Learning Communicating State Machines

Alexandre Petrenko, Florent Avellaneda

CRIM, Canada
{Alexandre.Petrenko, Florent.Avellaneda}@crim.ca

Abstract. We consider the problems of learning and conformance testing of components in a modular system. We assume that each component can be modelled as a Finite State Machine (FSM), the topology of the system is known, but some (or all) component FSMs are unknown and have to be learned by testing the whole system, as it cannot be disassembled. Thus the classical problem of active inference of an automaton in isolation is now further lifted to a system of communicating FSMs of an arbitrary topology. As opposed to the existing work on automata learning, the proposed approach neither needs a Minimally Adequate Teacher, also called the Oracle, nor uses it a conformance tester to approximate equivalence queries. The approach further enhances a SAT solving method suggested by the authors and allows to adaptively test conformance of a system with unknown components assuming that internal communications are observable. The resulting tests are much smaller than the classical universal conformance tests derived from the composite machine of the system.

Keywords: component-based systems, communicating FSMs, active inference, FSM learning, conformance testing, adaptive testing, testing in context, SAT solving.

1 Introduction

Software industry often uses a component-based development approach to create software intense systems by selecting appropriate off-the-shelf components and assembling them with a well-defined architecture [13]. While practitioners are typically using ad hoc development techniques, model-based software engineering is investigating formal approaches which can offer automation to various phases of modular system development. In most cases, however, the components do not come with formal models, just with executable or in some cases source code. Models are nevertheless highly desired since they document the design, support test generation and model checking various properties and facilitate refactoring of a system. This explains a growing interest in automata inference. This important topic is addressed in many works, see, e.g., [14, 5, 8, 2, 6, 3], which treat a system as one black box unit, even if it contains components with known models and only some need to be learned. Most of the existing methods for query learning of an automaton model in isolation involve a Minimally Adequate Teacher, also called the Oracle, and use conformance or random tests to approximate equivalence queries [2, 3, 17, 25, 16]. Grey box inference, i.e., learning modular systems has only recently started to be investigated [1, 20]. In our previous work [20], we considered a system of two communicating FSMs, one modelling an embedded component and another - its known context and proposed a SAT solving approach to test and infer the embedded component. In fact, testing and model inference are closely related problems [4, 19], which could be solved by the SAT solving approach. In this paper, we further advance this approach to apply to a system of communicating FSMs with an arbitrary number of components and arbitrary topology.

The difference between black and grey box inference is as follows. When a given system is treated as a black box, a correctly learnt conjecture must be equivalent to a black box FSM. If however, the system is treated as a grey box, conjectures for components of a grey box are not necessarily equivalent to their respective FSMs, they are only required to compose an FSM that is equivalent to the composed FSM of the grey box. This fact has been successfully used in the area of logical design to optimize a given sequential circuit containing several components. Redesigned components must preserve the external behavior of the original circuit [25].

The interest towards modelling systems by communicating FSMs can be traced back to the eighties since an important work of Zafiropulo et al. [26]. Luo et al. [15] suggested a method for conformance testing of interacting FSMs based on determining a composite FSM that presents the external observable behavior of component FSMs. Such a machine exists assuming that the system has a single message in transit and does not fall into livelock. Thus, the system is treated as a black box, even if its topology is known.

Systems of communicating FSMs with unknown components are also considered in previous work [9, 10, 21]. The goal is to verify a given system by detecting intermittent errors. The proposed approach combines techniques for machine inference, testing and reachability analysis. Inferring a composite FSM of the system, an approximated model in the form of a Σ-quotient is obtained; the precision of the model is defined by the inference parameter Σ. Components models can then be obtained from the Σ-quotient by projections. Differently from that work, the proposed approach infers exact models and does not need to disassemble the system for unit testing.

Another body of related work addresses the so-called unknown component problem, where a basic task is to synthesize an unknown component that when combined with the known part of the system (the context) satisfies a given overall specification. This problem arises in various applications ranging from sequential synthesis to the design of discrete controllers [25]. The monograph [24] details the approach for reducing this problem to solving equations over languages and FSMs. This problem statement is in fact similar to conformance testing and learning of unknown components considered in this paper since both problem statements require a specification composite FSMs. The approach based on solving FSM equations targets the largest solution from which a minimal one can then be chosen for the unknown component, while the SAT solving approach directly determines an FSM of a minimal size. For systems with several unknown components and unknown composite FSMs, only the SAT solving approach elaborated in this paper is applicable.

The paper is organized as follows. Section 2 provides definitions related to state machines and automata. Composition, topology and composite FSM for communicating FSMs are formally defined in Section 3. A SAT solving method for simultaneous inference of communicating machines from their observed traces is presented in Section 4. Section 5 details a method for checking conformance of a system with unknown components and Section 6 describes the method for learning component FSMs in an arbitrary grey box. Section 7 concludes.

2 Definitions

A *Finite State Machine* or simple machine M is a 5-tuple (S, s_0, I, O, T), where S is a finite set of states with an initial state s_0; I and O are finite non-empty disjoint sets of inputs and outputs, respectively; T is a transition relation $T \subseteq S \times I \times O \times S$, $(s, a, o, s') \in T$ is a transition. When we need to refer to the machine being in state $s \in S$, we write M/s.

M is *complete* (completely specified) if for each tuple $(s, a) \in S \times I$ there exists transition $(s, a, o, s') \in T$, otherwise it is *partial*. M is a *trivial* FSM, denoted Φ, if $T = \varnothing$. It is *deterministic* if for each $(s, a) \in S \times I$ there exists at most one transition $(s, a, o, s') \in T$, otherwise it is *nondeterministic*. FSM M is a *submachine* of $M' = (S', s_0, I, O, T')$ if $S \subseteq S'$ and $T \subseteq T'$.

An *execution* of M/s is a finite sequence of transitions forming a path from s in the state transition diagram of M. The machine M is *initially* connected, if for any state $s \in S$ there exists an execution from s_0 to s. Henceforth, we consider only deterministic initially connected machines.

A *trace* of M/s is a string in $(IO)^*$ which labels an execution from s. Let $Tr(s)$ denote the set of all traces of M/s and Tr_M denote the set of traces of M. For trace $\omega \in Tr(s)$, we use s-after-ω to denote the state M reached after the execution of ω from s; for an empty trace ε, s-after-$\varepsilon = s$. When s is the initial state we write M-after-ω instead of s_0-after-ω.

