State Identification Problems for Timed Automata*

Moez Krichen and Stavros Tripakis

Verimag Centre Equation,
2, avenue de Vignate, 38610 Gières, France
{krichen, tripakis}@imag.fr

Abstract. A well-established theory exists for testing finite state machines. One 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 tests which identify the initial or final state of the machine. In this paper, we study state identification in the context of timed automata which contrary to, say, Mealy or Moore machines, is a suitable model for real-time systems. We are interested in digital-clock tests which have a finite clock precision and are thus implementable. We develop a general technique, based on time-abstracting bisimulation, which reduces the problem to the case of non-deterministic finite-state Mealy machines. We illustrate our technique on a toy example.

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 [9], is *state identification*. We are given a *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* experiments), or verify that the machine is indeed in an assumed-to-be state (*state-verification* experiments), or identify the final state, reached at the end of the experiment (*homing* experiments), or lead the machine to a given state (*synchronizing* experiments), etc.

An extensive theory is available on state identification problems for finite-state machine models such as Mealy or Moore machines (see [8] for an excellent survey). These models are well-suited for some applications (e.g., synchronous

* Work partially supported by CNRS STIC project "CORTOS" and by IST Network of Excellence "ARTIST2".

F. Khendek and R. Dssouli (Eds.): TestCom 2005, LNCS 3502, pp. 175–191, 2005.

circuits) but not for others. In particular, the assumption that inputs and outputs are synchronous makes these models unsuitable when modeling real-time systems, where outputs are produced with variable delays governed by complex timing constraints.

In this paper, we study state identification (in particular, distinguishing and homing) problems in the context of *timed automata* [2]. The latter have been recognized as a useful model for real-time systems. Although some work has been done for this model on problems of conformance testing (e.g., see references in [7]) we are not aware of any previous work on problems of state identification in a real-time setting.

Since timed automata (TA) are based on a *dense-time* semantics, the first choice to make when dealing with testing on such a model is to define the observation capabilities of the tester, in particular in what concerns time. Two types of testers can thus be defined: *analog-clock* testers which can observe real-time precisely; *digital-clock* testers which can only observe the "ticks" of a digital clock (i.e., a counter). Caring about the implementability of our approach, we consider digital-clock testers in this paper. Indeed, analog-clock testers rely on an infinite-precision clock thus are difficult, if not impossible, to implement.

Assuming digital-clock testers has an additional benefit. It opens up the possibility for reducing the problem from the timed to the untimed case. However, carrying out this idea is less obvious than one may think. We summarize the main steps of the procedure in the sequel. The details are given in the main body of the paper.

The first thing to do is to compute the product $A||\mathsf{Tick}$ of the TA under test A with Tick, which is a TA modeling the digital clock of the tester. Tick emits the special output tick and does not synchronize with A except in time. Since the tester does not have access to any other timing information except the number of ticks, it becomes an "untimed controller" of the product TA, call it $A||\mathsf{Tick}$.

Thus, in principle, it seems possible at this point to reduce the problem to a problem of "untimed" testing by working on some kind of *time-abstracting* graph of $A||\mathsf{Tick}$. One choice is the *region graph* [2] but this is obviously to be avoided if we want our method to be tractable. An alternative is to use the *forward reachability graph* used in TA model-checking tools such as Kronos [4]. Unfortunately, this graph does not have the necessary properties for testing purposes. In particular, it is not *pre-stable*, that is, if $S_1 \stackrel{a}{\rightarrow} S_2$ is a transition in the abstract graph then there might be (concrete) states in the abstract state S_1 which have no a-transition in some state in S_2. To see why this is problematic, suppose a is an input: if the tester issues a in the abstract state S_1 then the abstract system will move to S_2; however, the concrete system is not guaranteed to do so. Our choice is, then, to use the *time-abstracting bisimulation* (TAB) quotient graph [11]. This graph has the same properties as the region graph (in particular, pre-stability) and is typically much smaller than the latter. Thus, it presents a good compromise between property preservation and size.

Once we have generated the TAB quotient graph of $A||\mathsf{Tick}$, call it G, we have a finite-state model which can be treated algorithmically. However, G is not a

Mealy machine: it is a labeled transition system (LTS) the transitions of which are labeled with input or output actions, ticks, or τ labels. The latter correspond to either *unobservable* actions of A or time-elapsing transitions abstracted by the bisimulation. Notice that we make no assumption on A (or Tick), in particular, it can be non-deterministic and partially observable. The reason is that we have to confront non-determinism anyway, even if A is deterministic: two distinct output sequences may appear the same to the tester because of its digital clock.

The last step consists in transforming G into a non-deterministic Mealy machine M, on which state-identification problems can be solved using existing techniques [1]. This is an original, to our knowledge, transformation technique which is general enough to be used for any finite LTS provided it satisfies some properties on boundedness of number of outputs (see below). Thus, the technique can be useful in an "untimed" context to reduce testing problems from asynchronous LTS models to synchronous Mealy machines.

In a nutshell, the transformation is as follows. For every pair of nodes v_1, v_2 of G, we compute the language L_{v_1,v_2} of *vertebrae* linking v_1 and v_2. A vertebra is a finite word ending with a tick symbol and containing a single tick. L_{v_1,v_2} is a regular language. We then compute $L^O_{v_1,v_2}$, the *projection* of L_{v_1,v_2} into *output vertebrae*, that is, vertebrae containing only outputs and tick. Using the hypothesis that A is *bounded output* (i.e., can only emit a bounded number of outputs in a bounded amount of time) we can show that $L^O_{v_1,v_2}$ is a finite language. For each $\sigma \in L^O_{v_1,v_2}$, we compute $L^I_{v_1,v_2,\sigma}$, the projected language of input vertebrae corresponding to σ. Finally, for each such σ, we construct a transition in M of the form $v_1 \xrightarrow{L/\sigma} v_2$, where L is an appropriate regular language derived from $L^I_{v_1,v_2,\sigma}$.

