# Conformance Testing and Inference of Embedded Components

Alexandre Petrenko(✉) and Florent Avellaneda

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

**Abstract.** The problems of active inference (learning) and conformance testing of a system modelled by an automaton have actively been studied for decades, however, much less attention has been paid to modular systems, modelled by communicating automata. In this paper, we consider a system of two communicating FSMs, one machine represents an embedded component and another the remaining part of the system, the context. Assuming that the context FSM is known, we want to learn the embedded FSM without directly interacting with it. This problem can be viewed as a generalization of the classical automata inference in isolation, i.e., it is the grey box learning problem. The proposed approach to solve this problem relies on a SAT-solving method for FSM inference from traces. It does not depend on the composition topology and allows at the same time to solve a related problem of conformance testing in context. The latter is to test whether an embedded implementation FSM composed with the given context is equivalent to the embedded specification FSM also composed with the context. The novelty of the conformance testing method is that it directly generates a complete test suite for the embedded machine and avoids using nondeterministic approximations with their tests, eliminating thus several sources of test redundancy inherent in the existing methods.

**Keywords:** Active inference · FSM learning · Conformance testing
Component-based systems · Embedded testing · Testing in context
SAT solving

## 1 Introduction

Componentization is an important engineering principle. Top-down design approaches brake down large systems into smaller parts, components, and bottom-up approaches compose existing components into larger systems. While practitioners are mostly using ad hoc techniques, model-based software engineering is investigating formal approaches which can offer automation to various phases of modular system development, including testing and using legacy components and components of the shelf, COTS.

The existing model-based testing approaches focus mostly on a holistic view of a modular system, based on a single state-oriented model, see, e.g., [7]. Conformance tests are then generated from a state machine, which models either a component in isolation or a whole system as observed on external interfaces [4, 5, 7]. On the other hand, when a system is built using existing components, testing efforts should be focused only on new components [17]. This motivates research in conformance testing in context aka embedded testing, which aims to check whether an embedded implementation FSM composed with the given context is equivalent to the embedded specification FSM also composed with the context.

All the known methods for complete tests generation for testing in context first construct from the context and embedded machine an embedded equivalent or the largest solution to the appropriate FSM equation, which represents the behavior of the embedded machine as can be controlled and observed via context [2, 11, 12, 14]. The resulting partial machine is a nondeterministic approximation of the embedded deterministic machine, it is then used to derive complete internal tests, which are finally translated into external ones executed on external interfaces of a modular system.

Conformance testing is closely related to active inference, aka query learning, as already been understood, see, e.g., [10], for the case when a system is considered "as a whole", i.e., modelled as an FSM.

Model inference helps in dealing with legacy components and COTS. Once a model is reengineered it can be used to perform verification with model checkers, regression and integration testing or redesign. Automata inference is an important topic addressed in many works, see, e.g., [1, 3, 8, 9, 13, 15], which treat a system as a single black box, even if it contains components with known models and only some need to be learned.

We propose to generalize the FSM inference problem to the case when an FSM to learn is a part of a modular system. Indeed, the traditional automaton/FSM inference problem statement is a particular case of this general situation, namely, when the rest of the system is a single state machine performing just a bijection of external and internal inputs. We know the only work [18] addressing the grey box learning problem, where the goal is to learn a tail FSM in the serial composition with the context FSM. We propose an approach for solving the grey box learning problem that does not depend on the composition topology, as opposed to the previous work [18].

To simplify the discussions, we model a system with two communicating FSMs, one machine represents an embedded component and another the remaining part of the system, the context. Assuming that the context FSM is known, we elaborate an approach to learn the embedded FSM without directly interacting with it. The proposed approach relies on a SAT-solving method for FSM inference from sample traces. The approach also allows to solve the problem of conformance testing in context, which is to check whether an embedded implementation FSM composed with the given context is equivalent to the embedded specification FSM also composed with the context. The novelty of the conformance testing method is that it directly generates a complete test suite for the embedded machine and avoids using nondeterministic approximations with their tests, eliminating thus several sources of test redundancy inherent in the existing methods.

The paper is organized as follows. Section 2 provides definitions related to state machines and automata needed to formalize the approach. Communicating FSMs are formally defined and illustrated on a working example in Sect. 3. A SAT-solving method for FSM inference from traces, which allows to obtain different conjectures [10] as required by the proposed methods, is recalled in Sect. 4. Section 5 presents our method for complete test generation for embedded components. In Sect. 6 we present some experimental results concerning test generation. Section 7 describes the method for embedded FSM inference and Sect. 8 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*. 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 $M/s_0$. For trace $\omega \in Tr(s)$, we use $s$-after-$\omega$ to denote the state $M$ reached after the execution of $\omega$, for an empty trace $\varepsilon$, $s$-after-$\varepsilon = s$. When $s$ is the initial state we write $M$-after-$\omega$ instead of $s_0$-after-$\omega$.

Given a string $\omega \in (IO)^*$, the *I-restriction* of $\omega$ is a string obtained by deleting from $\omega$ 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-$\omega$. The length of a trace 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 $\alpha$, we let $out(s, \alpha)$ denote the $O$-restriction of the trace that has $\alpha$ as its $I$-restriction. States $s, s' \in S$ are *equivalent w.r.t.* $\alpha$, if $out(s, \alpha) = out(s', \alpha)$, denoted $s \cong_\alpha s'$; they are *distinguishable* by $\alpha$, if $out(s, \alpha) \neq out(s', \alpha)$, denoted $s \ncong_\alpha s'$ or simply $s \ncong s'$. 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 is similarly defined. Two 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 $(s, s') \in P$, $(s, x, o, t) \in T$, $(s', x, o', t') \in T'$, and $o = o'$, 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.