Given a string $\omega \in (IO)^*$, the *I-restriction* of ω is a string obtained by deleting from ω all symbols that are not in I, denoted $\omega{\downarrow_I}$.

The I-restriction of a trace $\omega \in Tr(s)$ is said to be a *transfer* sequence from state s to state s-after-ω. The length of ω, denoted $|\omega|$, is defined as the length of its I-restriction. A *prefix* of trace $\omega \in Tr(s)$ is a trace $\omega' \in Tr(s)$ such that the I-restriction of the latter is a prefix of the former.

Given an input sequence α and state s, we let $out(s, \alpha)$ denote the O-restriction of the trace that has α as its I-restriction. States $s, s' \in S$ are *equivalent w.r.t.* α, denoted $s \cong_\alpha s'$, if $out(s, \alpha) = out(s', \alpha)$; they are *distinguishable* by α, denoted $s \not\cong_\alpha s'$ or simply $s \not\cong s'$, if $out(s, \alpha) \neq out(s', \alpha)$. States s and s' are *equivalent* if they are equivalent w.r.t. all input sequences, i.e., $Tr(s) = Tr(s')$, denoted $s \cong s'$. The equivalence and distinguishability relations between FSMs are similarly defined, e.g., FSMs are equivalent if their initial states are equivalent.

Given two FSMs $M = (S, s_0, I, O, T)$ and $M' = (S', s'_0, I, O, T')$, their *product* $M \times M'$ is the FSM (P, p_0, I, O, H), where $p_0 = (s_0, s'_0)$ such that P and H are the smallest sets satisfying the following rule: If $(s, s') \in P$, $(s, x, o, t) \in T$, $(s', x, o, t') \in T'$, then $(t, t') \in P$ and $((s, s'), x, o, (t, t')) \in H$. It is known that if M and M' are complete machines then they are equivalent if and only if the product $M \times M'$ is complete.

We also use the classical automaton model. A *Finite Automaton* A is a 5-tuple (P, p_0, X, T, F), where P is a finite set of states with the initial state p_0; X is a finite alphabet; T is a transition relation $T \subseteq S \times X \cup \{\varepsilon\} \times S$, where ε represents an internal action, and F is a set of *final* or *accepting* states, defining the language of A, denoted $L(A)$. We shall use several operations over automata, namely, expansion, restriction, and intersection, following [24].

Given an automaton $A = (P, p_0, X, T, F)$, and a finite alphabet U, the *U-expansion* of automaton A is the automaton, denoted $A{\uparrow_U}$, obtained by adding at each state a self-loop transition labeled with each action in $U \setminus X$.

For an automaton A and an alphabet U, the *U-restriction* of automaton A is the automaton, denoted $A{\downarrow_U}$, obtained by replacing each transition with the symbol in $X \setminus U$ by an ε-transition between the same states.

Given automata $A = (P, p_0, X, T, F_A)$ and $B = (R, r_0, Y, Z, F_B)$, such that $X \cap Y \neq \varnothing$, the *intersection* $A \cap B$ is the largest initially connected submachine of the automaton $(P \times R, (p_0, r_0), X \cap Y, Q, F_A \times F_B)$, where for each symbol $a \in X \cap Y$ and each state $(p, r) \in P \times R$, $((p, r), a, (p', r')) \in Q$, if $(p, a, p') \in T$ and $(r, a, r') \in Z$. The intersection operation is associative, hence it applies to more than two automata.

We also define an automaton corresponding to a given FSM M. The automaton, denoted by $A(M)$, is obtained by splitting each transition of M labeled by input/output into two transitions labeled by input and output, respectively, and connecting them with an auxiliary non-final state. The original states of M are only final states of $A(M)$, hence the language of $A(M)$ coincides with the set of traces of M.

3 FSM Composition

We consider a system of communicating FSMs defined as follows. Let $M_1, \ldots, M_k$ be a set of component FSMs in the system, where $M_i = (S_i, s_{i0}, X_i \cup V_i, O_i \cup U_i, T_i)$, is a complete deterministic machine. The input alphabets are partitioned into external X_i and internal V_i inputs. The output alphabets are also partitioned into external O_i and internal U_i outputs. The sets of states, inputs and outputs are assumed to be pairwise disjoint, except for pairs of sets of inputs V_i and outputs U_j, $j \neq i$. These sets define the topology of the given system, namely, if $V_i \cap U_j \neq \varnothing$, then the output of M_j is connected to the input of M_i.

Formally, the *topology* of the given system is the set$\{(V_i, U_j) \mid V_i \cap U_j \neq \varnothing, i, j \in \{1, \ldots, k\}\}$, denoted $T(M_1, \ldots, M_k)$. We define a well-defined topology by excluding insufficiently connected machines.

Then the topology of the given communicating FSMs is *well-defined*, if for each $i = 1, \ldots, k$, $V_i = \varnothing$ implies $U_i \neq \varnothing$, $U_i = \varnothing$ implies $V_i \neq \varnothing$, and $\cup_i^k V_i = \cup_i^k U_i$. Intuitively, a component without internal inputs (outputs) must have internal outputs (inputs), and all internal inputs as well as outputs must be corresponding outputs and inputs, respectively. Since simple removal of isolated machines could make topologies of the resulting systems well-defined, we assume henceforth well-definedness for granted. In this paper, we also assume that all communications between machines are unicast, so multicast is not used. Formally, this constrains the topology of a given system by requiring that all the sets of internal inputs are pairwise disjoint.

In the following, we shall use $X = \cup^k_{i=1} X_i$ for the set of external inputs, $O = \cup^k_{i=1} O_i$ for the set of external outputs and $I = \cup^k_{i=1} V_i = \cup^k_{i=1} U_i$ for the set of internal actions.