Intuitively, an input vertebra like $a \cdot b \cdot \mathsf{tick}$ corresponds to a basic command of the tester which is to issue input a followed by b and then wait until the next tick before it proceeds. An output vertebra like $c \cdot \mathsf{tick}$ corresponds to what the tester observes after executing the input vertebra. Notice that the lengths of the two need not be the same, although both end with a unique tick, since, according to the interpretation above, tick can be seen both as an input and output. $v_1 \xrightarrow{L/\sigma} v_2$ can be seen as a *symbolic* transition: L is the input symbol and σ is the output symbol. To exercise this transition, a tester chooses some input vertebra in L and then may observe the output vertebra σ. Notice that M is non-deterministic, thus, the same input can also result to another output σ'. The correctness of the method lies on the fact that input and output vertebrae end with their unique tick symbols. Thus, there is no danger that concatenating vertebrae may result in ambiguity for the tester. In particular, if σ_1, σ_2 are two vertebrae such that $\sigma_1 \neq \sigma_2$ then for any vertebrae σ'_1, σ'_2, $\sigma_1 \cdot \sigma'_1 \neq \sigma_2 \cdot \sigma'_2$.

The rest of this paper is organized as follows. In Section 2 we recall our model of timed automata with input, output and unobservable actions. In Section 3 we define the various state-identification problems. In Section 4 we recall the time-abstracting quotient graph and identify its properties of interest for our purposes. In Section 5 we show how to transform this graph to a non-

deterministic Mealy machine and reduce the problems from the timed to the untimed case. Section 6 summarizes the paper and gives some directions for future work.

2 The Model

The basic model is timed automata with inputs and outputs (TAIO) as defined in [7]. These are timed automata with *deadlines* to capture urgency [10, 3] and edges labeled by an *input action* in a finite set $\mathsf{Act}_{\mathsf{in}} = \{a, b, \cdots\}$, an *output action* in a finite set $\mathsf{Act}_{\mathsf{out}} = \{v, w, \cdots\}$ or an *unobservable action* $\tau \notin \mathsf{Act}_{\mathsf{in}} \cup \mathsf{Act}_{\mathsf{out}}$.

Let R be the set of non-negative reals. Given a finite set of *actions* Act, the set $(\mathsf{Act} \cup \mathsf{R})^*$ of all finite *real-time sequences* over Act will be denoted $\mathsf{RT}(\mathsf{Act})$. The *length* of a sequence ρ is denoted $|\rho|$. $\epsilon \in \mathsf{RT}(\mathsf{Act})$ is the empty sequence. Given $\mathsf{Act}' \subseteq \mathsf{Act}$ and $\rho \in \mathsf{RT}(\mathsf{Act})$, $P_{\mathsf{Act}'}(\rho)$ denotes the *projection* of ρ to Act', obtained by "erasing" from ρ all actions not in Act' and all delays. For example, if $\mathsf{Act} = \{a, b\}$, $\mathsf{Act}' = \{a\}$ and $\rho = a\,1\,b\,2\,a\,3$, then $P_{\mathsf{Act}'}(\rho) = a\,a$. $P_{\mathsf{Act}' \cup \mathsf{R}}(\rho)$ denotes the projection of ρ to $\mathsf{Act}' \cup \mathsf{R}$, obtained by "erasing" from ρ all actions not in Act' (but not delays). For example, $P_{\mathsf{Act}' \cup \mathsf{R}}(\rho) = a\,3\,a\,3$. The time spent in a sequence ρ, denoted $\mathsf{time}(\rho)$ is the sum of all delays in ρ, for example, $\mathsf{time}(\epsilon) = 0$ and $\mathsf{time}(a\,1\,b\,0.5) = 1.5$.

A *timed automaton over* Act is a tuple $(Q, X, \mathsf{Act}, \mathsf{E})$ where Q is a finite set of *locations*; X is a finite set of *clocks*; E is a finite set of *edges*. Each edge is a tuple (q, q', ψ, r, d, a), where $q, q' \in Q$ are the source and destination locations; ψ is the *guard*, a conjunction of constraints of the form $x \# c$, where $x \in X$, c is an integer constant and $\# \in \{<, \leq, =, \geq, >\}$; $r \subseteq X$ is the set of clocks to be *reset*; $d \in \{\mathsf{lazy}, \mathsf{delayable}, \mathsf{eager}\}$ is the *deadline*; and $a \in \mathsf{Act}$ is the action. Intuitively, a lazy deadline imposes no urgency on the transition; delayable means the transition, once enabled, must be taken before it becomes disabled; eager means the transition must be taken as soon as it becomes enabled. We will not allow eager edges with guards of the form $x > c$.

A TA A defines an infinite labeled transition system (LTS) the states of which are pairs $s = (q, v) \in Q \times \mathsf{R}^X$, where $q \in Q$ is a location and $v : X \to \mathsf{R}$ is a clock *valuation*. Given state $s = (q, v)$ and clock x, we write $x(s)$ to denote the value of x at s, i.e., $v(x)$. $\mathbf{0}$ is the valuation assigning 0 to every clock of A. S_A is the set of all states. There are two types of transitions, discrete and timed. Discrete transitions are of the form $s = (q, v) \xrightarrow{a} s' = (q', v')$, where $a \in \mathsf{Act}$ and there is an edge $e = (q, q', \psi, r, d, a)$, such that v satisfies ψ and v' is obtained by resetting to zero all clocks in r and leaving the others unchanged. We say that e is enabled at s and write $s \models e$ (or $s \models \psi$). Timed transitions are of the form $(q, v) \xrightarrow{t} (q, v + t)$, where $t \in \mathsf{R}, t > 0$ and there is no edge (q, q'', ψ, r, d, a), such that: either $d = \mathsf{delayable}$ and there exist $0 \leq t_1 < t_2 \leq t$ such that $v + t_1 \models \psi$ and $v + t_2 \not\models \psi$; or $d = \mathsf{eager}$ and there exists $0 \leq t_1 < t$ such that $v + t_1 \models \psi$. We use notation such as $s \xrightarrow{a}$, $s \not\xrightarrow{a}$, ..., to denote that there exists s' such that $s \xrightarrow{a} s'$, there is no such s', and so on. This notation naturally

extends to timed sequences. For example, $s \xrightarrow{a1b} s'$ if there exist s_1, s_2 such that $s \xrightarrow{a} s_1 \xrightarrow{1} s_2 \xrightarrow{b} s'$.