Two complete FSMs $M = (S, s_0, I, O, T)$ and $M' = \left(S', s_0', I, O, T'\right)$ are called *isomorphic* if there exists a bijection $f$: $S \rightarrow S'$ such that $f(s_0) = s'_{00}$ and for all $a \in I$, $o \in O$, and $s \in S$, $f(s\text{-after-}ao) = f(s)\text{-after-}ao$. Isomorphic FSMs are equivalent, but the converse does not necessarily hold.

Given a string $\omega \in (IO)^*$ of length $|\omega|$, let $Pref(\omega)$ be the set of all prefixes of $\omega$. We define a (linear) FSM $W(\omega) = (X, x_0, I, O, D_\omega)$, where $D_\omega$ 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 $\omega$-*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 $\Omega$ is the set of its traces, called an $\Omega$-*machine*. The bijection $f$ relates states of this machine to traces in $\Omega$.

While the set of traces of the $\Omega$-machine is $\Omega$, there are many FSMs which contain the set $\Omega$ among their traces. We restrict our attention to the set of all FSMs with at most $n$ states and alphabets $I$ and $O$, denoted $\Im(n, I, O)$. An FSM $C = (S, s_0, I, O, T)$, $C \in \Im(n, I, O)$ is called an $\Omega$-*conjecture*, if $\Omega \subseteq Tr_C$.

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

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

A finite set of input sequences $L \subset I^*$ is a *checking experiment* for a complete FSM $M$ with $n$ states if for each FSM $N \in \Im(n, I, O)$, such that $N \cong_L M$, it holds that $N \cong M$. Checking experiments are also called *complete* (i.e., sound and exhaustive) tests.

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 $\varepsilon$ represents an internal action, and $F$ is a set of *final* or *accepting* states. We shall use several operations over automata, namely, expansion, restriction, and intersection, following [16].

Given an automaton $A$ and a finite alphabet $U$, $U \cap X = \varnothing$, the *U-expansion* of automaton $A$ is the automaton denoted $A_{\uparrow U}$ obtained by adding at each state a self-looping transition labeled with each action in $U$.

For an automaton $A$ and an alphabet $U \subseteq X$, the *U-restriction* of automaton $A$ is the automaton denoted $A_{\downarrow U}$ obtained by replacing each transition with the symbol in $X \backslash U$ by an $\varepsilon$-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 $(r, a, r') \in Z$.

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 Communicating FSMs

The behavior of a modular system composed of FSM components depends on its environment. The two together constitute a closed system. Communications in it can either be via messages or by method calls, using no queues. We restrict our attention to the case when queues are not used, which is possible with a so-called slow environment assuming that the closed system operates with a single message in transit [12]. This is a sufficient condition for the existence of a deterministic FSM modelling the external behavior of a modular system [12, 16]. Such an environment can be modelled by a (chaos) automaton *Env* with two states, out of which the initial state is the final state, shown in Fig. 1. After issuing an external input in *X* to the system it waits until an external output in *O* is produced before executing a next input. Its language is the set (*XO*)*.

We further consider only two deterministic FSMs, one of them representing an embedded component *E* and another the remaining part of the modular system, aka context *K*, as shown in Fig. 1.

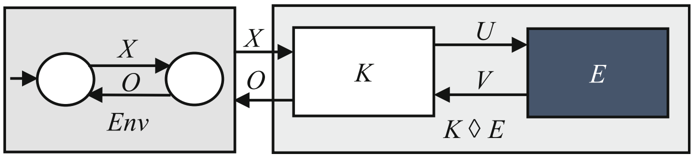

**Fig. 1.** Closed system of two FSMs *K* and *E* with the environment.

We assume that the sets *X*, *O*, *U*, and *V* are pairwise disjoint. The context FSM *K* assumed to be a complete machine interacts with the environment and can process an external input after it produces an internal output even before an external output is emitted. Since this violates the restriction of having a single message in transit, we constrain its behavior by composing it with the slow environment. Let *A*(*K*) be the automaton of the context FSM *K*. Then the intersection of automata $A(K) \cap Env_{\uparrow U \cup V} = A(K)_{\text{slow}}$ represents the behavior of the context constrained by the slow environment. Then the intersection $A(K)_{\text{slow}} \cap A(E)_{\uparrow X \cup O}$ denoted by *A*(*K*) ◊ *A* (*E*) describes the behavior of the closed system, called the *composite automaton* of the modular system.

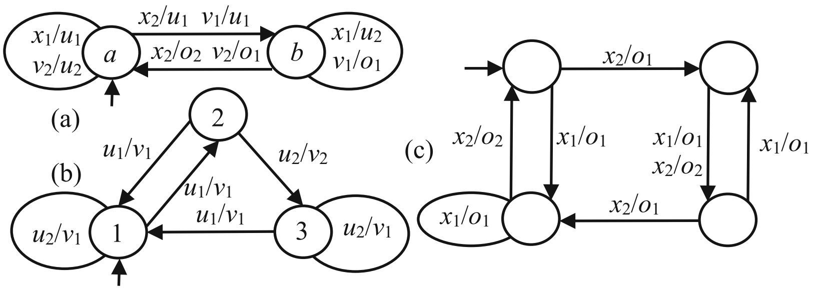

**Fig. 2.** The context FSM $K$ (a), embedded FSM $E$ (b) and composite FSM $K \lozenge E$ (c).

The language of $A(K) \lozenge A(E)$ is the set of all strings labelling all the executions of the system $L(A(K) \lozenge A(E))$. Restricting a string to the alphabets of a component FSM we obtain a trace of the context or embedded FSM. 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 $A(K) \lozenge A(E)$, i.e., external traces of the system. They are traces of an FSM, provided that $A(K) \lozenge A(E)$ has no livelocks, i.e., cycles labelled by symbols in $U \cup V$ [16], the machine can be obtained by removing ε-transitions in $A(K) \lozenge A(E)_{\downarrow X \cup O}$ and pairing each input with a subsequent output, if it exists, to an FSM transition's label. If some external input is not followed by an external output it is deleted from the corresponding state of $(A(K) \lozenge A(E))_{\downarrow X \cup O}$, as a result, the FSM becomes partial. If all inputs are deleted from the initial state then the machine has a single state and no transition. We let $K \lozenge E$ denote the resulting FSM, called the *composite FSM* of the modular system.