The behavior of a system of communicating FSMs is controlled by its environment which submits external inputs and receives external outputs. If the environment is allowed to submit inputs before it receives an external output, the system may need to buffer actions using queues. Then their size is defined by the number of consecutive inputs preceding the output caused by the first input. It is usual in testing to consider a so-called slow environment that ensures that there is only a single message in transit [15]. A *slow* environment can be modelled as a "chaos" automaton $Env = (\{p_0, p_1\}, p_0, X \cup O, T, \{p_0\})$, where $T = \{(p_0, x, p_1) \mid x \in X\} \cup \{(p_1, o, p_0) \mid o \in O\}$. After issuing an external input to the system it enters the non-initial state p_1 and returns to the final state when an external output is produced by the system where it issues a next input. Its language is the set $(XO)^*$.

In a system that has only a single message in transit, an internal output of one machine is immediately consumed only by one machine as its input, the communications are thus performed in fact by rendezvous. This allows to define an FSM composition operator using the intersection of their corresponding automata which has all the possible executions of the system with the slow environment.

Given $M_1, \ldots, M_k$, where $M_i = (S_i, s_{i0}, X_i \cup V_i, O_i \cup U_i, T_i)$, let $A(M_1), \ldots, A(M_k)$ be automata corresponding to the given FSMs. The *composite automaton*, denoted $A(M_1, \ldots, M_k)$, is the intersection $\cap^k_{i=1} A(M_i)\uparrow_I \cap Env\uparrow_I$. The language of $A(M_1, \ldots, M_k)$ is the set of all accepted words labelling all the executions of the closed system of communicating FSMs with the slow environment. The external behavior of the system is expressed in terms of external inputs X and outputs O, so it is the set of $(X \cup O)$-restrictions of accepted words of $A(M_1, \ldots, M_k)$, i.e., the set of external traces of the system. They are traces of an FSM that could be obtained by removing ε-transitions in $A(M_1, \ldots, M_k)_{\downarrow X \cup O}$ and pairing each input with a subsequent output, if it exists, to an FSM transition's label. Final states of $A(M_1, \ldots, M_k)$ become states of the FSM. If some external input is not followed by an external output it is deleted from the corresponding final state of $A(M_1, \ldots, M_k)_{\downarrow X \cup O}$, making the FSM partial. Thus, a complete FSM can be obtained only if the automaton $A(C)$ has no livelocks, i.e., cycles labelled by internal actions in I [24]. We let $C(M_1, \ldots, M_k)$ denote the resulting complete machine, called the *composite FSM* of the system.

Example. Consider two communicating FSMs M_1 and M_2 shown in Fig. 1 (a) and (b), respectively [20]. The composite FSM $C(M_1, M_2)$ is in Fig. 1 (c). The composite automaton $A(M_1, M_2)$ is shown in Fig. 2.

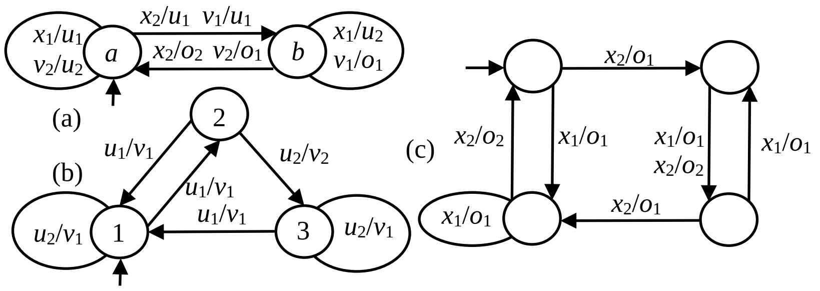

Fig. 1. The FSM M_1 (a), FSM M_2 (b) and composite FSM $C(M_1, M_2)$ (c).

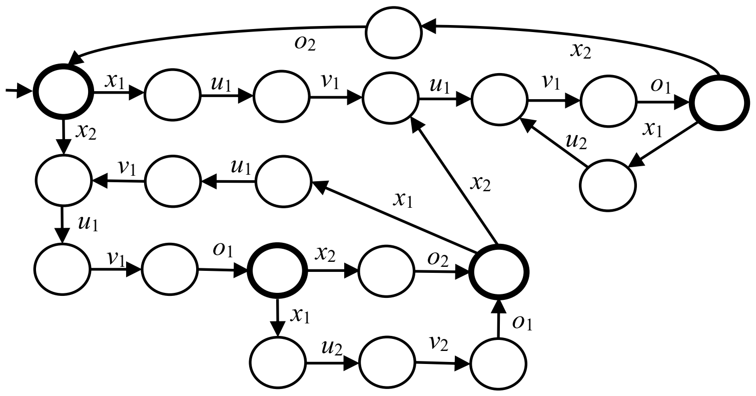

Fig. 2. The composite automaton $A(M_1, M_2)$, final states are in bold.

4 Inference from Traces

In this section, we address the following inference problem. Given a system of k communicating FSMs with a known topology $T(M_1, \ldots, M_k)$, such that some component FSMs are unknown, assume that we are also given a set of traces produced by each component when some sequences of external inputs are applied to the system. We want to infer for each unknown component a conjecture consistent with the observed traces.

Given a string $\omega \in (IO)^*$, let $Pref(\omega)$ be the set of all prefixes of ω. We define a (linear) FSM $W(\omega) = (X, x_0, I, O, D_\omega)$, where D_ω is a transition relation, such that $|X| = |\omega| + 1$, and there exists a bijection f: $X \rightarrow Pref(\omega)$, such that $f(x_0) = \varepsilon$, $(x_i, a, o, x_{i+1}) \in D_\omega$ if $f(x_i)ao = f(x_{i+1})$ for all $i = 0, \ldots, |\omega| - 1$, in other words, $W(\omega)$ has the set of traces $Pref(\omega)$. We call it the ω-*machine*. Similarly, given a finite prefix-closed set of traces $\Omega \subset (IO)^*$ of some deterministic FSM, let $W(\Omega) = (X, x_0, I, O, D_\Omega)$ be the acyclic deterministic FSM such that Ω is the set of its traces, called an Ω-*machine*. The bijection f relates states of this machine to traces in Ω.

While the set of traces of the Ω-machine is Ω, there are many FSMs which contain the set Ω among their traces. An FSM $C = (S, s_0, I, O, T)$ is called an Ω-*conjecture*, if $\Omega \subseteq Tr_C$.

