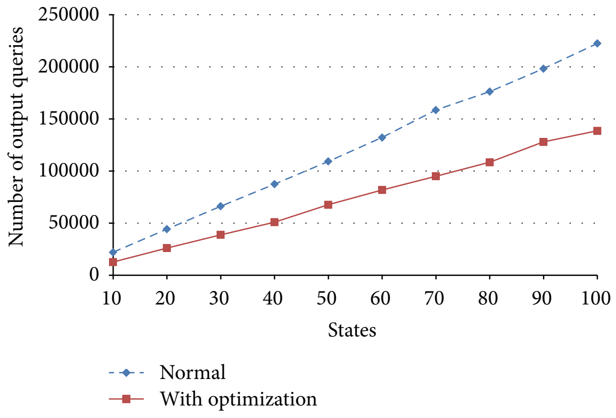

Figure 4: Random ONFSM examples learned with normal L^*_{NM} and with optimization, using $|I| = 10$, $|O| = 5$, and $k = 20$.

4.1. Sample Machines. The first set of experiments was conducted on nine sample FSMs, that is, one DFSM and eight ONFSMs, to evaluate our algorithm. All of the examples are different sizes in terms of the number of states. Moreover, we started with the number of repeated queries (k) to 1 and learned 10 times for each machine.

4.1.1. Experiences. The learned machines are isomorphic to the original FSMs, as expected. With respect to the number of states of each sample machine (number of States) and the number of repeated queries (k), which can guarantee the complete testing assumption, Table 4 shows the experimental results, including the number of used output queries (output queries), the number of used equivalence queries (EQ), and the average execution time in milliseconds (Avg. time).

From the table, L^*_{NM} can be applied with both DFSMs (e.g., M1) and ONFSMs (e.g., M2–M9). In addition, $k = 1$ is obviously sufficient for learning any DFSM. Note that the efficiency of the algorithm not only depends on the number of states and the size of the input alphabet, but it also depends on the value of k. Let us consider machines M2 and M3, which have 3 and 4 states, respectively. Learning M3, which has more states, is expected to require more output queries; however, because the value of k for inferring M3 is less than that for inferring M2, learning M2 requires slightly more queries than M3.

TABLE 5: Comparison of normal L^*_{NM} with the optimized version using the random ONFSM examples.

No. of states	Output queries	Output queries (with opt.)	Saved queries (%)
10	22000	12620	43
20	44200	26020	41
30	66200	38760	41
40	87400	50900	42
50	109200	67640	38
60	132200	81760	38
70	158552	94950	40
80	176202	108366	38
90	198200	127960	35
100	222402	138546	38

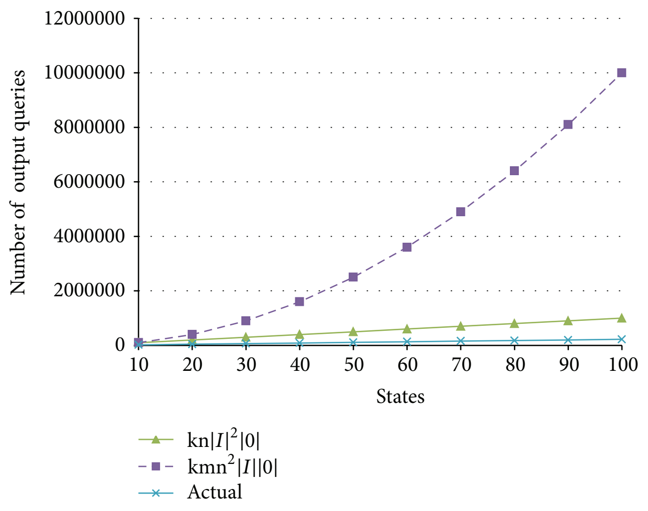

FIGURE 5: Comparison of the actual number of output queries of the normal L^*_{NM} algorithm and the theoretical upper bound on the random ONFSM examples.

4.2. Random Arbitrary Machines. Apart from the sample machines, we also performed a second set of experiments on random examples by varying the number of states. Specifically, we generated and learned ONFSMs with sizes ranging between 10 and 100 states (in steps of 10), with an input size of 10 and an output size of 5. For each number of states n, we randomly generated 10 ONFSMs, which have $n - 1$ states plus one sink state, to observe the effectiveness of our optimization.

First, we fixed the number of repeated queries to 5. However, we found that we cannot guarantee the complete testing assumption with this value. To compare the scalability of L^*_{NM}, we varied the number of states of the target machines and fixed the number of repeated queries. Thus, in this experiment, we set the number of repeated queries to 20, and we leave the topic of how to define this value to be discussed in the next section.

4.2.1. Experiences. We observed that the number of output queries relative to the number of states is linear and conforms to the part $kn|I|^2|O|$ in the complexity calculation (see Figure 4, in which we vary the number of states but fix the other factors such that $|I| = 10$, $|O| = 5$, and $k = 20$).