A *timed automaton with inputs and outputs* (TAIO) is a timed automaton over $\mathsf{Act}_\tau = \mathsf{Act} \cup \{\tau\}$. A TAIO is called *observable* if none of its edges is labeled by τ. A TAIO A is called *input-complete* if it can accept any input at any state: $\forall s \in S_A \,.\, \forall a \in \mathsf{Act_{in}} \,.\, s \xrightarrow{a}$. It is called *deterministic* if $\forall s, s', s'' \in S_A \,.\, \forall a \in \mathsf{Act}_\tau \,.\, s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow s' = s''$. It is called *non-blocking* if

$$\forall s \in S_A \,.\, \forall t \in \mathsf{R} \,.\, \exists \rho \in \mathsf{RT}(\mathsf{Act_{out}} \cup \{\tau\}) \,.\, \mathsf{time}(\rho) = t \wedge s \xrightarrow{\rho} . \tag{1}$$

The non-blocking property states that at any state, A can let time pass forever, even if it does not receive any input. This is a sanity property which ensures that a TAIO does not "force" its environment to provide an input by blocking time.

A is called *output-bounded* if there is a bound on the number of outputs A can produce in a bounded amount of time (say, one time unit). Formally:

$$\exists n \,.\, \forall s \in S_A \,.\, \forall \rho \in \mathsf{RT}(\mathsf{Act}_\tau) \,.\, (s \xrightarrow{\rho} \wedge \mathsf{time}(\rho) = 1) \Rightarrow |P_{\mathsf{Act_{out}}}(\rho)| \leq n. \tag{2}$$

An example of a TAIO is shown in Figure 1. This TAIO has three locations (q_1, q_2 and q_3), one input (click), two outputs (simple and double) and one clock (x). It models a mouse which produces a double-click when the button is pressed twice (or more) in one time unit, a simple-click otherwise. We will use this automaton as a running example in the rest of the paper. We annotate actions with ? and ! to denote inputs and outputs, respectively. Unless otherwise noted, deadlines are lazy.

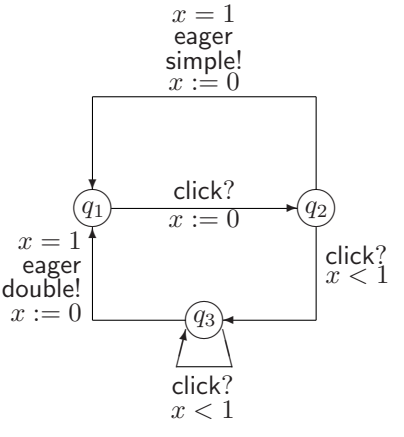

Fig. 1. An example of a TAIO: the simple- and double-click mouse

3 State-Identification Problems on Timed Automata

We consider a TAIO A which is non-blocking and output-bounded (but possibly non-deterministic, partially-observable or non input-complete) and the current state of which is unknown. We wish to perform an input/output experiment from which we can deduce either the initial state (the state A was occupying at the beginning of the experiment − the *distinguishing* problem) or the final state (the state A is occupying at the end of the experiment − the *homing* problem).

An input/output experiment consists in applying inputs on A and observing the generated outputs. The experiment may be *preset* or *adaptive* [5].[1] In a preset experiment the input sequence the tester applies is totally known in advance (before the experiment starts). In an adaptive experiment the tester is allowed to decide which inputs to apply depending on the outputs observed so far. Clearly, adaptive experiments are more general. While a preset experiment can simply be modeled as an input sequence, an adaptive experiment needs to be modeled as a decision tree (e.g., see Figure 3).[2]

In our case, the tester is "timed": it observes not only the outputs of the machine under test but also the time when these outputs occur. In practice, it is not possible to observe time in an infinitely-precise way, due to the fact that the tester has access only to a digital clock (i.e., a discrete counter updated by some physical process). In this paper, we make the assumption that the clock of the tester can be modeled by a timed automaton called Tick. Examples of Tick automata are given in Figure 2. Tick is a TAIO with a single action $\mathsf{tick} \notin \mathsf{Act_{in}} \cup \mathsf{Act_{out}} \cup \{\tau\}$ and no inputs. Tick must be non-blocking and ensure that tick always eventually occurs. Our method works for any such Tick model.

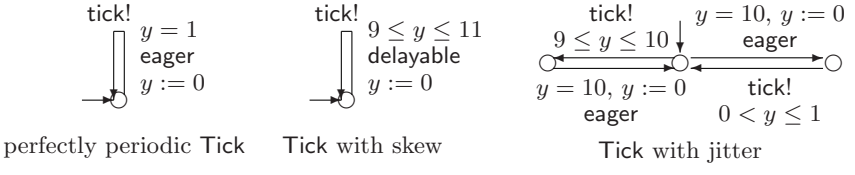

Fig. 2. Possible tester clock models

The initial uncertainty of the tester is modeled by a set of states $S_0 \subseteq S_A$. In other words, we assume that A is initially in some state in S_0. Notice that

[1] Adaptive experiments are called *branching* experiments in [9].

[2] In the literature a distinction is made between *simple* and *multiple* experiments [9, 5]. 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, since they assume less power on the tester side.

S_0 may equal S_A, which means we have no knowledge of the initial state. We are also given m pairwise disjoint subsets of S_A, $C_1, ..., C_m$. In the case of the distinguishing problem, $C_1, ..., C_m$ form a *partition* of S_0 (thus, $S_0 = \bigcup_i C_i$). This partition models our requirements from the tester: we want the tester to tell us, at the end of the experiment, in which of the m subsets A was at the beginning of the experiment. In the case of the homing problem, $C_1, ..., C_m$ form a partition of S_A (thus, $S_A = \bigcup_i C_i$). Here, we want the tester to tell us in which of the m subsets A is at the end of the experiment. For example, we might associate one set C_i with each location q_i of A, meaning we want to know the final location.

A *vertebra* is an element of $\mathsf{Vert} = (\mathsf{Act}_{\mathsf{in}} \cup \mathsf{Act}_{\mathsf{out}})^* \cdot \{\mathsf{tick}\}$. An *input-vertebra* (respectively, *output-vertebra*) is an element of $\mathsf{Vert}_{\mathsf{in}} = (\mathsf{Act}_{\mathsf{in}})^* \cdot \{\mathsf{tick}\}$ (respectively, $\mathsf{Vert}_{\mathsf{out}} = (\mathsf{Act}_{\mathsf{out}})^* \cdot \{\mathsf{tick}\}$). At each vertebra corresponds a unique input-vertebra and a unique output-vertebra which can be obtained by projection. For instance, $a? \cdot v! \cdot b? \cdot v! \cdot w! \cdot \mathsf{tick}$ is a vertebra with corresponding input-vertebra $a? \cdot b? \cdot \mathsf{tick}$ and output-vertebra $v! \cdot v! \cdot w! \cdot \mathsf{tick}$.