Given two FSMs $E$ and $E'$ over the alphabets $U$ and $V$, such that the composite machines $K \lozenge E$ and $K \lozenge E'$ are complete FSMs, we say that $E$ and $E'$ are *externally equivalent* (or *equivalent in context*) if $K \lozenge E \cong K \lozenge E$. Clearly, $E \cong E'$ implies $K \lozenge E \cong K$, but the converse does not hold. Testing in context uses external equivalence as a conformance relation between implementations of a component embedded in a modular system and its specification.

A finite set of input sequences $L \subset X^*$ is an *external* checking experiment (complete test suite) for the embedded FSM $E$ w.r.t. $\mathfrak{J}(n, U, V)$, if for each FSM $N \in \mathfrak{J}(n, U, V)$, where $n$ is the number of states in $E$ such that $K \lozenge N \cong_L K \lozenge E$, it holds that $K \lozenge N \cong K \lozenge E$.

*Example*. Consider the context FSM $K$ and embedded FSM $E$ shown in Fig. 2 together with the composite FSM $K \lozenge E$. The composite automaton $A(K) \lozenge A(E)$ is shown in Fig. 3.

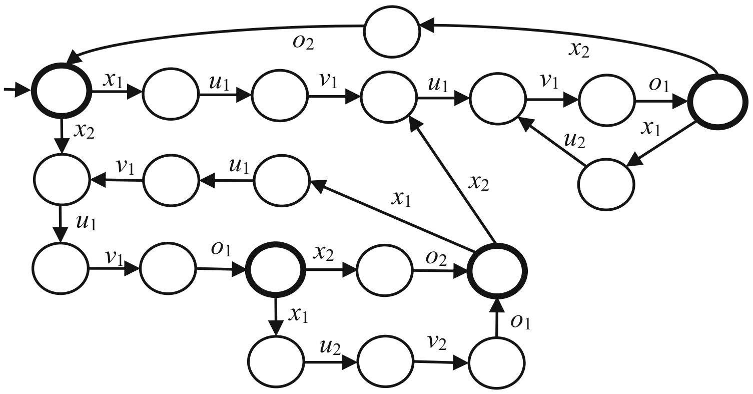

**Fig. 3.** The composite automaton $A(K) \diamondsuit A(E)$, final states are in bold.

## 4 Passive Inference with SAT-Solving

Henceforth, we first provide a brief overview of the SAT-solving based method for conjecture generation from a given set of traces avoiding regeneration of already considered conjectures which is the basic step of testing and learning an FSM in isolation [10] and an embedded FSM, as we show in Sect. 5. For a detailed presentation, the reader is referred to [10].

The basic step of conjecture inference from a given set of traces Ω is state merging of the Ω-machine. SAT-solving approaches [6, 9] encode the problem into Boolean constraints, a solution if it exists is a conjecture with a given number of states. We use an existing encoding of a set of traces Ω into a Boolean formula *formula* [6]. Let $W(\Omega) = (X, x_0, I, O, D_\Omega)$ be the Ω-machine. Each state of the Ω-machine is represented by a variable $x$, so $x_i \in \{0, \ldots, n-1\}$. Since the Ω-machine is deterministic, the state variables satisfy the constraint [1]:

$$\begin{gathered}\forall_{xi}, x_j \in\text{:}\ \textit{if}\ x_i \not\cong x_j\ \text{then}\ x_i \neq x_j\ \text{and}\\ \text{if}\ \exists a \in I\ \text{s.t.}\ out(x_i, a) = out(x_j, a) = o\ \text{then}\ x_i = x_j \Rightarrow x_i\text{-after-}ao = x_j\text{-after-}ao\end{gathered} \tag{1}$$

An assignment of values to variables such that the formula (1) is satisfied defines a mapping μ: $X \rightarrow S$, where $S$ is the set of states of an Ω-conjecture, i.e., the mapping μ defines a partition of $X$ into $n$ blocks.

These CSP (constraint satisfaction problem) formulas are then translated to SAT using unary coding for integer variables, represented by $n$ Boolean variables $v_{x,0}, \ldots, v_{x,n-1}$. For each $x \in X$, we have the clause:

$$v_{x,0} \vee \ldots \vee v_{x,n-1} \tag{2}$$

For each state $x \in X$ and all $i, j \in \{0, \ldots, n-1\}$ such that $i \neq j$, we have the clauses:

$$\neg v_{x,i} \vee \neg v_{x,j} \tag{3}$$

We use auxiliary variables $e_{x,\ y}$ [6]. For every $x, y \in X$ such that $x \not\simeq y$ we have

$$\neg e_{x,y} \tag{4}$$

For all $x, y \in X$ such that $out(x, a) = out(y, a) = o$, we have

$$e_{x,y} \Rightarrow e_{x-\text{after}-ao,y-\text{after}-ao} \tag{5}$$

For every $x, y \in X$ and all $i \in \{0, \ldots, n-1\}$

$$e_{x,y} \wedge v_{x,i} \Rightarrow v_{y,i} \tag{6}$$

$$\neg e_{x,y} \wedge v_{x,i} \Rightarrow \neg v_{y,i} \tag{7}$$

The resulting Boolean formula is the conjunction of clauses (2)–(7).

