State-Identification Problems for Finite-State Transducers (extended abstract)

Moez Krichen and Stavros Tripakis
Verimag
Centre Equation, 2, avenue de Vignate, 38610 Gières, France. www-verimag.imag.fr.
Email: krichen@imag.fr, tripakis@imag.fr.

Abstract. A well-established theory exists for testing finite-state machines, in particular Moore and Mealy machines. A fundamental class of problems handled by this theory is state identification: we are given a machine with known state space and transition relation but unknown initial state, and we are asked to find experiments which permit to identify the initial or final state of the machine, called distinguishing and homing experiments, respectively.
In this paper, we study state-identification for finite-state transducers. The latter are a generalization of Mealy machines where outputs are sequences rather than symbols. Transducers permit to model systems where inputs and outputs are not synchronous, as is the case in Mealy machines. It is well-known that every deterministic and minimal Mealy machine admits a homing experiment. We show that this property fails for transducers, even when the latter are deterministic and minimal. We also provide partial answers to the decidability question, namely, checking whether a given transducer admits a particular type of experiment. First, we show how the standard successor-tree algorithm for Mealy machines can be turned into a semi-algorithm for transducers. Second, we identify a sub-class of transducers for which the problem is decidable. A transducer in this sub-class can be transformed into a (possibly non-deterministic) Mealy machine, to which existing methods apply.

1 Introduction

Testing is a fundamental step in any development process. It consists in applying a set of experiments to a system, with multiple aims, from obtaining some piece of unknown information to checking correctness or measuring performance. These different aims give rise to different classes of testing problems, for instance, conformance testing or performance testing.

A particularly interesting class of testing problems, pioneered in the seminal 1956 paper of Moore [12], is *state identification*. We are given an input-output machine with known state-transition diagram but unknown initial state. We are asked to perform an experiment in order to, either find the unknown initial state (*distinguishing* experiment), or verify that the machine is indeed in an assumed-to-be state (*state-verification* experiment), or identify the final state, reached at the end of the experiment (*homing* experiment), or lead the machine to a given state (*synchronizing* experiment), etc.

An extensive theory is available on state identification problems for Moore and Mealy machines. These machines have a common characteristic, namely, that inputs and outputs are *synchronous*: an input is immediately followed by an output. This implies that each output symbol in an output sequence σ corresponds to a unique input symbol in the input sequence that generated σ. Such a machine models a *length-preserving* function from input sequences to output sequences.

Models where inputs and outputs are synchronous are particularly well-suited for a number of applications, for example, synchronous circuits. They are not suitable, however, for other applications such as multi-threaded software, concurrent or real-time systems. In such systems, inputs and outputs are inherently *asynchronous*: an input may not give rise to an output immediately, but only some time latter; an output may require more than one inputs to occur in a certain order (thus, a single input produces no output at all); an input may produce a sequence of outputs rather than a single output; and so on.

Sometimes, with appropriate modeling, such applications can be casted in a synchronous input-output framework. For example, one may model an absence of output by a special output symbol $\perp$ which denotes precisely "no output"; or one may model a sequence of outputs by a special output symbol which denotes precisely this sequence. However, when doing so in a testing context, it is important to realize that one implicitly makes certain assumptions on the capabilities of the tester. For instance, in the case of $\perp$, one implicitly assumes that the absence of output is observable. In the case of a sequence of outputs, one implicitly assumes that this sequence can be distinguished from sequences which are identical but result from more than one inputs (thus, the output symbol o_{ab} modeling the sequence of outputs ab is distinguishable from the output sequence $o_a \cdot o_b$, where o_a models the output a and o_b the output b).

In this paper we study state identification problems in a more general context, namely, for the model of *finite-state transducers*. Each transition of a transducer is labeled by an input symbol and a sequence (possibly empty) of output symbols. Thus, a transducer is a generalization of a Mealy machine where the output of a transition is a sequence rather than a single output symbol.

It is well-known that in the case of deterministic and minimal Mealy machines a homing experiment always exists. We show that this is not the case for finite-state transducers, even when the latter are deterministic and minimal. Consequently, the question arises: is there an algorithm to check, given a finite-state transducer, whether it admits a homing experiment? We generalize this question to consider distinguishing experiments as well. Since existence of any type of experiment is not guaranteed anyway, we make no assumptions on determinism or minimality of the transducers.

In the rest of the paper, we offer partial answers to the above question. First, we consider a standard algorithm for finding state-identification experiments in Mealy machine, namely, the *successor-tree* algorithm [8].[1] There is a simple way to modify this algorithm and obtain a *semi-algorithm* which can handle transducers. It is a semi-algorithm in the sense that termination is not guaranteed. Indeed, the standard termination conditions do not apply in our case, and it is not clear how they can be adapted. Nevertheless, we do provide some sufficient conditions for termination.