The states of the Ω-machine $W(\Omega) = (X, x_0, I, O, D_\Omega)$ and an Ω-conjecture $C = (S, s_0, I, O, T)$ are closely related to each other. Formally, there exists a mapping μ: $X \rightarrow S$, such that $\mu(x) = s_0$-after-$f(x)$, the state reached by C with the trace $f(x) \in \Omega$. The mapping μ is unique and induces a partition π_C on the set X such that x and x' belong to the same block of the partition π_C, denoted $x =_{\pi_C} x'$, if $\mu(x) = \mu(x')$.

Given an Ω-conjecture C with the partition π_C, let D be an Ω'-conjecture with the partition π_D, such that $\Omega' \subseteq \Omega$, we say that the partition π_C is an *expansion* of the partition π_D, if its projection onto states of Ω' coincides with the partition π_D.

We now lift the above notions to a system of communicating FSMs.

Given k FSMs, let $\Omega_1, \ldots, \Omega_k$ be the sets of their observed traces and $W(\Omega_i) = (Z_i, z_{i0}, X_i \cup V_i, O_i \cup U_i, D_{\Omega i})$, $i = 1, \ldots, k$ be the Ω-machines. We want to determine for each $W(\Omega_i)$ a conjecture $N_i = (S_i, s_0, X_i \cup V_i, O_i \cup U_i, T_i)$ with at most n_i states, i.e., $|S_i| \leq n_i$. Each state of the Ω-conjectures is represented by a variable s, that belongs to one of the sets $S_1 = \{1, \ldots, n_1\}$, $S_2 = \{n_1+1, \ldots, n_1+n_2\}$, $\ldots$, $S_k = \{\Sigma^{k-1}_{i=1}n_i+1, \ldots, \Sigma^{k}_{i=1}n_i\}$. To simplify further formulas, we use m_i to denote $\Sigma^{i}_{j=1}n_j$ so that $S_i = \{m_{i-1}+1, \ldots, m_i\}$. Thus we need to find k mappings μ_i: $Z_i \rightarrow S_i$, $i = 1, \ldots, k$ satisfying the following constraints:

$$\begin{gathered}\forall z, z' \in Z_i\text{: if } z \not\cong z' \text{ then } \mu_i(z) \neq \mu_i(z') \text{ and} \\ \text{if } \exists a \in X_i \cup V_i \text{ s.t. } out(z, a) = out(z', a) = o, \text{ where } o \in O_i \cup U_i, \text{ then} \qquad (1) \\ \mu_i(z) = \mu_i(z') \Rightarrow \mu_i(z)\text{-after-}ao = \mu_i(z')\text{-after-}ao\end{gathered}$$

A mapping μ_i satisfying (1) defines a partition on Z_i and each block becomes a state of the Ω_i-conjecture, we use $\pi(N_i)$ to denote the partition. All the mappings, if exist, define a global partition $\pi(N_1, \ldots, N_k)$ on $\cup^k_{i=1} Z_i$, the states of all k conjectures.

Inspired by [12], we translate these formulas to SAT using unary coding for integer variables, represented by m_k Boolean variables $v_{z,1}, \ldots, v_{z,m_k}$, where $m_k = \Sigma^k_{i=1} n_i$.

For each $z \in Z_i$ and all $i = 1, \ldots, k$ we have the clauses:

$$v_{z,m_{i-1}+1} \vee \ldots \vee v_{z,m_i} \qquad (2)$$

They mean that each state of each Ω-machines must be in at least one block.

For each $z \in Z_i$, all $i = 1, \ldots, k$, and all $p, q \in \{m_{i-1}+1, \ldots, m_i\}$ such that $p \neq q$, we have the clauses:

$$\neg v_{z,p} \vee \neg v_{z,q} \qquad (3)$$

They mean that each state of each Ω-machines must be in at most one block.

We use auxiliary variables $e_{z,z'}$. For each $z \in Z_i$ and each $z' \in Z_j$ such that $i \neq j$

$$\neg e_{z,z'} \qquad (4)$$

For every $z, z' \in Z_i$ such that $z \not\cong z'$ and all $i = 1, \ldots, k$, we have

$$\neg e_{z,z'} \qquad (5)$$

For every $z, z' \in Z_i$ such that $out(z, a) = out(z', a) = o$ for some $o \in O_i \cup U_i$ and all $i = 1, \ldots, k$, we have

$$e_{z,z'} \Rightarrow e_{z\text{-after-}ao,z'\text{-after-}ao} \; (6)$$

For every $z, z' \in Z_i$, all $i = 1, \ldots, k$, and all $p \in \{m_{i-1}+1, \ldots, m_i\}$

$$e_{z,z'} \wedge v_{z,p} \Rightarrow v_{z',p} \qquad (7)$$

$$\neg e_{z,z'} \wedge v_{z,p} \Rightarrow \neg v_{z',p} \qquad (8)$$

The resulting Boolean formula is the conjunction of clauses (2) - (8). If it is satisfiable then a solution is a set of conjectures for all unknown components. A solution might not necessarily be unique, hence to solve the problems of conformance testing and learning we need to make sure that the found set of conjectures is unique or to determine another set of non-equivalent conjectures.

This is achieved by using the following procedure inferring all conjectures for the unknown components at once. We let Π denote a set of global partitions each of which defines k conjectures already inferred from the previously observed traces. To ensure generation of different conjectures, partitions are used to formulate additional constraints. The procedure generates a set of conjectures, such that the number of states of each conjecture does not exceed a given upper bound, if they exist. The conjectures are verified for livelock by composing them (with the known FSMs, if any) into a composite automaton, as explained in the previous section. If it has a livelock then the procedure tries to find another set of conjectures and uses the global partition induced by the conjectures to avoid repeated regeneration in further iterations. The procedure is formalized in the following algorithm.