A *digital preset experiment* (PX for short) is a finite sequence $\pi \in (\mathsf{Vert}_{\mathsf{in}})^*$, for example

$$\pi = a \cdot \mathsf{tick} \cdot b \cdot c \cdot \mathsf{tick}.$$

This experiment is to be interpreted as follows:

> Issue input a; wait until the next clock tick occurs; issue input b, then input c; wait until the next clock tick occurs; collect the observed output.

This interpretation assumes that the tester has enough time to issue the entire sequence of input actions appearing between two successive ticks before the next tick is received. However, this is not a restrictive assumption: as we make no assumption on input-completeness on A, assumptions on A's environment can be *modeled directly* within A. In particular, timing constraints on the tester, such as how much time it takes to issue an input can be modeled this way. As we shall see later, the tester is not allowed to issue an input which may not be accepted by A, thus, must obey the modeled timing restrictions.

A *digital adaptive experiment* (AX for short) is defined as a finite decision tree like the ones shown in Figure 3. Each internal node of the tree is labeled with an input-vertebra. Each edge is labeled with an output-vertebra: the labels of two edges emanating from the same internal node must be distinct. Each leaf is labeled with an element from $\{C_1, ..., C_m\}$. The AX to the left of the figure is to be interpreted as follows:

> Issue input a, issue input b, wait until the next tick and collect the observed output sequence. If the latter equals $v \cdot w \cdot \mathsf{tick}$ then stop the experiment and declare that the result of the experiment is C_2. Otherwise (i.e., $v \cdot \mathsf{tick}$ is observed), issue input a twice, wait until the next tick and collect the observed output sequence. If the latter equals $v \cdot \mathsf{tick}$ then the

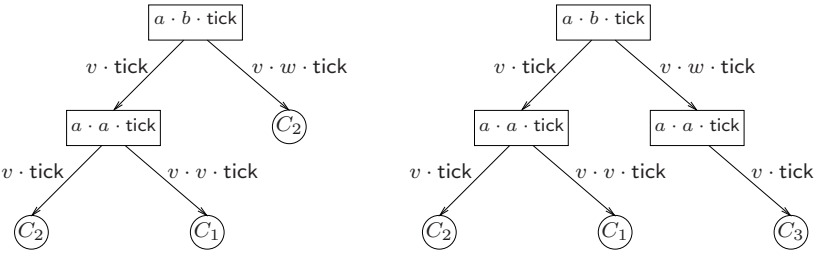

Fig. 3. Two digital experiments: adaptive (left) and preset (right)

result of the experiment is C_2. Otherwise (i.e., $v \cdot v \cdot \mathsf{tick}$ is observed) the result is C_1.[3]

Notice that we allow decisions to be taken in an AX only after ticks. Indeed, this is the only choice that makes sense implementation-wise, because this is the point where the inputs and outputs of the tester *synchronize*. Suppose we allowed the AX to make a decision after every input it issues. This would give a tree where an internal node could be labeled, say, by $a \in \mathsf{Act_{in}}$. How would such an experiment be interpreted? It is not legal to interpret it as: "Issue input a and observe the output. If the latter is b then ... else ...". This is because the output cannot always be observed immediately after an input is given, but only after some time. Indeed, for some inputs, there might be no output at all until the next tick. Thus, it is natural to interpret the above experiment as: "Issue input a and wait for the next output. If the latter is ...". But waiting for the next output is equivalent to waiting for the next tick and then observing the output, since the tester does not know how much it has to wait. This is precisely the same as labeling the internal node by $a \cdot \mathsf{tick}$, which brings us to our case where nodes are labeled with input-vertebrae.

Another thing to point out on AX is the fact that the branches from an internal node do not cover all possible outputs. Indeed, they cannot, since the AX needs to be finite and the number of output-vertebrae is infinite. However, as we shall see, for an output-bounded TAIO, the above number turns out to be finite. We shall use this hypothesis in the following sections. In terms of execution of an AX, if the tester observes an output sequence which is not in the set of possible outputs in the current node of the decision tree, then the tester declares the system under test or the tester's clock as *non-conforming* to their respective models.

Notice that a PX is a special type of an AX, where the inputs given do not depend on the outputs observed during the experiment. Thus, an AX T is a PX if all leaves of T are at the same depth and all internal nodes of T which are at the same depth are labeled with the same input-vertebra. For example, the AX to the right of Figure 3 is preset.

[3] Depending on whether we are dealing with the distinguishing problem or the homing problem, the result of the experiment is interpreted differently, see below.

Before giving the definitions of distinguishing and homing sequences, we need to make the link between the possible observable input- and output-vertebrae and the real-time traces of the model. For that, we consider $A^{\mathsf{tick}} = A||\mathsf{Tick}$ the product of A and Tick. A *timed-vertebra* is an element of $\mathsf{Vert_{RT}} = \mathsf{RT}(\mathsf{Act}) \cdot \{\mathsf{tick}\}$, for example, $\rho = a \cdot 0.4 \cdot v \cdot \tau \cdot 0.6 \cdot \mathsf{tick}$. Let $\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}} = \mathsf{Act}_{\mathsf{in}} \cup \{\mathsf{tick}\}$ and $\mathsf{Act}_{\mathsf{out}}^{\mathsf{tick}} = \mathsf{Act}_{\mathsf{out}} \cup \{\mathsf{tick}\}$. To each $\rho \in (\mathsf{Vert_{RT}})^*$ correspond the unique sequences $\pi = P_{\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}}}(\rho) \in (\mathsf{Vert_{in}})^*$ and $\sigma = P_{\mathsf{Act}_{\mathsf{out}}^{\mathsf{tick}}}(\rho) \in (\mathsf{Vert_{out}})^*$. For example, ρ given above matches $\pi = a \cdot \mathsf{tick}$ and $\sigma = v \cdot \mathsf{tick}$. Intuitively, if ρ matches π and σ then ρ is a possible behavior of A^{tick} for which the latter can produce σ when "fed" with π.