Second, we consider a sub-class of the transducer model, called *output-bounded, wait-synchronize* transducers (OBWS-FSTs). Roughly speaking, an OBWS-FST is a transducer where (1) the user (tester) can eventually apply a special input wait after which it is allowed to "synchronize" with the outputs (i.e., it can safely observe all remaining outputs) and (2) the number of outputs that can be generated until the synchronization occurs is finite. We show that state identification is decidable for OBWS-FSTs. The method consists in transforming the transducer to a (possibly non-deterministic) Mealy machine and then applying existing algorithms [1].

The rest of this paper is organized as follows. In Section 2 we present the model of finite-state transducers. In Section 3 we define the various state-identification problems. In Section 4 we recall the successor-tree algorithm and show how it can be turned into a semi-algorithm for the transducer model. In Section 5 we define the sub-class of OBWS-FSTs and provide an algorithm

[1] Although this is not the most efficient algorithm, it is the simplest, thus, serves as a good starting point for studying decidability.

for state-identification problems in this class. Section 6 gives directions for future work.

Related work

As mentioned above, an extensive theory is available on state identification problems for various machine models, including deterministic Moore and Mealy machines [12, 5, 17, 18, 8, 19, 9, 20, 10] and non-deterministic Mealy machines [1].

We are not aware of any previous work on testing in general or state identification in particular in the context of transducers. Transducers are well-studied, though, from the point of view of automata theory: for instance, see [3, 6, 7, 16, 2, 14, 11]. Variants of the transducer model have been used in natural-language processing and other applications: for instance, see [15, 13].

2 Finite-state transducers

A *finite-state transducer* (FST) is a quadruple $T = (Q, \mathsf{In}, \mathsf{Out}, E)$, where:

- $Q = \{q_1, q_2, ..., q_n\}$ is a finite set of states;
- $\mathsf{In} = \{a, b, ...\}$ is a finite set of input symbols;
- $\mathsf{Out} = \{0, 1, ...\}$ is a finite set of output symbols;
- $E \subseteq Q \times Q \times \mathsf{In} \times \mathsf{Out}^*$ is a finite set of transitions.

A transition $(q, q', a, \sigma) \in E$ is denoted as $q \xrightarrow{a/\sigma} q'$. The interpretation is that, when the transducer is at state q and receives input a, it may move to q' and output σ. Notice that σ may be the empty sequence, $\sigma = \epsilon$. Also note that non-determinism is allowed.

For $a_1, ..., a_k \in \mathsf{In}$, $\sigma_1, ..., \sigma_k \in \mathsf{Out}^*$ and $q_0, q_1, ..., q_k \in Q$, we use the notation $q_0 \xrightarrow{\pi/\sigma} q_k$ iff $\forall i \in \{1, ..., k\}.q_{i-1} \xrightarrow{a_i/\sigma_i} q_i$, $\pi = a_1 \cdots a_k$ and $\sigma = \sigma_1 \cdots \sigma_k$. By convention $q \xrightarrow{\epsilon/\epsilon} q$ holds for any $q \in Q$. We also use the notation $q \xrightarrow{\pi/\sigma}$ as a shorthand for $\exists q'.q \xrightarrow{\pi/\sigma} q'$.

T is called *input-complete* if $\forall q \in Q.\forall a \in \mathsf{In}.\exists \sigma \in \mathsf{Out}^*.q \xrightarrow{a/\sigma}$. It is called *deterministic* if $\forall q, q', q'' \in Q\,.\,\forall a \in \mathsf{In}\,.\,\forall \sigma', \sigma'' \in \mathsf{Out}^*\,.\,(q \xrightarrow{a/\sigma'} q' \wedge\ q \xrightarrow{a/\sigma''} q'') \Rightarrow (q' = q'' \wedge\ \sigma' = \sigma'')$. A deterministic FST is called *minimal* if for any distinct states $q_1, q_2 \in Q$ there exist $\pi \in \mathsf{In}^*$ and $\sigma_1, \sigma_2 \in \mathsf{Out}^*$ such that $q_1 \xrightarrow{\pi/\sigma_1} \wedge\ q_2 \xrightarrow{\pi/\sigma_2} \wedge\ \sigma_1 \neq \sigma_2$.

A FST is a generalization of a Mealy machine. A Mealy machine is a special case of a FST where every output sequence σ consists of a single output symbol.

Two examples of FSTs are given in Figure 1. T is a Mealy machine. It has four states (q_1, q_2, q_3 and q_4), one input (a) and two outputs (0 and 1). T is input-complete, deterministic and minimal. T' is obtained from T by substituting the output symbol 1 by the output sequence 00. T' is input-complete, deterministic and minimal, too.

3 State-identification problems for finite-state transducers

We consider a FST T the model of which is known.[2] The current state of T is not known precisely, but known to be in a set of states $Q_0 \subseteq Q$. Q_0 models the *uncertainty* of the tester at the beginning

[2] For the moment, we make no assumption on T (i.e., T may be non-input-complete, non-deterministic or non-minimal).