The number of output queries is reduced by an average of approximately 39% using the optimized version compared to the basic L^*_{NM} algorithm. Moreover, the best reduction that we achieved in this setting was 43% in an ONFSM with 10 states. With the specific example of a size of 100 states, the optimized Learner took approximately 10 minutes with 138,546 output queries, a reduction of approximately 38%. The detailed results can be found in Table 5, in which it can be seen that the optimized Learner performs better in every case. This scenario might indicate that, for an ONFSM with a certain structure, we can make the algorithm perform better through our optimization.

Note that, because the number of equivalence queries in the optimized version does not change from the number in the basic algorithm, we do not report the query in this experiment.

Interestingly, when we plotted a graph to study the relationship between the actual number and the theoretical upper bound of the Output queries, as shown in Figure 5, we observed that the part $kn|I|^2|O|$ of the calculated upper bound is closer to the experimental results than the other part, that is, $kmn^2|I||O|$. The reason for this similarity is that the Learner in our setting asks a few equivalence queries in practice. Thus, a small number of columns will be added to the observation table; that is, the maximum number of columns is $|I| + \varepsilon$, where ε is a small integer.

5. Discussion and Conclusions

This work has presented a refined algorithm L^*_{NM} for ONFSMs inference from unknown non-deterministic systems. In contrast to the previous approach [21], L^*_{NM} does not require complicated answers from the Teacher. It tries to collect all possible output sequences by asking the same query many times. Thus the number of repeated queries (k) is considered as one factor in our approach. We have proved that the algorithm will eventually terminate no matter whether a complete testing assumption is satisfied or not. However, the correct ONFSM conjecture can be inferred if the value of k is sufficient to satisfy the assumption.

From the performance perspective, the algorithm L^*_{NM} can infer the corresponding ONFSMs of the unknown non-deterministic systems efficiently (i.e., in polynomial time). Our method uses $O(kn|I|^2|O| + kmn^2|I||O|)$ output queries to learn an ONFSM which has n states, $|I|$ inputs, and $|O|$ outputs. Moreover, the evaluation results indicate that the number of repeated queries k affects the performance of our algorithm.

In addition, when dealing with systems that are known to be equivalent to some (unknown) partially specified ONFSMs, the algorithm offers a faster run by our proposed optimization. Based on the experimental results, the optimization can reduce the number of output queries by approximately 39% on average.

To answer the question of how to define the value k to guarantee the complete testing assumption, we used the initial value of k to be a small integer (e.g., $k = |O|$) in our experiments. Because our algorithm is assured to terminate regardless of the complete testing assumption, we eventually obtain a conjecture ONFSM. Using the equivalence query, we do receive either the answer *yes* or a counterexample from the Teacher. There are two cases of the counterexamples. (a) The counterexample has not been recorded in the table. Then it will be used to extend the table, as described in Section 3.3. (b) The counterexample has already been observed in the table but the recorded answer is not the same (line 10 in Algorithm 1). This situation means that we cannot explore every possible output of the machine with the current value of k. Therefore, we restart the algorithm with an increased value of k. Although this process can be run incrementally, performing incremental steps appears to be inefficient. Thus, it is a challenge to obtain a method for selecting the most appropriate value of k. This is for the reason that the value of k may not necessarily be minimal but it is sufficient to ensure the complete testing assumption. Moreover, since the value of k is relative to the number of possible outputs, it is easy to particularly construct ONFSMs that cannot be learned efficiently such as ones where the number of possible outputs grows exponentially with the length of the input. However, to the best of our knowledge, the case is rare in practice.

Our current research works focus mainly on methods for applying a learning approach to automatic verification in software engineering. Because these methods may not be suitable with practical applications, we intend to continue our future research in this direction to obtain further improvements.

Acknowledgment

This work was partially funded by CU. Graduate school thesis grant.

References

[1] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," in *Proceedings of the 12th Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE '99)*, pp. 225–240, Kluwer Academic Publishers, 1999.

[2] A. Groce, D. Peled, and M. Yannakakis, "Adaptive model checking," in *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '02)*, pp. 357–370, Springer, 2002.

[3] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu, "Learning assumptions for compositional verification," in *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '03)*, pp. 331–346, Springer, 2003.

[4] S. Chaki, E. Clarke, N. Sharygina, and N. Sinha, "Verification of evolving software via component substitutability analysis," *Formal Methods in System Design*, vol. 32, no. 3, pp. 235–266, 2008.

[5] S. M. Lucas and T. J. Reynolds, "Learning DFA: evolution versus evidence driven state merging," in *Proceedings of the Congress on Evolutionary Computation (CEC '03)*, vol. 1, pp. 351–358, 2003.

[6] S. M. Lucas and T. J. Reynolds, "Learning deterministic finite automata with a smart state labeling evolutionary algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1063–1074, 2005.

[7] F. Tsarev and K. Egorov, "Finite state machine induction using genetic algorithm based on testing and model checking," in *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO '11)*, pp. 759–762, ACM, July 2011.

[8] V. Ulyantsev and F. Tsarev, "Extended finite-state machine induction using SAT-solver," in *Proceedings of the 10th International Conference on Machine Learning and Applications (ICMLA '11)*, pp. 346–349, IEEE Computer Society, December 2011.