For some set of states $S \subseteq S_A$ and an input-vertebra sequence $\pi \in (\mathsf{Vert_{in}})^*$ we say that π is *accepted* by S if $\forall s \in S \cdot \exists \rho \in (\mathsf{Vert_{RT}})^* \cdot s \xrightarrow{\rho} \wedge\, \pi = P_{\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}}}(\rho)$. The set of output-vertebra sequences that can be observed starting from some state in S, due to the execution of π is $outputs(S, \pi) = \{\sigma \in (\mathsf{Vert_{out}})^* \mid \exists s \in S \cdot \exists \rho \in \mathsf{Vert_{RT}} \cdot s \xrightarrow{\rho} \wedge \pi = P_{\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}}}(\rho) \wedge \sigma = P_{\mathsf{Act}_{\mathsf{out}}^{\mathsf{tick}}}(\rho)\}$. Moreover for some output-vertebra sequence $\sigma \in (\mathsf{Vert_{out}})^*$, we introduce the two following sets of states $init(S, \pi, \sigma) = \{s \in S \mid \exists \rho \in \mathsf{Vert_{RT}} \cdot s \xrightarrow{\rho} \wedge \pi = P_{\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}}}(\rho) \wedge \sigma = P_{\mathsf{Act}_{\mathsf{out}}^{\mathsf{tick}}}(\rho)\}$ and $succ(S, \pi, \sigma) = \{s' \in S_A \mid \exists s \in S \cdot \exists \rho \in \mathsf{Vert_{RT}} \cdot s \xrightarrow{\rho} s' \wedge \pi = P_{\mathsf{Act}_{\mathsf{in}}^{\mathsf{tick}}}(\rho) \wedge \sigma = P_{\mathsf{Act}_{\mathsf{out}}^{\mathsf{tick}}}(\rho)\}$. Intuitively, $init(S, \pi, \sigma)$ corresponds to the subset of states of S from which it is possible to observe σ after applying π and $succ(S, \pi, \sigma)$ the subset of states of S_A to which it is possible to move after applying π and observing σ.

Definition 1 (Valid experiments). *Let T be an AX, A a TAIO and $S_0 \subseteq S_A$. T is said to be valid w.r.t. A and S_0 if for each node u of T it is possible to assign a set of states $S_u \subseteq S_A$ such that the following hold:*

- *for r, the root of T, we have $S_r = S_0$;*
- *for each internal node u of T, if π is the input-vertebra label of u, then:*
  - *π is accepted by S_u;*
  - *for each $\sigma \in outputs(S_u, \pi)$, there exists an outgoing edge from u labeled with σ; furthermore, u has as many outgoing edges as the number of elements of $outputs(S_u, \pi)$;*
  - *if $u \xrightarrow{\sigma} u'$ is an edge of T then $S_{u'} = succ(S_u, \pi, \sigma)$.* □

Validity guarantees that, at each step of its execution, the input provided by the experiment is accepted by the current state of the machine, no matter what this state is. Validity also ensures that any output the machine may produce is taken into account in the experiment.

Let T be an AX. For each leaf u of T, π_u denotes the unique sequence of input-vertebrae obtained by concatenating the labels of the internal nodes on the path from the root of T to u. Similarly, σ_u denotes the unique sequence of output-vertebrae obtained by concatenating the labels of the edges on this path. Finally, C_u denotes the label of u (i.e., $C_u \in \{C_1, ..., C_m\}$).

Definition 2 (Distinguishing and homing experiments). *An experiment T is* distinguishing *(respectively,* homing*) for A w.r.t.* Tick*, S_0 and $\{C_1, ..., C_m\}$*

iff T is valid w.r.t. A and S_0 and for any leaf u of T we have: $succ(S_0, \pi_u, \sigma_u) \subseteq C_u$ (respectively, $init(S_0, \pi_u, \sigma_u) \subseteq C_u$). □

We use abbreviations DAX, DPX, HAX and HPX for distinguishing or homing, adaptive or preset experiments.

The objective of this paper is to develop algorithms which, given A, Tick, S_0 and $\{C_1, ..., C_m\}$, check whether there exists a DPX, HPX, DAX, or HAX and if so construct one. It can be easily shown that for any type of experiment, a solution does not always exist. This is no surprise, since A is generally partially observable and non-deterministic. But also because the tester only has limited observation capabilities regarding time (i.e., a digital-clock). Notice that even in the case of finite Mealy machines, non-determinism implies that solutions do not always exist for any of these experiments [1]. On the other hand, in the case of deterministic Mealy machines, a homing (preset) experiment always exists, whereas distinguishing experiments may or may not exist [9, 8]. Also note that there are cases where an adaptive experiment exists whereas no preset experiment exists.

4 The Time-Abstracting Bisimulation Quotient Graph

The first step toward solving the state-identification problems defined in the previous section is to generate the *quotient graph* G of the product A^{tick} with respect to the *time-abstracting bisimulation* (TAB). Due to space limitations, we will not define what a TAB is, nor show how to construct G. These topics are presented in detail in [11]. Here, we only recall the basic properties of G which are relevant for the purposes of this paper. Readers not familiar with TABs may think of G as the *region graph* of A^{tick}. Indeed, the latter is in fact a TAB quotient graph, but not the coarsest possible in general.

G is a finite graph. The edges of G are labeled either with some $a \in \mathsf{Act_{in}} \cup \mathsf{Act_{out}} \cup \{\mathsf{tick}, \tau\}$ (corresponding to the discrete transitions of A^{tick}), or with ϵ (corresponding to the passage of time). For our purposes, both τ and ϵ transitions model events which are unobservable to the tester. Thus, we rename all ϵ transitions into τ transitions. From now on, we assume that G has been transformed in that way. That is, the set of labels of G is $\Sigma = \mathsf{Act_{in}} \cup \mathsf{Act_{out}} \cup \{\mathsf{tick}, \tau\}$.

Every node v of G corresponds to a set of states of A^{tick} and consequently to a set of states of A, S_v. We assume that G *respects* all sets $S_0, C_1, ..., C_m$. This means that either $S_v \subseteq S_0$ or $S_v \cap S_0 = \emptyset$ and similarly for every C_i. Constructing G in order to respect such subsets of the state space is not a problem (see [11] for details).

Finite paths of G define sequences of symbols which are in Σ^*. In particular, a *discrete-vertebra* is an element of $\mathsf{Vert_{disc}} = (\Sigma \setminus \{\mathsf{tick}\})^* \cdot \{\mathsf{tick}\}$. As in the TAIO case, we make the link between these discrete-vertebrae and the corresponding observable input- and output-vertebrae. For $\rho \in (\mathsf{Vert_{disc}})^*$, $\pi \in (\mathsf{Vert_{in}})^*$ and $\sigma \in (\mathsf{Vert_{out}})^*$, we say that ρ matches π (resp. σ) if the projection of ρ to $\mathsf{Act_{in}^{tick}}$ (resp. $\mathsf{Act_{out}^{tick}}$) equals π (resp. σ).