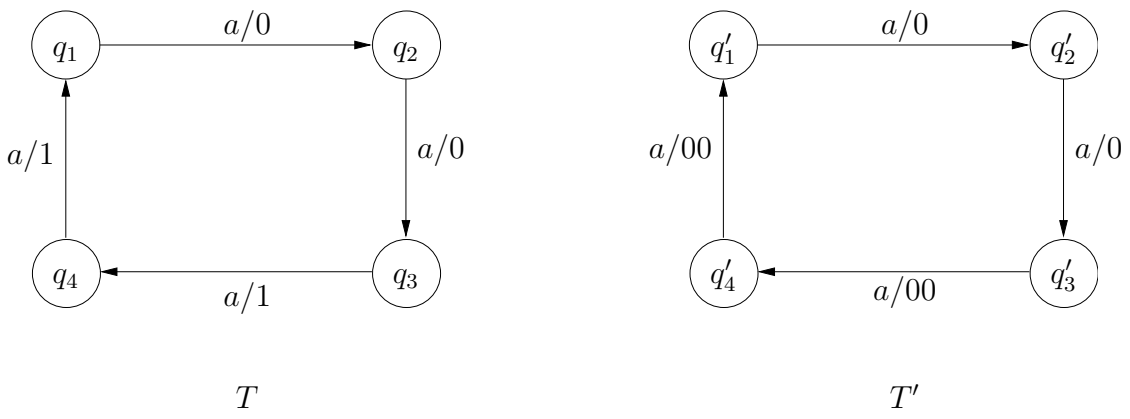

Figure 1: Two FSTs T and T'.

of the experiment. Notice that Q_0 may equal Q, which means the tester has no knowledge of the initial state. The goal is to perform an input/output experiment which allows to deduce the state occupied by T either at the beginning of the experiment (the initial state) or at the end of the experiment (the final state). The two types of experiments are called *homing* and *distinguishing* experiments, respectively.

An input/output experiment consists in applying inputs on T and observing the generated outputs. The experiment may be *preset* or *adaptive* [4].[3] In a preset experiment (PX for short) the input sequence the tester applies is totally known in advance (before the experiment starts). In an adaptive experiment (AX for short) the tester is allowed to decide which inputs to apply depending on the outputs observed so far. Clearly, adaptive experiments are more general.[4]

AXs and PXs are illustrated in Figure 2. An AX is a tree the internal nodes of which are labeled with finite non-empty input sequences $\pi_i \in \mathsf{In}^+$. The edges of the tree are labeled with finite output-sequences $\sigma_i \in \mathsf{Out}^*$. The labels of two edges emanating from the same internal node must be distinct (e.g., in the figure, $\sigma_1 \neq \sigma_2$ and $\sigma_3 \neq \sigma_4$). Each leaf is labeled with state in Q. The AX shown in the figure is to be interpreted as follows:

> Issue the input sequence π_1 and collect the observed output sequence. If the latter equals σ_2 then stop the experiment and declare that the result of the experiment is q_2. Otherwise (i.e., σ_1 is observed), issue the input sequence π_2 and collect the observed output sequence. If the latter equals σ_3 then the result of the experiment is q_2. Otherwise (i.e., σ_4 is observed) the result is q_1.[5]

The definition of AX proposed above is a slight generalization of the standard definition (for instance, found in [10]) in the sense that we allow π_i to contain more than one symbols. As mentioned in the introduction, a definition of an experiment captures a set of *implicit assumptions made on the observational capabilities of the tester*. This is particularly true for adaptive experiments.

[3] Adaptive experiments are called *branching* experiments in [12].

[4] In the literature a distinction is made between *simple* and *multiple* experiments [12, 4]. A multiple experiment can be executed multiple times and the assumption is that the machine is always at the same state at the beginning of each execution: this essentially means there is a special "reset" button which brings the machine back to the same (unknown) initial state at the beginning of each experiment. We only consider simple experiments in this paper.

[5] Depending on whether we are dealing with a distinguishing or a homing experiment, the result will be interpreted differently.

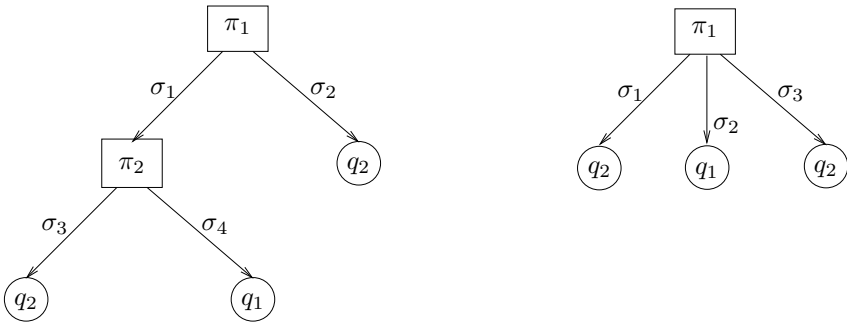

Figure 2: The general scheme of adaptive experiments (left) and preset experiments (right).