Algorithm 1. *Infer_conjectures*($\{\Omega_1, \ldots, \Omega_k\},\{M_1, \ldots, M_m\},\{n_1, \ldots, n_k\}, \Pi$)

Input: Sets of FSM traces $\Omega_1, \ldots, \Omega_k$, a set of known FSMs $M_1, \ldots, M_m$, a set of integers $n_1, \ldots, n_k$, and a set of global partitions Π

Output: A set of k conjectures and an updated set of partitions or False.

```
1. formula = conjunction of the clauses (2) - (8)
2. loop do
3.   for all $\pi \in \Pi$ do
4.     clause = False
5.     for all $z, z'$ such that $z =_\pi z'$ do
6.       clause = clause $\vee \neg e_{z,z'}$
7.     end for
8.     formula = formula $\wedge$ clause
9.   end for
10.  if formula is not satisfiable then
11.    return False
12.  end if
13.  $\{N_1, \ldots, N_k\}$ := call-solver(formula)
14.  if $A(N_1, \ldots, N_k, M_1, \ldots, M_m)$ has no livelock then
15.    return $\{N_1, \ldots, N_k\}$, $\Pi$
16.  end if
17.  $\Pi := \Pi \cup \pi(N_1, \ldots, N_k)$
18. end loop
```

To check the satisfiability of a formula one can use any of the existing solvers, calling the function *call-solver*(*formula*).

5 Checking Conformance with Unknown Components

In this section, we consider the following conformance testing problem. Given a system of communicating FSMs, such that some component FSMs are unknown, assume that we are also given an FSM that describes the expected external behavior of the system, called a specification composite FSM. We assume that the system has no livelocks, so its external behavior can be represented by a complete composite FSM. We need to determine whether the composite FSM of the system conforms (is equivalent) to the specification or find a counterexample, i.e., an external test that distinguishes them, otherwise. Moreover, if the system conforms to the specification then we want to learn all its unknown component FSMs. We assume that all the internal interfaces are observable, but only external inputs are controllable, so the system is a grey box with a single message in transit. The problem reflects a practical situation when in a modular system some components are replaced by their updated versions and one needs to test whether the external behavior is not changed.

The proposed method for checking conformance and learning component FSMs verifies whether the current conjectures obtained from already observed traces when composed together with known component FSMs behave as the given specification FSM. If they do not then the product of the specification and composite FSMs is used to determine a sequence of external inputs that distinguishes them. It is applied to the grey box obeying the property of a slow environment *Env*, so observed traces are inputs always interleaved with outputs. The observed traces extend the set of traces of unknown components unless the grey box does not produce the output sequence of the specification. In the last case, the external input sequence is returned as a counterexample, the current conjectures are also reported as a diagnostics of the observed non-conformance of the grey box. The process iterates as long as the current conjectures form a conforming system. The method calls Algorithm 1 that builds conjectures, checks whether they are unique and returns them, if it is the case, terminating the process. Algorithm 1 calls in turn a SAT solver constraining it to avoid solutions of already considered conjectures. The solver may not find any solution when the assumed bounds on the state numbers are insufficiently low. In this case, the algorithm needs to be executed with increased bounds. The procedure is implemented in Algorithm 2.

Let *GB* denote the system of the component FSMs $M_1, \ldots, M_{k'}, \ldots, M_k$, such that the first k', $0 < k' \leq k$ components are unknown, and $n_1, \ldots, n_{k'}$ are the bounds on the number of their states, respectively. We let *M* denote a complete FSM over the same external inputs and outputs as *GB*, called the specification FSM.

Algorithm 2. Checking conformance and learning components

Input: A *GB* with known components $M_{k'+1}, \ldots, M_k$ and a specification FSM *M*.

Output: Unknown component FSMs or a test that distinguishes the composite FSM of *GB* from *M*.

```
 1. $\Omega_i := \emptyset$, $i = 1, \ldots, k$
 2. $\Pi := \emptyset$
 3. while conjectures $N_1, \ldots, N_{k'}$ and $\Pi$ are returned by Infer_conjectures($\{\Omega_1, \ldots,$
    $\Omega_{k'}\}, \{M_{k'+1}, \ldots, M_k\}, \{n_1, \ldots, n_{k'}\}, \Pi$) do
 4.   if $C(M_1, \ldots, M_k) \times C(N_1, \ldots, N_{k'}, \ldots, M_k)$ is complete then
 5.     return $N_1, \ldots, N_{k'}$
 6.   end if
 7.   Let $\beta a$ be an external input sequence such that $\beta$ is the shortest transfer sequence
      to a state with the undefined input $a$ in $C(M_1, \ldots, M_k) \times C(N_1, \ldots, N_{k'}, \ldots, M_k)$
 8.   Let $\sigma$ be the external trace and $\sigma_1, \ldots, \sigma_{k'}$ be unknown components' traces
      observed when the input sequence $\beta a$ is applied to GB
 9.   If $\sigma$ is not the trace of $C(M_1, \ldots, M_k)$ then
10.     return "the test $\beta a$ distinguishes GB from M and the conjectures $N_1, \ldots, N_{k'}$"
```

```
11.   end if
12.   $\Omega_i := \Omega_i \cup \{\sigma_i\}$, $i = 1, \ldots, k'$
13. end while
14. return "the bounds $n_1, \ldots, n_{k'}$ are too low"
```

Note that the Boolean formula used by the SAT solver is built incrementally; a current formula is saved and new clauses are added when any set Ω_i or Π is augmented.

Theorem 1. Algorithm 2 learns unknown components of a conforming grey box or returns a counterexample test, otherwise.

Sketch of Proof. If in line 5, $C(M_1, \ldots, M_k) \times C(N_1, \ldots, N_{k'}, M_{k'+1}, \ldots, M_k)$ is complete, then the grey box is equivalent to the specification and $N_1, \ldots, N_{k'}$ is a solution for unknown components. If in line 9, σ is not a trace of $C(M_1, \ldots, M_k)$, then since σ is an external trace, the test βa as its input sequence distinguishes *GB* from the specification *M*. At each iteration in the while loop, a trace is added to at least one set Ω_i. Thus, at least one potential solution $N_1, \ldots, N_{k'}$ is eliminated among the possible solutions. Since the number of states in components is fixed, the number of potential solutions is bounded. Thus, the loop will end when all potential solutions are eliminated. ■

The algorithm was implemented in C++ with MiniSat solver [7] and we use this prototype for experiments in a VirtualBox with 8 GB of RAM and i5-7500 processor.