The traditional use of SAT solvers for state minimization aims at obtaining a single conjecture, while the problems of conformance testing and learning require that constraints should allow a solver to check, once a conjecture is found, whether another non-equivalent conjecture exists. Absence of a conjecture proves that a checking experiment is constructed and the machine is identified.

This is achieved by using the following procedure to infer a conjecture that differs from already considered conjectures. Isomorphic conjectures are identified by their common partition, encoded into an additional constraint. Recall that states of an Ω-machine form a partition defined by an Ω-conjecture. We let Π denote a set of partitions of states of Ω′-machines, where Ω′ ⊆ Ω.

**Algorithm 1.** *Infer_conjecture*(Ω, *n*, Π) [10]
**Input:** A set of traces Ω, an integer *n*, and a set of partitions Π
**Output:** An Ω-conjecture with at most *n* states such that its partition does not expand any partition in Π, or False.

```
1. $formula$ = conjunction of the clauses (2) - (7)
2. for all $\pi \in \Pi$ do
3.         $clause$ = False
4.         for all $x, y$ such that $x =_\pi y$ do
5.         $clause = clause \vee \neg e_{x,y}$
6.         end for
7.         $formula = formula \wedge clause$
8. end for
9. return $call\text{-}solver(formula)$
```

To check the satisfiability of a formula one can use any of the existing solvers, calling the function *call-solver*(*formula*). If a solution exists then we have an Ω-conjecture with *n* or fewer states. The latter is obtained from the determined partition on *X*.

## 5 External Checking Experiment Construction

Solving the problem of external checking experiment generation, we use as in our previous work [10] Algorithm 1 for conjecture inference from a current set of traces. The difference, however, is that traces now no longer belong to a machine considered in isolation (this becomes even more crucial for active inference), they are produced by an embedded component. Accordingly, instead of checking the equivalence of a conjecture to the specification machine, we must check their external equivalence. To this end, we need to compose a conjecture $C$ and the context $K$. As discussed above, if the resulting composite automaton $A(K) \Diamond A(C)$ has a livelock, its external behavior cannot be specified by an FSM, since an external input triggering livelock cannot be paired with any output. To deal with this issue we formulate a new constraint (in the form of a partition, as before) avoiding its regeneration by a solver. Once the current conjecture composed with the context yields a composite FSM $K \Diamond C$ an external input sequence distinguishing it from the given composite machine $K \Diamond E$ can be determined, if they are not equivalent. The found sequence is added to a current set of input sequences. The distinguishing external input sequence is the $X$-restriction of the word of the automaton $A(K) \Diamond A(C)$ from which a trace of the embedded component is obtained as the $(U \cup V)$-restriction and used to generate a next conjecture. If, however, no new trace of the embedded component is obtained and the state partition of the conjecture distinguishable from the specification machine is added as a constraint to avoid its regeneration. The process iterates until the constraints are no longer satisfiable. The procedure is implemented in Algorithm 2.

**Algorithm 2.** Generating external checking experiment

**Input:** Complete deterministic FSMs $K$ and $E$ such that the composite machine $K \Diamond E$ is a complete FSM

**Output:** An external checking experiment $\Psi$ for the embedded FSM $E$ w.r.t. $\Im(n, U, V)$

```
1. $\Omega := \varnothing$
2. $\Psi := \varnothing$
3. $\Pi := \varnothing$
4. while an $\Omega$-conjecture $C$ is returned by $Infer\_conjecture(\Omega, n, \Pi)$ do
5.     if $A(K) \Diamond A(C)$ has a livelock or $(K \Diamond C) \times (K \Diamond E)$ is complete then
6.         $\Pi := \Pi \cup \{\pi_C\}$
7.     else
8.         Determine an input sequence $\beta a$ such that $\beta$ is the shortest transfer
           sequence to a state with the undefined input $a$ in $(K \Diamond C) \times (K \Diamond E)$
9.         $\Psi := \Psi \cup \{\beta a\}$
10.        Let $\sigma \in L(A(K) \Diamond A(E))$ such that $\sigma_{\downarrow X} = \beta a$
11.        if $\sigma_{\downarrow(U \cup V)} \in \Omega$ then
12.            $\Pi := \Pi \cup \{\pi_C\}$
13.        else
14.            $\Omega := \Omega \cup \{\sigma_{\downarrow(U \cup V)}\}$
15.        end if
16.    end if
17. end while
18. return $\Psi$
```

Algorithm 2 calls *Infer_conjecture*(Ω, $n$, Π), which in turn calls a SAT solver constraining it to avoid solutions of already considered conjectures.

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

*Example.* We illustrate Algorithm 2 using the context and embedded machines in Fig. 2. We assume $n = 2$. Initially, the set of external input sequences Ψ is empty, so is the set of internal input sequences Ω. The function *Infer_conjecture*(Ω, $n$, Π) for the empty set Π returns a Ω-conjecture $C_0$ as an FSM with a single state and no transitions. The composite machine $K \lozenge C$ has no transitions either. We choose the external input $x_1$, so $\Psi = \{x_1\}$. This input is the $X$-restriction of the word $\sigma = x_1u_1v_1u_1v_1o_1$ in $A(K) \lozenge A(E)$. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1 . \Omega = \{u_1v_1u_1v_1\}$. The function *Infer_conjecture*(Ω, $n$, Π) for the empty set Π returns a Ω-conjecture $C_1$ as an FSM with a single state and transition labelled $u_1/v_1$. The composite FSM $K \lozenge C_1$ is shown in Fig. 4(a).