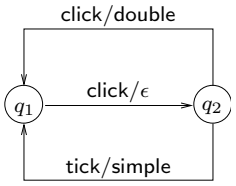

Figure 3: A mouse device producing single and double clicks.

For instance, the AX shown in Figure 2 implicitly assumes that, having issued input sequence π_1 the tester can "stop and wait", until it observes the entire output sequence produced as a result of π_1. This assumption may not be valid in all situations.

For example, the FST in Figure 3 models a mouse device which produces single and double clicks. Here, the tester cannot stop and wait after issuing the first click, because waiting implicitly means that a "timeout" will occur and the mouse will output a single click. Thus, waiting for the timeout must be considered as an input action of the tester.

The above discussion shows that it is probably a good idea to model explicitly the assumptions on the observational capabilities of the tester. We do this by a subset of *synchronizing* input events, $\mathsf{In}_{sync} \subseteq \mathsf{In}$. The tester is allowed to stop and wait after issuing a iff $a \in \mathsf{In}_{sync}$. Formally, each π_i in an internal (non-leaf) node of the AX must end with a symbol in In_{sync}.

In the special case where $\mathsf{In}_{sync} = \mathsf{In}$, the problem of finding an AX for a FST T can be reduced to an equivalent problem of finding an AX for a Mealy machine M: it suffices to associate, for each output sequence ρ appearing in a transition of T, an output symbol o_ρ in M, such that $o_\rho \neq o'_\rho$ iff $\rho \neq \rho'$. Thus, the problem is interesting only when In_{sync} is a strict subset of In.

The situation is somewhat simpler in the case of PXs. Here, the tester is not allowed to make decisions while executing the test, but only at the end. Thus, there is an implicit "end-of-test" action where the tester is allowed to observe the entire output sequence. Here, we will assume that the tester cannot distinguish which part of the output sequence corresponds to which symbol of the input sequence (otherwise, we can reduce the problem to a problem for Mealy machines, as previously). For example, if the tester issues aa to the FST T' of Figure 1 while T' is initially at state q'_2, then the tester will observe 000. The same will happen if T' is initially at state q'_4.

We now define distinguishing and homing adaptive experiments. Let T be a FST with initial uncertainty Q_0 and let X be an AX. For each leaf u of X, π_u denotes the unique input sequence obtained by concatenating the labels of the internal nodes on the path from the root of X to u.

Similarly, σ_u denotes the unique output sequence obtained by concatenating the labels of the edges on this path. Finally, q_u denotes the label of u. X must be *complete* w.r.t. T and Q_0, meaning that it must accept all possible outputs that T can produce when "fed" with the inputs specified in X. Formally, for every $q \in Q_0, q' \in Q, \sigma \in \mathsf{Out}^*$ and every leaf u of X, if $q \xrightarrow{\pi_u/\sigma} q'$ then there exists a leaf v of X such that $\sigma_v = \sigma$.

Definition 1 (Distinguishing and homing experiments) *X is a distinguishing adaptive experiment or DAX (resp., homing adaptive experiment or HAX) w.r.t. T and Q_0 iff (1) X is complete w.r.t. T and Q_0, and (2) for any leaf u of X, $\forall q \in Q_0 \,.\, \forall q' \in Q$: $q \xrightarrow{\pi_u/\sigma_u} q' \;\Rightarrow\; q = q_u$ (resp., $q' = q_u$).*

A distinguishing preset experiment or DPX (resp. homing preset experiment or HPX) is a special case of a DAX (resp. HAX) with a single internal node, the root.

It is well-known that, for deterministic Mealy machines, every distinguishing experiment is also a homing experiment. For example, the sequence aa is both a DPX and a HPX for the machine T shown in Figure 1. This property carries over to deterministic FSTs as well: if T is deterministic then every DAX is a HAX and every DPX is a HPX. However, other properties do not carry over. In particular, it is known that every deterministic and minimal Mealy machine possesses a HPX (although it may not possess a DPX or DAX). We now show that this is not true for deterministic and minimal FSTs.

Consider the FST T' of Figure 1. We claim that T' has no HPX w.r.t. $Q_0 = \{q'_1, q'_2, q'_3, q'_4\}$. To show this, we first argue that none of a, aa, or aaa is a HPX. Then we argue that the effect of input sequence a^n, where $n \geq 4$ is equivalent to the effect of input sequence $a^{n \bmod 4}$, where mod is the *modulo* operator.

a is not a HPX because, on input a and starting from q'_1 and q'_2, respectively, T' produces the same sequence of outputs, 00, and moves to different final states, q'_2 and q'_3, respectively. Thus, a cannot resolve the initial uncertainty between q'_1 and q'_2. Similarly, aa cannot resolve the initial uncertainty between q'_2 and q'_4, and aaa cannot resolve the initial uncertainty between q'_1 and q'_4.