Step	External trace	Added variables	Added clauses	Time μsec
1	ε	5	6	13
2	x_1o_1	30	63	17
3	x_2o_1	22	42	10
4	$x_1o_1x_2o_2$	14	29	8
5	$x_1o_1x_1o_1$	43	90	15
6	$x_1o_1x_1o_1x_1o_1$	53	115	415
7	$x_2o_1x_1o_1x_2o_1$	136	300	48
8	$x_1o_1x_2o_2x_2o_1x_1o_1$	176	391	158
9	$x_1o_1x_2o_2x_1o_1x_1o_1x_2o_2$	314	701	109
10	$x_1o_1x_1o_1x_2o_2x_1o_1$	272	628	89
11	$x_1o_1x_1o_1x_2o_2x_1o_1x_1o_1x_2o_2$	215	479	76
12	$x_1o_1x_1o_1x_1o_1x_1o_1x_2o_2$	235	525	226
13	$x_2o_1x_1o_1x_1o_1x_1o_1x_2o_1$	651	1481	199
14	$x_1o_1x_1o_1x_2o_2x_2o_1x_1o_1x_2o_1$	626	1411	173
Total		2792	6261	1156

Table 1. Testing conformance and learning component FSMs.

Example. We consider the communicating FSMs shown in Fig. 1. Assuming that both FSMs are unknown, but their composite FSM shown in Fig. 1 (c) is known, we use the prototype tool to learn the component FSMs. Let the bounds on the number of states in the components be two for M_1 and three for M_2.

Algorithm 2 executes 14 cycles in about one millisecond and terminates. As expected, no test distinguishing the system from the specification FSM is found. Fig. 3 shows the resulting conjectures N_1 and N_2. The conjecture N_2 is a partial machine, the first two states can be merged to obtain a minimal FSM with two states. Their composite FSM is equivalent to the specification FSM in Fig. 2. One can notice that the learnt conjectures are simpler that the original FSMs; in fact, they have fewer transitions and the minimal form of the second conjecture has not three, but two states. Table 1 provides a summary of execution details for all cycles of Algorithm 2, namely, the number of variables and clauses added in each step and time in microseconds. It is interesting to notice in Table 1 that time required to solve a SAT instance does not grow linearly with the number of variables and clauses. In our incremental approach, newly added clauses often just speed up the process of finding a solution. In the context of automata learning, similar observations were also previously reported [19]. This indicates that the number of variables and clauses cannot be directly used to characterize the complexity of the SAT solving approach to the FSM inference problem. It is intuitively clear that the more component FSMs are unknown the higher the complexity of the learning and testing problems. In the running example, we consider that both component FSMs are unknown, if, however, we assume that only M_2 is unknown then conformance testing and learning a single unknown component FSM requires fewer tests.

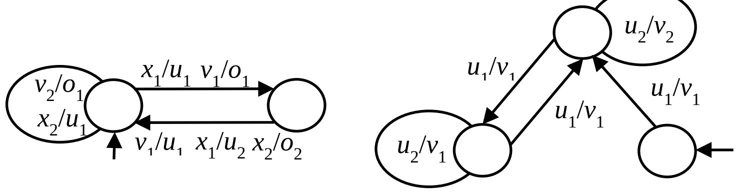

Fig. 3. The resulting conjectures N_1 and N_2.

To demonstrate how Algorithm 2 executes when given a nonconforming grey box, we assume that the second component FSM is like M_2 in Fig. 2 (b), except for a transition from state 3 to state 1 that has the label u_1/v_2 instead of u_1/v_1. The first seven executed steps are exactly the steps shown in Table 1. Then three more steps are performed to determine the sequence of external inputs $x_2x_1x_2x_2$ that distinguishes the composed FSM of the mutated system from the specification FSM. The two produce different external traces $x_2o_1x_1o_1x_2o_1x_2o_1$ and $x_2o_1x_1o_1x_2o_1x_2o_2$.

We now compare the cost of checking conformance of the grey box system with observable internal communications to that of the black box system without such observations. As Table 1 indicates, to test conformance of the given grey box 13 tests suffice and overall 60 test actions (47 external inputs and 13 resets) need to be applied to the system. When internal interactions cannot be observed, the whole system becomes a black box. To test communicating FSMs we use universal conformance tests which could be derived from their composite FSM. In our example, the composite FSM is shown in Fig. 1 (c). The conformance tests should be constructed assuming that the number of states in the system can reach six, since the bounds for the components' state number are two and three, respectively. Considering the set $\{x_2x_1x_2, x_1x_2\}$ as a characterization set W of the composite FSM, the W-method [23] generates 40 input sequences of the total length of 276. We conclude that the proposed approach for testing conformance of communicating FSMs as a grey box offers an important save in testing efforts. Testing a system as a grey box is adaptive, since test actions are determined based on the observations and compared to the execution of universal conformance tests against the system treated as a black box it could not be less effective. To the best of our knowledge, it is the first method for adaptive conformance testing of communicating FSMs of an arbitrary topology.

6 Learning a Grey Box

In this section, we consider the problem of inferring communicating FSMs. This is a generalization of the classical FSM inference problem [11, 18, 22] to a system of FSMs. As in the previous section, we are given a system of communicating FSMs with a known topology and external input alphabet. It is assumed that all the internal interfaces are observable, but only external inputs are controllable, so the system is a grey box with a single message in transit. We also assume that the grey box has no livelock, so its external behavior can be represented by an FSM.

Differently from the conformance testing problem in Section 5, we now know neither specification composite FSM nor any component FSM. We need to learn all component FSMs at once. Yet, as before, we fix the upper bounds on the number of states of each component. As discussed above, the "right" bounds could be determined by iterative execution of our learning method with increasing bounds.

We let *GB* denote a system of unknown FSMs $M_1, \ldots, M_k$ and $n_1, \ldots, n_k$ be the bounds on the number of their states, respectively. The learning procedure is implemented in Algorithm 3. It is an enhancement of Algorithm 2 replacing the specification FSM M by the composite FSM of the current conjectures.