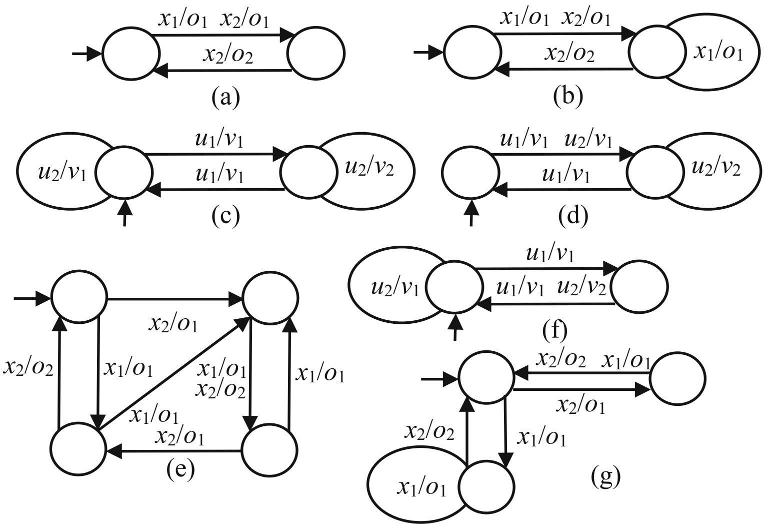

**Fig. 4.** Constructing the external checking experiment.

The FSM $K \lozenge C_1$ has an undefined input $x_1$ in the second state, we take the external input sequence $x_1x_1$, so now $\Psi = \{x_1x_1\}$. It is the $X$-restriction of the word $\sigma = x_1u_1v_1u_1v_1o_1x_1u_2v_1o_1$ in $A(K) \lozenge A(E)$. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1u_2v_1 . \Omega = \{u_1v_1u_1v_1u_2v_1\}$. The function *Infer_-conjecture*(Ω, $n$, Π) for the empty set Π returns a Ω-conjecture $C_2$ as an FSM with a single state and two transitions labelled $u_1/v_1$ and $u_2/v_1$. The composite FSM $K \lozenge C_2$ is shown in Fig. 4(b). It is a complete machine, however, the product $(K \lozenge C) \times (K \lozenge E)$ is a partial machine, since its behavior is not specified for the input sequence $x_2x_1x_2$. In fact, this sequence demonstrates that $K \lozenge C_2 \cong K \lozenge E$, since $K \lozenge C_2$ reacts with the $o_1o_1o_2$, while $K \lozenge E$ with $o_1o_1o_1$.

The input sequence $x_2x_1x_2$ is added to $\Psi$, which becomes $\{x_1x_1, x_2x_1x_2\}$. The sequence $x_2x_1x_2$ is the $X$-restriction of the word $\sigma = x_2u_1v_1o_1\ x_1u_2v_2o_1x_2u_1v_1o_1$ in $A(K) \lozenge A(E)$. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_2v_2u_1v_1$. $\Omega = \{u_1v_1u_1v_1u_2v_1, u_1v_1u_2v_2u_1v_1\}$. Next $\Omega$-conjecture $C_3$ is shown in Fig. 4(c). The composite FSM $K \lozenge C_3$ is isomorphic to $K \lozenge E$. Now the set of partitions $\Pi$ should include the following partition of prefixes of $\Omega$, as each of them is a state of the $\Omega$-machine:

$$\pi_1 = \{\varepsilon, u_1v_1u_1v_1, u_1v_1u_1v_1u_2v_1, u_1v_1u_2v_2u_1v_1; u_1v_1, u_1v_1u_2v_2\}.$$

In the next iteration, *Infer_conjecture*($\Omega$, $n$, $\Pi$) returns the $\Omega$-conjecture $C_4$ shown in Fig. 4(d). The composite FSM $K \lozenge C_4$ is shown in Fig. 4(e). It is not equivalent to $K \lozenge E$, and the shortest input sequence distinguishing them is $x_1x_1x_1x_2$, it extends the existing sequence $x_1x_1$.

The input sequence $x_1x_1x_1x_2$ is added to $\Psi$, which becomes $\{x_1x_1x_1x_2, x_2x_1x_2\}$. It is the $X$-restriction of the word $\sigma = x_1u_1v_1u_1v_1o_1x_1u_2v_1o_1x_1u_2v_1o_1x_2o_2$ in $A(K) \lozenge A(E)$. We have $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1u_2v_1u_2v_1$. $\Omega = \{u_1v_1u_1v_1u_2v_1u_2v_1, u_1v_1u_2v_2u_1v_1\}$. Next $\Omega$-conjecture $C_5$ is shown in Fig. 4(f). The composite FSM $K \lozenge C_5$ is shown in Fig. 4 (g). It is not equivalent to $K \lozenge E$, and the shortest input sequence distinguishing them is $x_2x_1x_2x_1x_2$, it extends the existing sequence $x_2x_1x_2$.

The input sequence $x_2x_1x_2x_1x_2$ is added to $\Psi$, which becomes $\{x_1x_1x_1x_2, x_2x_1x_2x_1x_2\}$. It is the $X$-restriction of the word $\sigma = x_2u_1v_1o_1\ x_1u_2v_2o_1x_2u_1v_1o_1x_1$-$u_2v_1o_1x_2o_2$. We have $\sigma_{\downarrow(U \cup V)} = u_1v_1u_2v_2u_1v_1u_2v_1$. $\Omega = \{u_1v_1u_1v_1u_2v_1u_2v_1, u_1v_1u_2v_2\ u_1u_2v_1\}$. The function *Infer_conjecture*($\Omega$, $n$, $\Pi$) returns False, since there is no solution which does not extend the partition $\pi_1$. Algorithm 2 terminates with the external checking experiment $\Psi = \{x_1x_1x_1x_2, x_2x_1x_2x_1x_2\}$.