Longer input sequences do not help: a^4 has the same effect as giving no input at all, since it brings the machine to exactly the state it started from and it produces the output sequence 0^6, no matter what the initial state was. Similarly, a^5 has the same effect as a, a^6 has the same effect as aa, and so on.

Notice that, assuming $\mathsf{In}_{sync} = \emptyset$, T' has no HAX either. On the other hand, if we assume $\mathsf{In}_{sync} = \{a\} = \mathsf{In}$ then, as discussed above, the problem can be reduced to the equivalent problem of finding a homing experiment for the Mealy machine T.

4 The successor-tree method

A standard method for solving state-identification problems for a Mealy machine is based on the machine's *successor-tree* [4, 8]. Let us briefly recall this method before studying its application to FSTs. For simplicity, we restrict our discussion to homing preset experiments. It can be generalized to other types of experiments as well.

We first recall the notion of *current uncertainty*. Given a FST T and input sequence π of it, $C(\pi)$ the initial uncertainty of T w.r.t Q_0 is defined as the set of subblocks $B_{\sigma_1}, B_{\sigma_2}, ..., B_{\sigma_N}$, where σ_i are all the possible output sequences T may produce starting from any arbitrary state in Q_0. The subblock B_{σ_i} contains all the states q' for which $\exists q \in Q_0$ such that $q \xrightarrow{\pi/\sigma_i} q'$. That is, if on π

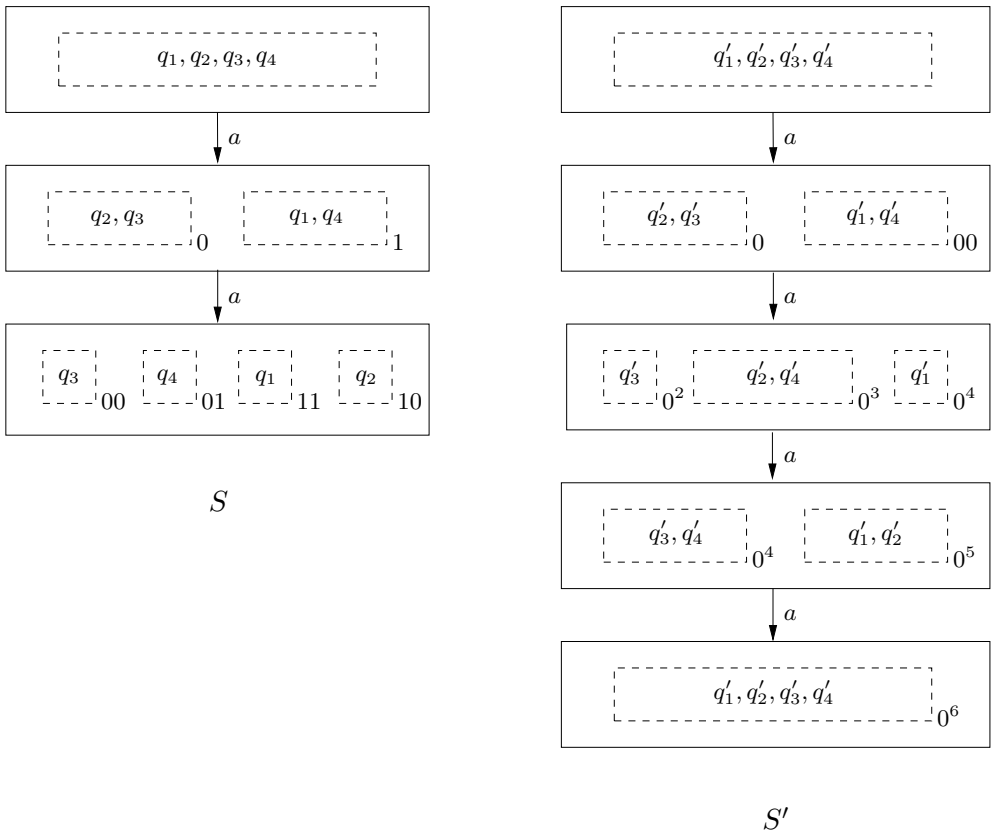

Figure 4: S and S': portions of the successor trees of the FSTs T and T' given in Figure 1, respectively.

the FST T produces σ_i then we are sure that the current state of T is in B_{σ_i}. For example for the FST T given in Figure 1, we have $C(a) = \{\{q_2, q_3\}_0, \{q_4, q_1\}_1\}$.

Clearly, a given input sequence π is a HPX for T iff $C(\pi)$ is made up only by singletons. For instance, aa is a HPX for the FST T given in Figure 1 since we have $C(aa) = \{\{q_3\}_{00}, \{q_4\}_{01}, \{q_1\}_{11}, \{q_2\}_{10}\}$.