The sanity properties of A and Tick induce similar properties on G. G is *non-blocking* in the sense that for any node v of G there exists a node v' of G and a sequence $\rho \in \mathsf{Vert}_{\mathsf{disc}}$ such that $v \xrightarrow{\rho} v'$ is a possible path in G and ρ matches the input-vertebrae tick (i.e., if no input is given, time will elapse). An equally important property is that G is also *output-bounded* in the sense that the output-vertebrae that G can produce are of bounded length. In other words, there exists n such that for any discrete-vertebra ρ of G, the length of the output-vertebra corresponding to ρ is at most n.

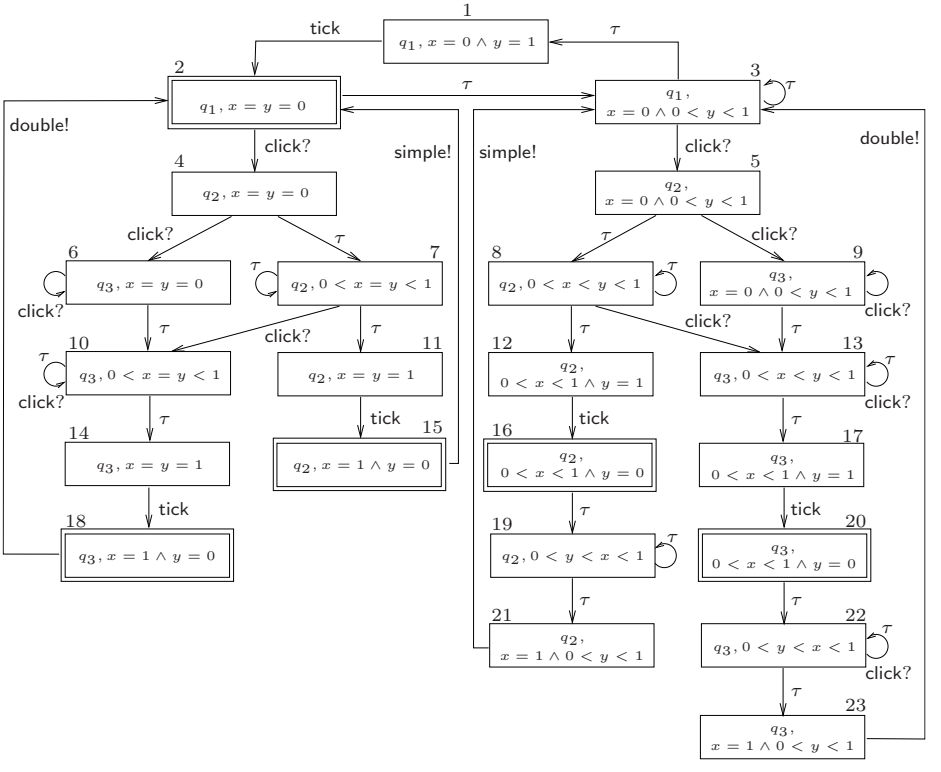

Fig. 4. A time-abstracting bisimulation quotient graph

The graph G shown in Figure 4 is the TAB quotient graph of the product of the TAIO of Figure 1 and the left-most Tick automaton of Figure 2. [4] The nodes of G are numbered from 1 to 23. G is made up of three subblocks:

– The cycle made by nodes 1, 2 and 3 models the behavior of the system when no stimuli are received from the external environment (i.e., only time elapses).

[4] Notice that since the automaton of Figure 1 is not input-complete, click? actions are not allowed at every node of the quotient graph.

- The subgraph of G induced by nodes $2, 4, 6, 7, 10, 11, 14, 15$ and 18 models the behavior of the system when a first click and the tick actions happen simultaneously.
- The second subgraph of G induced by the rest of the nodes models the behavior of the system when the first click and the tick actions happen at different times.

G is not a Mealy machine, thus, existing methods for solving state identification problems [8] do not apply. Indeed, in Mealy machines inputs and outputs are *synchronous* whereas in G they are inherently *asynchronous*: an input may result in some output later in time, or even not at all; an output may be emitted without any explicit input but simply with the passage of time; a single input may produce more than one outputs or more than one inputs may be necessary to produce an output; and so on. This motivates the next section, which proposes a transformation of G to a Mealy machine M capturing all necessary information in order to solve the problems of the previous section. Notice that M is a non-deterministic machine, that is, a given input may result in more that one outputs and/or lead to more that one states. This is to be expected, as mentioned above. Still, input and output symbols in M are synchronous, which allows us to use existing methods on non-deterministic such machines [1].

5 Transformation to a Non-deterministic Mealy Machine and Reduction

As explained in Section 4 the main objective of the transformation is to remove the asynchronism between inputs and outputs in G. To do this, we observe that the basic external stimuli in G are input-vertebrae. Thus, it should suffice to consider the way in which G behaves w.r.t. elements in $\mathsf{Vert}_{\mathsf{in}}$. For this we need to identify the response of G w.r.t. any possible input-vertebra which is "accepted" by the current considered node. An input-vertebra π is accepted by node v if there exists a discrete-vertebra ρ such that $v \xrightarrow{\rho}$ and ρ matches π.

Consequently, a first idea is to transform G into a Mealy machine M with the same set of nodes as G and label the edges of M with pairs $(\pi, \sigma) \in \mathsf{Vert}_{\mathsf{in}} \times \mathsf{Vert}_{\mathsf{out}}$. Formalizing this, we get that for any two nodes v, v' of G and any pair $(\pi, \sigma) \in \mathsf{Vert}_{\mathsf{in}} \times \mathsf{Vert}_{\mathsf{out}}$, we add an edge $v \xrightarrow{\pi/\sigma} v'$ in M iff there exists $\rho \in \mathsf{Vert}_{\mathsf{disc}}$ such that

$$v \xrightarrow{\rho} v' \text{ is a path of } G \text{ and } \rho \text{ matches both } \pi \text{ and } \sigma.$$

For instance, the Mealy machine deduced from the graph shown in Figure 4 has an edge from node 2 to node 18 labeled with $\mathsf{click} \cdot \mathsf{click} \cdot \mathsf{tick}/\mathsf{tick}$ and another edge from 2 to 15 labeled with $\mathsf{click} \cdot \mathsf{tick}/\mathsf{tick}$.