This example was used in the previous work [12] to illustrate a number of various approaches to construct complete tests for the embedded component, compared to them the SAT solving approach elaborated here generates a much smaller test suite. For comparison, we construct the same experiment assuming this time that $n = 3$. The prototype tool presented in Sect. 7 returns just seven tests.

Notice that the algorithm not only delivers an external checking experiment for the embedded component, but also infers an FSM that is externally equivalent to the given embedded FSM. In our example, the $\Omega$-conjecture $C_3$ in Fig. 4(c) is externally equivalent to the FSM $E$ in Fig. 2. This observation indicates that the approach should work for active inference of an embedded component. We elaborate a corresponding algorithm in Sect. 7.

**Theorem 1.** Given complete deterministic FSMs $K$ and $E$ such that the composite machine $K \lozenge E$ is a complete FSM, Algorithm 2 returns an external checking experiment for the embedded FSM $E$ w.r.t. $\mathfrak{J}(n, U, V)$.

**Sketch of Proof.** When Algorithm 2 terminates the resulting set of external input sequence $\Psi$ is indeed a checking experiment, since by the post-condition of *Infer_-conjecture* no conjecture exists that is not externally equivalent to the given embedded FSM $E$. Note that all complete conjectures externally equivalent to $E$ are excluded because as soon as one if found (including $E$ itself), its partition is added to $\Pi$. Algorithm 2 will not generate the same conjecture all over again and always terminates because the set of all possible conjectures with at most $n$ states is finite.

## 6 Preliminary Experiments

The complexity of checking experiments for complete deterministic FSMs is well understood, however, no result exists yet on estimating complexity of external checking experiments for embedded FSMs. Considering a system of two communicating machines, context and embedded FSMs, the question arises which of them contribute more to the complexity of external checking experiments. We decided to perform experiments aiming at shedding some light on this.

Both machines are generated randomly for $|X| = |O| = |U| = |V| = 2$. In the first experiment, we fix the number of states of an embedded FSM to six and vary that of the context; in the second experiment, we fix the number of states of a context FSM to six and vary that of the embedded FSM. For each pair of values, the average of ten instances obtained with a prototype tool implementing Algorithm 2 is calculated and the results are illustrated in Fig. 5. They indicate that the length of experiments grows with the number of states in an embedded machine similar to an FSM considered in isolation, but the complexity of the context seems not to be a significant contributor. More experiments are needed to check this conclusion.

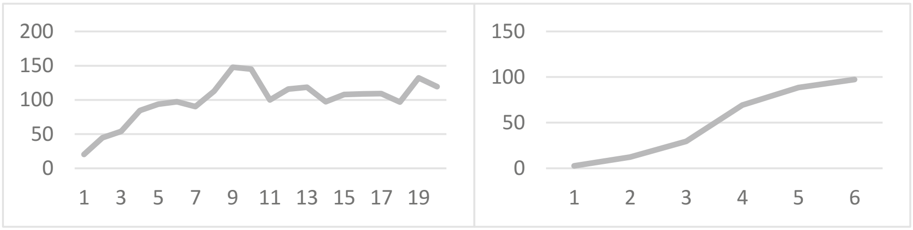

**Fig. 5.** The length of external checking experiments vs the number of states in the context (left hand side) and in the embedded machine (right hand side).

## 7 Active Inference of Embedded FSM

Given a composition of two complete deterministic FSMs $K$ and $E$ with the topology in Fig. 1, called a *grey box*, *GB*, where the context FSM $K$ is known, while the embedded FSM $E$ is not, we want to learn the machine $E$ by applying external inputs $I$ and observing external as well as internal outputs $O$, $U$, $V$, assuming that the embedded FSM $E$ has at most $n$ states.

External input sequences are applied to the grey box obeying the property of a slow environment *Env* (Fig. 1), i.e., inputs are interleaved with outputs. We assume that livelocks are removed from the grey box. The learning procedure is implemented in Algorithm 3. It is an enhancement of Algorithm 2 replacing the FSM *E* by a current conjecture.

**Algorithm 3.** Inferring the embedded FSM and determining an external checking experiment for it

**Input** A grey box *GB* and integer *n*

**Output** A minimal complete conjecture with at most *n* states and an external checking experiment w.r.t. $\Im(n, U, V)$

```
1: $\Omega := \varnothing$
2: $\Psi := \varnothing$
3: $\Pi := \varnothing$
4: $C := Infer\_conjecture(\Omega, n, \Pi)$
5: while an $\Omega$-conjecture $D$ is returned by $Infer\_conjecture(\Omega, n, \Pi)$ do
6:    if $A(K) \lozenge A(D)$ has a livelock or $(K \lozenge D) \times (K \lozenge C)$ is complete then
7:    $\Pi := \Pi \cup \{\pi_D\}$
8:    else
9:     Determine an input sequence $\beta a$ such that $\beta$ is the shortest transfer sequence
       from a state of $K \lozenge C$ reached by an input sequence $\mu \in \Psi \cup \{\varepsilon\}$ to a state with
       the undefined input $a$ in $(K \lozenge D) \times (K \lozenge C)$
10:    $\Psi := \Psi \cup \{\mu\beta a\}$
11:    Let $\sigma$ be a trace observed in $GB$ when the external input sequence $\mu\beta a$ is applied
12:    if $\sigma_{\downarrow(U \cup V)} \in \Omega$ then
13:    $\Pi := \Pi \cup \{\pi_D\}$
14:    else
15:    $\Omega := \Omega \cup \{\sigma_{\downarrow(U \cup V)}\}$
16:       if $\sigma_{\downarrow(U \cup V)} \notin Tr_C$ then
17:       $C := Infer\_conjecture(\Omega, n, \Pi)$
18:       end if
19:    end if
20: end while
21: return $C$ and $\Psi$
```