In the case of Mealy machines, the classical way for checking whether such a HPX exists or not consists in computing the successor tree of the considered Mealy machine T. The successor tree of T is a (possibly infinite) graph S. For each node v of S, the edges emanating from v are labeled with input symbols: one outgoing edge for each input symbol of the machine. Let π_v denote the input sequence obtained by the concatenation of the labels of the edges on the path from the root of S to v. The node v is labeled with $C(\pi_v)$.

The definition of the successor tree of a FST is given in the same manner as above. For instance, a FST and a portion of its successor tree (up to depth 3) are given in Figure 5. Furthermore, the graphs S and S' given in Figure **??** are portions of the successor trees of the FSTs T and T' given in Figure 1, respectively. [6] S and S' are one-branch trees since T and T' have only one input.

The difference between S and S' starts from depth 3. This difference amounts to the fact that the former is able to distinguish between the two output sequences " $0 \cdot 1$ " and " $1 \cdot 0$ ", however,the

[6] In order not to overload the figure, we write 0^2 instead of 00, 0^3 instead of 000 and so on.

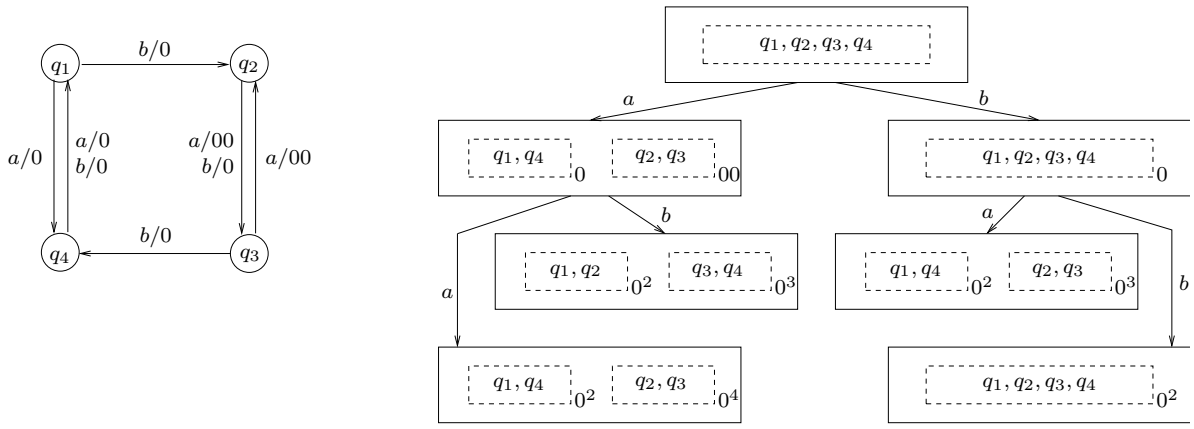

Figure 5: A FST (left) and a portion of its successor tree (right).

latter considers " $0 \cdot 00$ " and " $00 \cdot 0$ " as the same output sequence (" 000 "). So in S, q_2 and q_4 are splitted into $\{q_2\}_{10}$ and $\{q_4\}_{01}$. However in S', q'_2 and q'_4 are grouped together in $\{q'_2, q'_4\}_{000}$.

In the case of Mealy machines, checking the existence of a HPX is decidable due to the fact that two nodes v and v' of the successor tree of the considered machine labeled with $\{B_{\sigma_1}, .., B_{\sigma_N}\}$ and $\{B_{\sigma'_1}, ..., B_{\sigma'_N}\}$ are considered equivalent if $B_{\sigma_1} = B_{\sigma'_1} \wedge ... \wedge B_{\sigma_N} = B_{\sigma'_N}$. Now, since the number of combinations of the form $\{B_1, .., B_N\}$ is finite then a equivalent finite representation of the successor tree of the machine can be given and therefore the problem turns out to be decidable.

Now for the case of FSTs, v and v' are to be considered equivalent if we have $B_{\sigma_1} = B_{\sigma'_1} \wedge ... \wedge B_{\sigma_N} = B_{\sigma'_N}$ and also $\exists \alpha \in \mathsf{Out}^*$ such that $\sigma_1 = \alpha \cdot \sigma'_1 \wedge ... \wedge \sigma_N = \alpha \cdot \sigma'_N$. For example in the successor tree given in Figure 5, the nodes labeled with $\{q_1, q_2, q_3, q_4\}$, $\{q_1, q_2, q_3, q_4\}_0$ and $\{q_1, q_2, q_3, q_4\}_{00}$, respectively, are equivalent. The nodes of the successor tree S' (Figure 4) labeled with $\{q_1, q_2, q_3, q_4\}$ and $\{q_1, q_2, q_3, q_4\}_{0^6}$, respectively, are equivalent. They even allow us to deduce that the corresponding FST has no HPX.