The problem with the above definition is that some nodes of M may have an infinite number of outgoing edges, since the number of paths $v \xrightarrow{\rho} v'$ is a-priori unbounded. For instance, in the preceding example, we need to draw an

edge from node 2 to node 18 for each input-vertebrae which is in $\mathsf{click} \cdot \mathsf{click} \cdot \mathsf{click}^* \cdot \mathsf{tick}$. Observe, however, that any of these input-vertebrae produces the same output-vertebra, namely, tick. Thus, we can remedy the above problem by grouping all input-vertebrae together and representing them *symbolically* in a single transition. More precisely, we will only add a single edge $2 \stackrel{L/\mathsf{tick}}{\rightarrow} 18$, where L is the regular language $\mathsf{click} \cdot \mathsf{click} \cdot \mathsf{click}^* \cdot \mathsf{tick}$.

The method we propose consists of the following steps:

Step 1 We identify the nodes of G with an incoming edge labeled with tick. These nodes are called tick-*nodes*. The latter are these nodes which can be reached by an input-vertebra. In Figure 4, the tick-nodes are drawn with double rectangles.

Step 2 For every node v and every tick-node v_{tick} of G, we compute the language $L_{v,v_{\mathsf{tick}}}$ containing all $\rho \in \mathsf{Vert}_{\mathsf{disc}}$ such that $v \stackrel{\rho}{\rightarrow} v_{\mathsf{tick}}$. $L_{v,v_{\mathsf{tick}}}$ is a regular language since it is induced by a subgraph of G.

Step 3 For each v and v_{tick}, we compute $L^O_{v,v_{\mathsf{tick}}} = \{P_{\mathsf{Act}^{\mathsf{tick}}_{\mathsf{out}}}(\rho) \mid \rho \in L_{v,v_{\mathsf{tick}}}\}$, the projection of $L_{v,v_{\mathsf{tick}}}$ to the set of outputs and tick actions. $L^O_{v,v_{\mathsf{tick}}}$ is a set of output-vertebrae. Since G is output-bounded, $L^O_{v,v_{\mathsf{tick}}}$ is a finite set.

Step 4 For each $\sigma \in L^O_{v,v_{\mathsf{tick}}}$, we compute $L^I_{v,v_{\mathsf{tick}},\sigma} = \{\pi \mid \exists \rho \in L_{v,v_{\mathsf{tick}}}$ such that ρ matches both π and $\sigma\}$, the set of input-vertebrae the execution of which may generate σ. $L^I_{v,v_{\mathsf{tick}},\sigma}$ can be defined equivalently as

$$L^I_{v,v_{\mathsf{tick}},\sigma} = P_{\mathsf{Act}^{\mathsf{tick}}_{\mathsf{in}}}(P^{-1}(\sigma) \cap L_{v,v_{\mathsf{tick}}})$$

where $P^{-1}(\cdot)$ denotes the *inverse projection* function. Since all the operations in the right-hand side of the above formula preserve regular languages, $L^I_{v,v_{\mathsf{tick}},\sigma}$ is a regular language.

After computing $L^I_{v,v_{\mathsf{tick}},\sigma}$, we add in M a new edge from v to v_{tick} labeled with $L^I_{v,v_{\mathsf{tick}},\sigma}/\sigma$.

At this point, we have obtained a finite, non-deterministic Mealy machine which has the same nodes as G and the edges of which are labeled with pairs (L, σ) where L is a regular language of input-vertebrae (called the *language-symbol* of the edge) and σ is an output-vertebra. Unfortunately, we are still not done. The problem is that language-symbols must be disjoint across the entire set of edges of M. Only if this holds we have the right to consider two different (and disjoint) language-symbols L_1 and L_2 as different input symbols in M.[5] The example shown in Figure 5 illustrates the problem. If we consider L_1 and L_2 as different input symbols then we do not find a homing preset experiment for this machine: L_1 is not a HPX because L_1 is not accepted at state 2; similarly for L_2. However, a HPX exists, namely, $a \cdot a \cdot \mathsf{tick}$. In order to be able to detect this, we need to "split" L_1 into L'_1 and L'_2 and to update the edges as shown in the figure.

[5] Notice that for the output symbols of M there is no such issue: two output symbols σ_1 and σ_2 are the same iff the output-vertebrae σ_1 and σ_2 are identical.

In the general case, this transformation is done as follows:

Step 5 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. Then 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 $v \overset{L_i/\sigma}{\rightarrow} v'$ by the edges $v \overset{L'_{j_1}/\sigma}{\rightarrow} v'$, ..., $v \overset{L'_{j_i}/\sigma}{\rightarrow} v'$.

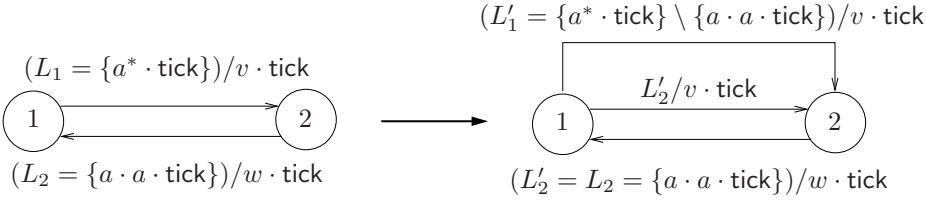

Fig. 5. A time-abstracting bisimulation quotient graph

Step 5 completes the transformation. Figure 6 shows the result of the transformation technique applied up to Step 4 to our running example (Step 5 is omitted because it results in a graph too complex to be readable). In the figure, the tick-nodes are drawn with double circles. The rest of the nodes are drawn with rectangles. In order not to overload the figure, we group together some nodes which are equivalent in the sense that no input sequence can distinguish them (nodes 19 and 21 for example). We also duplicate the nodes 15 and 18. The list of (non-disjoint) input language-symbols for this machine are: $A = \mathsf{tick}$, $B = \mathsf{click} \cdot \mathsf{tick}$, $C = \mathsf{click} \cdot \mathsf{click}^* \cdot \mathsf{tick}$, $D = \mathsf{click} \cdot \mathsf{click} \cdot \mathsf{click}^* \cdot \mathsf{tick}$, $E = \mathsf{click}^* \cdot \mathsf{tick}$. The corresponding disjoint input language-symbols are: $A' = \mathsf{tick}$ $(= A)$, $B' = \mathsf{click} \cdot \mathsf{tick}$ $(= B)$, $C' = \mathsf{click} \cdot \mathsf{click} \cdot \mathsf{click}^* \cdot \mathsf{tick}$ $(= D)$. The output symbols are: $X = \mathsf{tick}$, $Y = \mathsf{simple} \cdot \mathsf{tick}$, $Z = \mathsf{double} \cdot \mathsf{tick}$.