*Example.* We illustrate Algorithm 3 using the context and embedded machines in Fig. 2. The embedded FSM is the one to be inferred and we use the composite automaton in Fig. 3 as the grey box. We assume $n = 2$. Initially, the set of external input sequences $\Psi$ is empty, so is the set of internal input sequences $\Omega$. The function *Infer_conjecture*($\Omega$, $n$, $\Pi$) for the empty set $\Pi$ returns a $\Omega$-conjecture $C_0$ as an FSM with a single state and no transitions. Its second execution yields $D_0$ which has no transition. $(K \lozenge C_0) \times (K \lozenge D_0)$ has no transitions either. We choose the external input $x_1$, so $\Psi = \{x_1\}$. When this input is applied to the grey box, the trace $\sigma = x_1u_1v_1u_1v_1o_1$ is observed. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1$. $\Omega = \{u_1v_1u_1v_1\}$. The function *Infer_conjecture*($\Omega$, $n$, $\Pi$) for the empty set $\Pi$ returns a $\Omega$-conjecture with a single state and transition labelled $u_1/v_1$. This machine becomes now the conjecture $C_1$. The composite FSM $K \lozenge C_1$ is shown in Fig. 4(a). Next execution of the loop yields the conjecture $D_1$ equivalent to $C_1$.

The product $(K \lozenge C_1) \times (K \lozenge D_1)$ has an undefined input $x_1$ in the second state, we take the external input sequence $x_1x_1$, so now $\Psi = \{x_1x_1\}$. When this input sequence is applied to the grey box, the trace $\sigma = x_1u_1v_1u_1v_1o_1x_1u_2v_1o_1$ is observed. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1u_2v_1$. $\Omega = \{u_1v_1 u_1v_1u_2v_1\}$. The function *Infer_conjecture*($\Omega$, $n$, $\Pi$) for the empty set $\Pi$ returns a $\Omega$-conjecture $D_2$ with a single state and two transitions labelled $u_1/v_1$ and $u_2/v_1$. This machine becomes now the conjecture $C_2$. The composite FSM $K \lozenge C_2$ is shown in Fig. 4 (b). The product $(K \lozenge C_2) \times (K \lozenge D_2)$ is a complete machine, $\Pi := \Pi \cup \{\pi_{D_2}\}$ is executed, where $\pi_{D2} = \{\varepsilon, u_1v_1, u_1v_1u_1v_1, u_1v_1u_1v_1u_2v_1\}$. The function *Infer_conjecture* ($\Omega$, $n$, $\Pi$) for the set $\Pi$ returns a $\Omega$-conjecture $D_3$ shown in Fig. 6(a). The composite FSM $K \lozenge D_3$ is shown in Fig. 6(b). Its behavior is not specified for the input sequence $x_2x_1$.

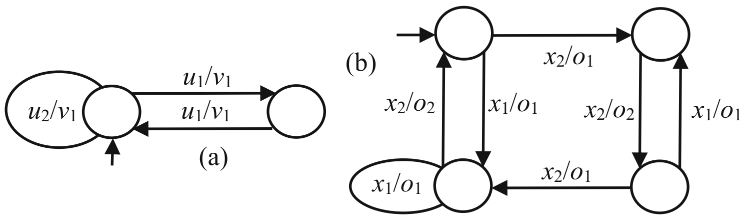

**Fig. 6.** $\Omega$-conjecture $D_3$ (a) and the composite FSM $K \lozenge D_3$.

The input sequence $x_2x_1$ is added to $\Psi$, which becomes $\{x_1x_1, x_2x_1\}$. The grey box produces the trace $\sigma = x_2u_1v_1o_1\ x_1u_2v_2o_1$ when the sequence $x_2x_1$ is applied. Its restriction onto the alphabets of the embedded component is $\sigma_{\downarrow(U \cup V)} = u_1v_1u_2v_2$. $\Omega = \{u_1v_1u_1v_1u_2v_1,\ u_1v_1u_2v_2\}$. Next $\Omega$-conjecture $D_4$ is shown in Fig. 4(c). This machine becomes now the conjecture $C_3$. The composite FSM $K \lozenge C_3$ is isomorphic to the FSM in Fig. 2(c). Then the set of partitions $\Pi$ should include the following partition: $\pi_{D_4} = \{\varepsilon, u_1v_1u_1v_1, u_1v_1u_1v_1u_2v_1; u_1v_1, u_1v_1u_2v_2\}$.

In the next iteration, *Infer_conjecture*($\Omega$, $n$, $\Pi$) returns the $\Omega$-conjecture $D_5$ shown in Fig. 4(d). The composite FSM $K \lozenge D_5$ is shown in Fig. 4(e). It is not equivalent to $K \lozenge C_3$, and the shortest input sequence distinguishing them is $x_1x_1x_1x_2$, it extends the sequence $x_1x_1$.

The input sequence $x_1x_1x_1x_2$ is added to $\Psi$, which becomes $\{x_1x_1x_1x_2,\ x_2x_1\}$. The sequence applied to the grey box produces the trace $\sigma = x_1u_1v_1u_1v_1o_1x_1u_2v_1o_1x_1 u_2v_1o_1x_2o_2$. We have $\sigma_{\downarrow(U \cup V)} = u_1v_1u_1v_1u_2v_1u_2v_1$. $\Omega = \{u_1v_1u_1v_1u_2v_1u_2v_1,\ u_1v_1 u_2v_2\}$. Next $\Omega$-conjecture $D_5$ is shown in Fig. 4(f). The composite FSM $K \lozenge D_5$ is shown in Fig. 4(g). It is not equivalent to $K \lozenge E$, and the shortest sequence distinguishing them is $x_2x_1x_2x_1x_2$, it extends the sequence $x_2x_1$.