The problem with FSTs is that the number of possible combinations of the form $\{B_{\sigma_1}, .., B_{\sigma_N}\}$ may be infinite, since the sequences σ_i may be arbitrarily long. Thus, we are not always guaranteed to have a finite representation of the considered FST. Consequently, we are not sure whether checking the existence of a HPX for a FST is decidable or not.

5 Decidability for a sub-class of finite-state transducers

We first consider a sub-class of FSTs called *wait-synchronize* FSTs, or WS-FSTs. A WS-FST is a FST which has a special input action $\mathsf{wait} \in \mathsf{In}$ and a special output action $\mathsf{sync} \in \mathsf{Out}$ satisfying the following properties:

P1. wait and sync appear only in transitions of the form $q \xrightarrow{\mathsf{wait}/\sigma\cdot\mathsf{sync}} q'$ where $\sigma \in (\mathsf{Out} \setminus \{\mathsf{sync}\})^*$.

P2. From any state q and any infinite sequence of transitions starting at q, wait eventually appears on this sequence.

Property P1 says that the output symbol sync is generated iff the input symbol wait is applied. It also says that once wait is applied the considered FST T generates an output sequence which ends

with sync and contains no other sync. Property P2 says that from any state q, wait is eventually possible.[7]

Intuitively, wait and sync are to be interpreted as follows: wait models explicitly the waiting of the user of the machine; sync models the "timeout" after which the user can safely assume that all remaining outputs have been generated.

An input sequence $\pi \in (\mathsf{In} \setminus \{\mathsf{wait}\})^* \cdot \mathsf{wait}$ is called an *input-vertebra*. Similarly, an output sequence $\sigma \in (\mathsf{Out} \setminus \{\mathsf{sync}\})^* \cdot \mathsf{sync}$ is called an *output-vertebra*. The sets of input- and output-vertebrae are denoted $\mathsf{Vert}_{\mathsf{in}}$ and. $\mathsf{Vert}_{\mathsf{out}}$, respectively. In view of the discussion in Section 3, we set $\mathsf{In}_{sync} = \{\mathsf{wait}\}$. Thus, the internal nodes of the AX tree are labeled with input-vertebrae and its edges are labeled with output-vertebrae.

We further restrict the class of FSTs we consider by defining *output-bounded* WS-FSTs (OBWS-FSTs). A WS-FST T is output-bounded iff all the possible output-vertebrae that T may produce are of bounded length. Formally:

P3. $\exists n_{max} \,.\, \forall q \in Q \,.\, \forall \pi \in \mathsf{Vert}_{\mathsf{in}} \,.\, \forall \sigma \in \mathsf{Vert}_{\mathsf{out}} \,.\, q \overset{\pi/\sigma}{\longrightarrow\!\!\!\!\!\Rightarrow} |\sigma| \leq n_{max}$.

We now propose a method which permits to solve state-identification problems for a given OBWS-FST T. The method is based on transforming T into a (possibly non-deterministic) Mealy machine M and then applying existing algorithms for state-identification problems on non-deterministic Mealy machines [1].

M has the same set of states as T. The transformation consists of the following steps:

Step 1. We identify the states of T with an incoming edge the input-label of which is wait. These states are called wait-*states*. The latter are these states which can be reached by an input-vertebra.

Step 2. For every state q and every wait-state q_{wait} of T, we compute the language $L^O_{q,q_{\mathsf{wait}}}$ containing all $\sigma \in \mathsf{Vert}_{\mathsf{out}}$ such that $q \overset{\pi/\sigma}{\longrightarrow} q_{\mathsf{wait}}$, for some $\pi \in \mathsf{Vert}_{\mathsf{in}}$. $L^O_{q,q_{\mathsf{wait}}}$ is a finite set of output-vertebrae since T is output-bounded.

Step 3. For each $\sigma \in L^O_{q,q_{\mathsf{wait}}}$, we compute $L^I_{q,q_{\mathsf{wait}},\sigma} = \{\pi \mid \pi \in \mathsf{Vert}_{\mathsf{in}} \text{ and } q \overset{\pi/\sigma}{\longrightarrow} q_{\mathsf{wait}}\}$, the set of input-vertebrae the execution of which may generate σ. $L^I_{q,q_{\mathsf{wait}},\sigma}$ is a regular language since it is induced by a subgraph of T. After computing $L^I_{q,q_{\mathsf{tick}},\mathsf{wait}}$, we add, in M, a new edge from q to q_{wait} labeled with $L^I_{q,q_{\mathsf{wait}},\sigma}/\sigma$. $L^I_{q,q_{\mathsf{wait}},\sigma}$ is called the *language-symbol* of the edge.