Algorithm 3. Learning a grey box
Input A grey box *GB* with a known external alphabet and integers $n_1, \ldots, n_k$
Output Conjectures for all k components

```
1.  $\Omega_i := \varnothing$, $i = 1, \ldots, k$
2.  $\Pi := \varnothing$
```

```
3.  $N_i = \Phi$ (trivial FSM), $i = 1, \ldots, k$
4.  **while** conjectures $D_1, \ldots, D_k$ and $\Pi$ are returned by *Infer_conjectures*($\{\Omega_1, \ldots, \Omega_k\}, \{\},\{n_1, \ldots, n_k\}, \Pi$) **do**
5.    **if** $C(D_1, \ldots, D_k) \times C(N_1, \ldots, N_k)$ is complete **then**
6.      $\Pi := \Pi \cup \pi(D_1, \ldots, D_k)$
7.    **else**
8.      Let $\beta a$ be an input sequence such that $\beta$ is the shortest transfer sequence to a state with the undefined input $a$ in $C(D_1, \ldots, D_k) \times C(N_1, \ldots, N_k)$
9.      Let $\sigma$ be the external trace and $\sigma_1, \ldots, \sigma_k$ be components' traces observed when the input sequence $\beta a$ is applied to $GB$
10.     $\Omega_i := \Omega_i \cup \{\sigma_i\}, i = 1, \ldots, k$
11.     **if** $\sigma$ is not a trace of $C(N_1, \ldots, N_k)$ **then**
12.       $\{N_1, \ldots, N_k\}, \Pi :=$ *Infer_conjectures*($\{\Omega_1, \ldots, \Omega_k\}, \{\},\{n_1, \ldots, n_k\}, \Pi$)
13.     **end if**
14.   **end if**
15. **end while**
16. **return** $N_1, \ldots, N_k$
```

The algorithm returns the conjectures as a main result, but also it determines sets of observed traces of each component used to infer them. The uniqueness of the conjectures is that they form a composite FSM of the given grey box, as stated in the following.

Theorem 2. Algorithm 3 returns the conjectures $N_1, \ldots, N_k$ such that $C(N_1, \ldots, N_k) \cong C(M_1, \ldots, M_k)$ and for each grey box with components $L_1, \ldots, L_k$ such that L_i has at most n_i states, $Tr_{Li} \supseteq \Omega_i$ for each L_i and the topology $T(M_1, \ldots, M_k)$ we have $C(L_1, \ldots, L_k) \cong C(M_1, \ldots, M_k)$.

Proof. When Algorithm 3 terminates, the result $N_1, \ldots, N_k$ is such that $C(N_1, \ldots, N_k) \cong C(M_1, \ldots, M_k)$ because $N_1, \ldots, N_k$ are FSMs consistent with traces $\{\Omega_1, \ldots, \Omega_k\}$, and all other solutions $L_1, \ldots, L_k$ are such that $C(L_1, \ldots, L_k) \cong C(N_1, \ldots, N_k)$ or have livelock. Now we prove by contradiction that if there exist $L_1, \ldots, L_k$ such that for each L_i, $Tr_{Li} \supseteq \Omega_i$ and the number of states does not exceed n_i, then $C(L_1, \ldots, L_k) \cong C(M_1, \ldots, M_k)$. Assume that there exist $L_1, \ldots, L_k$ such that $C(L_1, \ldots, L_k) \ncong C(M_1, \ldots, M_k)$. Because the formula used in *Infer_conjectures*($\{\Omega_1, \ldots, \Omega_k\}, \{\},\{n_1, \ldots, n_k\}, \Pi$) is not satisfiable when Algorithm 3 returns $N_1, \ldots, N_k$, the global partition induced by $L_1, \ldots, L_k$ is in Π. This would mean that either $C(L_1, \ldots, L_k) \cong C(N_1, \ldots, N_k)$ or there exists σ in Ω such that σ is not a trace of $C(L_1, \ldots, L_k)$. However, these two cases are not possible. ■

The algorithm was also implemented in C++ with MiniSat solver and we use this prototype for experiments as in Section 5.

Example. We consider our running example of the FSMs shown in Fig. 1. Assume we know only the external alphabet and the bounds on the number of states in the components, two for M_1 and three for M_2.

Algorithm 3 executes 42 cycles in about ten milliseconds and terminates returning the same conjectures as Algorithm 2, see Fig. 3. 24 cycles out of 42 do not add any new traces, they just update the set of global partitions by adding a single clause in line 7 of Algorithm 3. Table 2 provides the details for the remaining 18 steps; the missing numbers belong to 24 omitted steps. The observations made in Section 5 about relations between time and the number of variables and clauses in Table 1 are also valid for Table 2.

As Table 2 indicates to learn both components of the given grey box 17 tests suffice and overall 76 test actions (59 external inputs and 17 resets) need to be applied to the system. Comparing this to the scenario of learning components with a given specification composite FSM where 13 tests with 60 test actions are used, we can conclude that the absence of an oracle played by the specification FSM complicates the inference problem. Notice that if in this example, the FSM M_1 is known and only M_2 needs to be learnt then it is sufficient to use only 5 tests with 19 test actions (14 external inputs and 5 resets).

Step	External trace	Added variables	Added clauses	Time µsec
1	ε	5	6	22
2	x_1o_1	30	63	59
3	x_2o_1	22	42	46
5	$x_1o_1x_1o_1$	39	84	47
6	$x_1o_1x_2o_2$	18	35	36
7	$x_1o_1x_1o_1x_1o_1$	53	115	63
8	$x_1o_1x_1o_1x_1o_1x_1o_1$	63	140	107
9	$x_1o_1x_2o_2x_1o_1$	122	284	123
13	$x_1o_1x_1o_1x_1o_1x_2o_2x_1o_1$	188	436	199
18	$x_1o_1x_1o_1x_2o_2x_1o_1$	228	535	196
22	$x_2o_1x_1o_1$	129	293	206
23	$x_2o_1x_1o_1x_1o_1$	228	550	203
24	$x_2o_1x_1o_1x_2o_1$	122	263	188
27	$x_2o_1x_1o_1x_2o_1x_1o_1x_2o_2$	231	525	294
28	$x_1o_1x_2o_2x_2o_1x_1o_1x_2o_1$	522	1177	336
30	$x_1o_1x_2o_2x_1o_1x_1o_1x_2o_2$	291	653	318
36	$x_1o_1x_1o_1x_2o_2x_2o_1x_1o_1x_2o_1$	642	1450	643
40	$x_2o_1x_1o_1x_1o_1x_1o_1x_2o_1$	504	1150	620

Total	3437	7825	9846

Table 2. Learning the grey box.