[9] D. Chivilikhin and V. Ulyantsev, "Learning finite-state machines with ant colony optimization," in *Swarm Intelligence*, vol. 7461 of *Lecture Notes in Computer Science*, pp. 268–275, Springer, 2012.

[10] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[11] J. Oncina and P. Garcia, "Inferring regular languages in polynomial update time," in *Pattern Recognition and Image Analysis*, pp. 49–61, 1992.

[12] D. Angluin and M. Kharitonov, "When won't membership queries help?" in *Proceedings of the 23th Annual ACM Symposium on Theory of Computing (STOC '91)*, pp. 444–454, ACM, New York, NY, USA, 1991.

[13] T. Yokomori, "Learning nondeterministic finite automata from queries and counterexamples," in *Machine Intelligence*, vol. 13, pp. 169–189, Oxford University Press, New York, NY, USA, 1995.

[14] F. Denis, A. Lemay, and A. Terlutte, "Learning regular languages using non deterministic finite automata," in *Grammatical Inference: Algorithms and Applications*, Lecture Notes in Computer Science, pp. 213–214, Springer, 1891.

[15] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of NFA," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pp. 1004–1009, July 2009.

[16] F. Coste and D. Fredouille, "Unambiguous automata inference by means of state-merging methods," in *Machine Learning*, vol. 2837 of *Lecture Notes in Computer Science*, pp. 60–71, Springer, 2003.

[17] M. Shahbaz, K. Li, and R. Groz, "Learning and integration of parameterized components through testing," in *Testing of Software and CommunicatIng Systems*, vol. 4581 of *Lecture Notes in Computer Science*, pp. 319–334, Springer, 2007.

[18] G. Luo, G. V. Bochmann, and A. Petrenko, "Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method," *IEEE Transactions on Software Engineering*, vol. 20, no. 2, pp. 149–162, 1994.

[19] F. Ipate, "Bounded sequence testing from non-deterministic finite state machines," in *Proceedings of the 18th International Conference on Testing of Communicating Systems (TestCom '06)*, vol. 3964 of *Lecture Notes in Computer Science*, pp. 55–70, Springer, 2006.

[20] R. M. Hierons, "Testing from a nondeterministic finite state machine using adaptive state counting," *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 1330–1342, 2004.

[21] K. El-Fakih, R. Groz, M. N. Irfan, and M. Shahbaz, "Learning finite state models of observable nondeterministic systems in a testing context," in *Proceedings of the 22nd IFIP International*

Conference on Testing Software and Systems: Short Papers, pp. 97–102, 2010.

[22] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *Lecture Notes in Computer Science*, Springer, 1980.

[23] Y. Watanabe and R. K. Brayton, "State minimization of pseudo non-deterministic FSM's," in *Proceedings of the European Design and Test Conference*, pp. 184–191, March 1994.

[24] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Theory and algorithms for state minimization of nondeterministic FSM's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1311–1322, 1997.

[25] M. Shahbaz and R. Groz, "Inferring Mealy machines," in *Proceedings of the 2nd World Congress on Formal Methods (FM '09)*, pp. 207–222, Springer, 2009.

[26] H. Hungar, O. Niese, and B. Steffen, "Domain-specific optimization in automata learning," in *Computer Aided Verification*, vol. 2725 of *Lecture Notes in Computer Science*, pp. 315–327, Springer, 2003.

[27] E. Makinen and T. Systa, "MAS—an interactive synthesizer to support behavioral modelling in UML," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE '01)*, pp. 15–24, IEEE Computer Society, Washington, DC, USA, 2001.

[28] O. Niese, *An integrated approach to testing complex systems [Ph.D. thesis]*, University of Dortmund, 2003.

[29] R. Rivest and R. Schapire, "Inference of finite automata using homing sequences," in *Machine Learning: From Theory to Applications*, vol. 661 of *Lecture Notes in Computer Science*, pp. 51–73, Springer, 1993.

[30] A. Petrenko and N. Yevtushenko, "Conformance tests as checking experiments for partial nondeterministic FSM," in *Formal Approaches to Software Testing*, vol. 3997 of *Lecture Notes in Computer Science*, pp. 118–133, Springer, 2006.

[31] H. Miao, P. Liu, and J. Mei, "An improved algorithm for building the characterizing set," in *Proceedings of the 4th International Symposium on Theoretical Aspects of Software Engineering (TASE '10)*, pp. 67–74, IEEE Computer Society, Washington, DC, USA, August 2010.

[32] J. Magee and J. Kramer, *Concurrency: State Models & Java Programs*, John Wiley & Sons, New York, NY, USA, 2006.

[33] W. Pacharoen, T. Aoki, P. Bhattarakosol, and A. Surarerks, "Verifying conformance between Web service choreography and implementation using learning and model checking," in *Proceedings of the 5th International Conference on New Trends in Information Science and Service Science (NISS '11)*, pp. 375–381, IEEE Computer Society, October 2011.

[34] T. Berg, B. Jonsson, M. Leucker, and M. Saksena, "Insights to Angluin's learning," *Electronic Notes in Theoretical Computer Science*, vol. 118, pp. 3–18, 2005.