Once we have transformed G into M we can reduce the problem of finding a digital-clock experiment for A to the problem of finding the corresponding (untimed) experiment for M. We omit the definitions of untimed preset/adaptive homing/distinguishing experiments for non-deterministic Mealy machines, as they can be found in [1].[6] The following proposition gives the main result of this work.

[6] There are slight differences in the framework considered in the above paper, namely, the machines considered there are input-complete and homing experiments are not studied. However, extending the definitions and algorithms to cover these cases is straightforward.

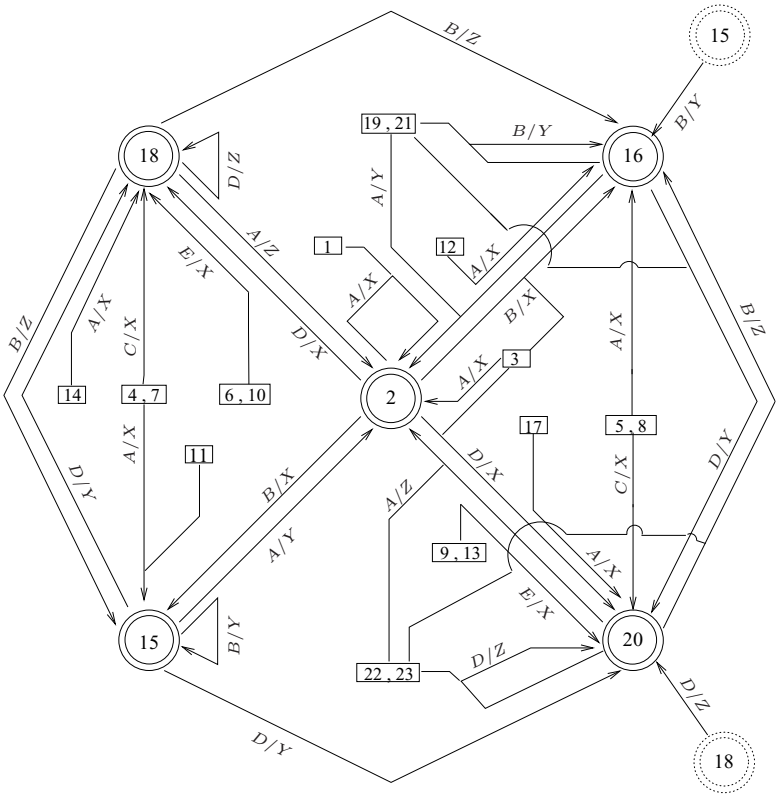

Fig. 6. A time-abstracting bisimulation quotient graph

Proposition 1. *A has a DPX (resp., HPX, DAX, HAX) iff M has a DPX (resp., HPX, DAX, HAX).*

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 its non-determinism. The strategy of the tester corresponds to choosing the inputs. The tester has incomplete information because it only observes the output, 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 remains to show how to construct an experiment for A given an experiment for M. We explain this in the case of preset homing experiments. The idea carries to other types of experiments as well. A HPX for M is a finite sequence

of language-symbols $L_1 \cdot L_2 \cdot \ ... \ \cdot L_m$, where each L_i is a regular language of input-vertebrae. For each i we choose arbitrarily an input-vertebra $\pi_i \in L_i$ (e.g., we may choose a π_i of minimal length). We claim that $\pi = \pi_1 \cdots \pi_m$ is a HPX for A. This is based on the following. First, the fact that different language-symbols in M are disjoint. Thus, when issuing a certain input-vertebra π_i, there is no ambiguity as to which language-symbol in M this π_i corresponds to. In our example above, it will be the language-symbol L_i. Second, the fact that all output-vertebrae end with a tick symbol. This ensures that when concatenating output-vertebrae to form the final output sequence given to the tester, the latter will have no ambiguity in interpreting the result. In particular, if σ_1, σ_2 are two vertebrae such that $\sigma_1 \neq \sigma_2$ then for any vertebrae σ_1', σ_2', $\sigma_1 \cdot \sigma_1' \neq \sigma_2 \cdot \sigma_2'$. Thus, if the output sequences are different at the level of M they will also differ at the level of A.

6 Summary and Future Work

We presented a method for solving state-identification problems for timed automata by generating their time-abstracting bisimulation quotient graph and then transforming the latter into a non-deterministic Mealy machine on which the same problems can be solved. Although we only studied distinguishing and homing experiments in this paper, the method should adapt easily to state-verification and synchronizing experiments as well. In the short term, we plan to identify upper and lower complexity bounds on the problems studied in this paper and experiment with a prototype implementation. One direction for future work is to consider analog-clock experiments. Apart from their theoretical interest, such experiments could also be useful in limiting state-explosion, in particular in cases where the constants involved in the Tick automaton are significantly smaller than those in the automaton under test. Still, a solid theory for implementation of analog-clock devices is lacking, thus should also be topic of future research. Another direction would be to remove, if possible, the bounded-output hypothesis used in this paper. Since most realistic systems meet this hypothesis, removing it is probably only of theoretical interest. However, it gives rise to an interesting question, namely, studying testing problems in the context of generalized Mealy machines or *sequential transducers*. Finally, another interesting direction is to consider related problems such as machine identification and learning. Some work in this direction has recently been reported in [6].

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. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

3. S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality*, volume 1536 of *LNCS*. Springer, 1998.
4. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Hybrid Systems III, Verification and Control*, volume 1066 of *LNCS*, pages 208–219. Springer-Verlag, 1996.
5. A. Gill. State-identification experiments in finite automata. *Information and Control*, 4:132–154, 1961.
6. O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. In *Joint conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS-FTRTFT'04)*, volume 3253 of *LNCS*. Springer, 2004.
7. M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th International SPIN Workshop on Model Checking of Software (SPIN'04)*, volume 2989 of *LNCS*. Springer, 2004.
8. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. *Proceedings of the IEEE*, 84:1090–1126, 1996.
9. E.F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, number 34. Princeton University Press, 1956.
10. J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symposium on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *LNCS*. Spinger-Verlag, 1996.
11. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, January 2001.