7 Conclusions

We considered the problems of learning components and conformance testing of a modular system. The system is modelled by FSMs communicating by message passing with a single message in transit. Communications between machines defines the topology of the system. The composite FSM represents the external behavior of the system. Formulating the learning problem we assume that some or all component FSMs are unknown and have to be learned by testing the whole system, as it cannot be disassembled. The system is then tested as a grey box, since internal actions are observed when external tests are executed. Thus the classical problem of active inference of an automaton in isolation is now further lifted to a system of communicating FSMs of an arbitrary topology. To the best of our knowledge, no method was proposed to solve this problem yet. Compared to the existing work on automata learning, the proposed approach neither needs a Minimally Adequate Teacher (Oracle), nor uses it a conformance tester to approximate equivalence queries.

The problem of conformance testing of communicating FSMs with unknown components is quite similar to the above problem. Since checking conformance of a given system requires a specification FSM, a composite FSM is used as an oracle for conformance testing. The proposed approach allows to adaptively test conformance of a system with unknown components. The resulting tests are much smaller that the classical universal conformance tests derived from the composite FSM of the system. Moreover, unknown components are also learned once the system is found to be conformant.

It is worth to notice that while we assumed that all the internal interactions can be observed, the approach works even when a given system is partially observable. Any part of a system with fully observable inputs and outputs can be learnt as a single FSM that is a composite FSM of all components of the subsystem.

As a future work it could be interesting to relax some assumptions used in the proposed approach, e.g., determinism and absence of queues and to investigate learning of systems which use communications other than message passing.

Acknowledgements. This work was partially supported by MEI (Ministère de l'Économie et Innovation) of Gouvernement du Québec and NSERC of Canada.

References

1. Abel A, Reineke J. Gray-box learning of serial compositions of Mealy machines. NASA Formal Methods Symposium, 272-287 (2016)

2. Angluin D., Learning regular sets from queries and counterexamples. Information and computation, 75(2), 87–106 (1987)
3. Bennaceur, A., Giannakopoulou, D., Hähnle, R., Meinke, K. Machine learning for dynamic software analysis: Potentials and limits, Springer, LNCS 11026, (2018)
4. Berg, T. et al. On the correspondence between conformance testing and regular inference, in Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering, LNCS 3442, 175-189, (2005).
5. Biermann, A. W., Feldman, J. A. On the synthesis of finite-state machines from samples of their behavior. IEEE Transactions on Computers, 100(6), 592-597 (1972)
6. De la Higuera, C. Grammatical inference: learning automata and grammars. Cambridge University Press (2010)
7. Eén, N., Sörensson, N. An extensible SAT-solver. In International conference on theory and applications of satisfiability testing, Springer, Berlin Heidelberg, 502-518 (2003)
8. Gold, E. M. Complexity of automaton identification from given data. Information and control, 37(3), 302-320 (1978)
9. Groz, R., Li, K., Petrenko, A., Shahbaz, M. Modular system verification by inference, testing and reachability analysis. In Testing of Software and Communicating Systems, Springer, Berlin Heidelberg, 216-233 (2008)
10. Groz, R., Li, K., Petrenko, A. Integration testing of communicating systems with unknown components. Annals of telecommunications, 70 (3-4), 107-125 (2015)
11. Groz, R., Simao, A., Petrenko, A. Oriat, C. Inferring FSM models of systems without reset. In Machine Learning for Dynamic Software Analysis: Potentials and Limits. LNCS 11026, Springer, Cham., 178-201, (2018)
12. Heule, M.J., Verwer, S. Exact DFA identification using SAT solvers. In International Colloquium on Grammatical Inference. Springer Berlin Heidelberg, 66-79 (2010)
13. Jaffar-ur Rehman, M., Jabeen, F., Bertolino, A., Polini, A. Testing software components for integration: a survey of issues and techniques. Softw. Test. Verif. Reliab., 17, 95–133 (2007)
14. Kella, J. Sequential machine identification. IEEE Transactions on Computers, 100(3), 332-338 (1971)
15. Luo, G., von Bochmann, G., Petrenko, A. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. IEEE Transactions on Software Engineering, 20(2), 149-162, (1994)
16. Meinke, K. CGE: A sequential learning algorithm for Mealy automata. In ICGI, 148-162 (2010)
17. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. Journal of Automata, Languages and Combinatorics, 7(2), 225–246 (2001)
18. Petrenko, A., Avellaneda, F., Groz, R., Oriat, C. From passive to active FSM inference via checking sequence construction. In IFIP International Conference on Testing Software and Systems. Springer, 126-141 (2017)
19. Petrenko, A., Avellaneda, F., Groz, R. Oriat, C. FSM inference and checking sequence construction are two sides of the same coin, Software Quality Journal, 2018, https://doi.org/10.1007/s11219-018-9429-3
20. Petrenko, A., Avellaneda, F. Inference and conformance testing of embedded components, In IFIP International Conference on Testing Software and Systems. Springer, LNCS 11146, 119-134, (2018)

21. Shahbaz, M., Shashidhar, K.C. and Eschbach, R., Iterative refinement of specification for component based embedded systems. In Proceedings of the 2011 International Symposium on Software Testing and Analysis, 276-286, (2011)
22. Steffen, B. et al. Active Automata Learning: From DFAs to interface programs and beyond, ICGI, 195–209 (2012)
23. Vasilevski, M. P. Failure diagnosis of automata. Cybernetics, Plenum Publishing Corporation, New York, No 4, 653-665 (1973)
24. Villa, T., Yevtushenko, N., Brayton, R. K., Mishchenko, A., Petrenko, A., Sangiovanni-Vincentelli A. L. The Unknown Component Problem: Theory and Applications, Springer, (2012).
25. Villa, T., Petrenko, A., Yevtushenko, N., Mishchenko, A., Brayton, R., Component-based design by solving language equations, Proceedings of the IEEE, Vol. 103, No. 11, 2152-2167 (2015)
26. Zafiropulo, P., West, C., Rudin, H., Cowan, D., Brand, D. Towards analyzing and synthesizing protocols. IEEE Transactions on Communications, 28(4): 651–661 (1980)