The input sequence $x_2x_1x_2x_1x_2$ is added to $\Psi$, which becomes $\{x_1x_1x_1x_2, x_2x_1x_2x_1x_2\}$. Applied to the grey box it produces the trace $\sigma = x_2u_1v_1o_1\,x_1u_2v_2o_1x_2u_1v_1o_1x_1u_2v_1o_1x_2o_2$. We have $\sigma_{\downarrow(U \cup V)} = u_1v_1u_2v_2u_1v_1u_2v_1$. $\Omega = \{u_1v_1u_1v_1u_2v_1u_2v_1, u_1v_1u_2v_2u_1v_1u_2v_1\}$.

The function *Infer_conjecture*($\Omega$, $n$, $\Pi$) returns False, since no solution with a state partition which does not extend the partition $\pi_{D_4}$ can be found. Algorithm 3 terminates with the conjecture $C_3$ (Fig. 4(c)) that is externally equivalent to the embedded FSM $E$ in Fig. 3 and its external checking experiment $\Psi = \{x_1x_1x_1x_2, x_2x_1x_2x_1x_2\}$. In this example, both algorithms give the same experiment, though this should not be expected for other systems, since the function *call-solver*(*formula*) can make nondeterministic choices in solving constraints. Moreover, various input sequences can be chosen to deal with partial FSM products (see line 13 in Algorithm 3).

**Theorem 2.** If a grey box behaves as a complete FSM and the embedded FSM $E$ has $n$ states, Algorithm 3 infers a conjecture with at most $n$ states that is externally equivalent to $E$ and constructs an external checking experiment for it.

**Sketch of Proof.** Algorithm 3 follows the steps of Algorithm 2, just replacing the FSM $E$ by a current conjecture. This does not influence its termination since it only occurs when no more externally distinguishable conjecture can be found. At some point, because the grey box behaves as a composite FSM of the known context FSM and the embedded machine with $n$ states, an FSM that is externally equivalent to $E$ will be returned by *Infer_conjecture*. The resulting set of external input sequences is an external checking experiment for the resulting FSM, as in Theorem 1.

## 8 Conclusions

We considered a system of communicating FSMs and investigated possibilities for active learning and testing of an embedded FSM without disassembling the system. The contribution of this paper is the generalization of the isolated FSM inference problem to that of an FSM embedded in a modular system (grey box learning) and an approach for solving this problem that does not depend on the composition topology. The approach also offers a novel solution to embedded testing by generating a complete test suite directly for the embedded machine that avoids intermediate testing of non-deterministic approximations, eliminating thus several sources of test redundancy inherent in the existing methods. We plan to perform more experiments to assess the proposed methods, especially for learning embedded components.

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

## References

1. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. **100**(6), 592–597 (1972)
2. El-Fakih, K., Petrenko, A., Yevtushenko, N.: FSM test translation through context. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 245–258. Springer, Heidelberg (2006). https://doi.org/10.1007/11754008_16
3. Gold, E.M.: Complexity of automaton identification from given data. Inf. Control **37**(3), 302–320 (1978)

4. Groz, R., Li, K., Petrenko, A., Shahbaz, M.: Modular system verification by inference, testing and reachability analysis. In: Suzuki, K., Higashino, T., Ulrich, A., Hasegawa, T. (eds.) FATES/TestCom-2008. LNCS, vol. 5047, pp. 216–233. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68524-1_16
5. Groz, R., Li, K., Petrenko, A.: Integration testing of communicating systems with unknown components. Ann. Telecommun. **70**(3), 107–125 (2015)
6. Heule, Marijn J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J. M., García, P. (eds.) ICGI 2010. LNCS (LNAI), vol. 6339, pp. 66–79. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15488-1_7
7. Luo, G., Bochmann, G.V., Petrenko, A.: Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. IEEE Trans. Softw. Eng. **20**(2), 149–162 (1994)
8. Meinke, K.: CGE: a sequential learning algorithm for Mealy automata. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS (LNAI), vol. 6339, pp. 148–162. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15488-1_13
9. Oliveira, A.L., Silva, J.P.M.: Efficient algorithms for the inference of minimum size DFAs. Mach. Learn. **4**(1), 93–119 (2001)
10. Petrenko, A., Avellaneda, F., Groz, R., Oriat, C.: From passive to active FSM inference via checking sequence construction. In: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (eds.) ICTSS 2017. LNCS, vol. 10533, pp. 126–141. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67549-7_8
11. Petrenko, A., Yevtushenko, N., Bochmann, G.V., Dssouli, R.: Testing in context: framework and test derivation. Comput. Commun. **19**(14), 1236–1249 (1996)
12. Petrenko, A., Yevtushenko, N., Bochmann, G.V.: Fault models for testing in context. In: Gotzhein, R., Bredereke, J. (eds.) Formal Description Techniques IX. IFIP Advances in Information and Communication Technology. Springer, Heidelberg (1996). https://doi.org/10.1007/978-0-387-35079-0_10
13. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. In: Hanson, S.J., Remmele, W., Rivest, R.L. (eds.) Machine Learning: From Theory to Applications. LNCS, vol. 661, pp. 51–73. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56483-7_22
14. Petrenko, A., Yevtushenko, N.: Testing faults in embedded components. In: 10th International Workshop on Testing of Communicating Systems, pp. 272–287 (1997)
15. Steffen, B., et al.: Active automata learning: from DFAs to interface programs and beyond. In: ICGI, pp. 195–209 (2012)
16. Villa, T., Petrenko, A., Yevtushenko, N., Mishchenko, A., Brayton, R.: Component based design by solving language equations. Proc. IEEE **103**(11), 2152–2167 (2015)
17. 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)
18. Abel, A., Reineke, J.: Gray-Box learning of serial compositions of mealy machines. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 272–287. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_21