Step 4. We collect the language-symbols that appear on the edges of the machine M so far constructed. Let $L_1, ..., L_N$ be the list of these language-symbols. For being able to solve the identification problems, the latter must be disjoint. Only if this holds we have the right to consider two different language-symbols as different input symbols in M. For this purpose, we compute $L'_1, ..., L'_{N'}$, the coarsest partition of $L_1 \cup L_2 \cup ... \cup L_N$ which respects each L_i. Thus, L'_k are pairwise disjoint and each L_i is "split" into a number of L'_k, namely:

$$L_i = L'_{j_1} \cup \cdots \cup L'_{j_i}.$$

Then, we replace each edge $q \overset{L_i/\sigma}{\longrightarrow} q'$ by the edges $q \overset{L'_{j_1}/\sigma}{\longrightarrow} q'$, ..., $q \overset{L'_{j_i}/\sigma}{\longrightarrow} q'$.

[7] Notice that Mealy machines are essentially a special case of WS-FSTs: we can "split" every transition $q \overset{a/x}{\longrightarrow} q'$ of a Mealy machine into two transitions $q \overset{a/x}{\longrightarrow} q'' \overset{\mathsf{wait}/\mathsf{sync}}{\longrightarrow} q'$ and obtain an equivalent, for the purposes of testing, WS-FST.

Checking whether M has a given type of experiment can be done using the algorithms of [1]. These algorithms permit not only to check existence but also to construct an experiment in case it exists. The algorithms are based on the synthesis of *strategies* in *games with incomplete information*. The game is played between the tester who provides the inputs and the system under test who provides the outputs. The strategy of the system corresponds to resolving non-deterministic choices (when such choices exist). The strategy of the tester corresponds to choosing the inputs. The tester has incomplete information because it only observes the outputs, not the current state of the game. Finding preset experiments corresponds to finding a *blindfold* strategy for the tester, that is, a strategy which is totally defined in advance. Finding preset and adaptive experiments is shown in the above paper to be PSPACE-complete and EXPTIME-complete problems, respectively.

It is not difficult to show that T has a DPX (resp., HPX, DAX, HAX) iff M has a DPX (resp., HPX, DAX, HAX). Moreover, the way for constructing an experiment for T given an experiment for M is straightforward.

6 Perspectives

We have presented a framework for state-identification problems for finite-state transducers. A number of open questions remain: decidability in the general case, complexity, properties on the worst-case size of experiments, when the latter exist, and so on. We are currently studying these questions. We are also experimenting with the modeling possibilities of the sub-class identified in this paper, in particular in the context of testing with timing constraints.

References

[1] R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *27th ACM Symposium on Theory of Computing (STOC'95)*, pages 363–372, 1995. 2, 3, 9, 10

[2] C. Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoret. Comp. Sci.*, 5:325–338, 1977. 3

[3] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, 1974. 3

[4] A. Gill. State-identification experiments in finite automata. *Information and Control*, 4:132–154, 1961. 4, 6

[5] S. Ginsburg. On the length of the smallest uniform experiment which distinguishes the terminal states of a machine. *J. Assoc. Comput. Mach.*, 5:266–280, 1958. 3

[6] S. Ginsburg. *An Introduction to Mathematical Machine Theory.* Addison-Wesley, 1962. 3

[7] S. Ginsburg and G.F. Rose. A characterization of machine mappings. *Can. J. Math.*, 18:381–388, 1966. 3

[8] Z. Kohavi. *Switching and finite automata theory, 2nd ed.* McGraw-Hill, 1978. 2, 3, 6

[9] D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3), March 1994. 3

[10] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. *Proceedings of the IEEE*, 84:1090–1126, 1996. 3, 4

[11] M. Mohri. Minimization algorithms for sequential transducers. *Theoret. Comp. Sci.*, 234(1-2):177–201, 2000. 3

[12] E.F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, number 34. Princeton University Press, 1956. 1, 3, 4

[13] J. Oncina, P. Garcia, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5), 1993. 3

[14] C. Reutenauer and M.P. Schützenberger. Minimization of rational word functions. *SIAM Journal on Computing*, 20(4):669–685, 1991. 3

[15] E. Roche and Y. Schabes. *Finite-State Devices for Natural Language Processing*. MIT Press, 1997. 3

[16] M.P. Schützenberger. Sur une variante des fonctions séquentielles. *Theoret. Comp. Sci.*, 4(1), 1977. 3

[17] M. Sokolovskii. Diagnostic experiments with automata. *Kibernetika*, (6):44–49, 1971. In Russian. 3

[18] M. Vasilevskii. Failure diagnosis with automata. *Kibernetika*, (4):98–108, 1973. In Russian. 3

[19] M. Yannakakis and D. Lee. Testing finite state machines. In *23rd ACM Symposium on Theory of Computing*, 1991. 3

[20] M. Yannakakis and D. Lee. Testing finite state machines: fault detection. *Journal of Computer and System Sciences*, 50(2), 1995. 3