Inference of Event-Recording Automata using Timed Decision Trees *

(Extended Abstract)

Olga Grinchtein, Bengt Jonsson, and Paul Pettersson

Department of Computer Systems, Uppsala University, Sweden
{olgag, bengt, paupet}@it.uu.se

Abstract. We present an algorithm for inferring a timed-automaton model of a system from information obtained by observing its external behavior. Since timed automata can not in general be determinized, we restrict our attention to systems that can be described by deterministic *event-recording automata*. In previous work we have presented algorithms for event-recording automata that satisfy the restriction that there is at most one transition per alphabet symbol from each state. This restriction was lifted in subsequent work by an algorithm based on the region graph. In this paper, we extend previous work by considering the full class of event-recording automata, while still avoiding to base it on the (usually prohibitively large) region graph. Our construction deviates from previous work on inference of automata in that it first constructs a so called timed decision tree from observations of system behavior, which is thereafter folded into an automaton.

1 Introduction

Research during the last decades has developed powerful techniques for using models of reactive systems in specification, automated verification (e.g., [7]), test case generation (e.g., [9, 23]), implementation (e.g., [15]), and validation of reactive systems in telecommunication, embedded control, and related application areas. Typically, such models are assumed to be developed *a priori* during the specification and design phases of system development. In practice, however, often no formal specification is available, or becomes outdated as the system evolves over time. One must then construct a model that describes the behavior of an existing system or implementation. In software verification, techniques are being developed for generating abstract models of software modules by static analysis of source code (e.g., [8, 18]). However, peripheral hardware components, library modules, or third-party software systems do not allow static analysis. In practice, such systems must be analyzed by observing their external behavior.

* This work was supported in part by the Swedish Research Council, by the European Network of Excellence ARTIST2, and by the European Research Training Network Games.

Techniques for constructing finite-state models by analysis of externally observable behavior (black-box techniques) have been used, e.g., in regression testing [14, 19]), and in model checking [13] of finite-state systems for which no model or source code is available.

The construction of models from observations of system behavior can be seen as a learning problem. For finite-state reactive systems, this can be formulated as the problem of *regular inference:* to infer a (deterministic) finite automaton by posing a finite set of *membership queries*, each of which asks whether a certain word is accepted by the automaton or not. There are several slightly different algorithms for regular inference (e.g., [3, 10, 20, 22, 4]), which guarantee that a correct automaton will be constructed if "sufficiently many" membership queries have been posed. In some settings, the inference algorithm may also pose *equivalence queries* that ask whether a hypothesized automaton is equivalent to the one that is being investigated: such a query is answered either by *yes* or by a counterexample on which the hypothesis and the correct automata disagree.

In this paper, we extend the inference algorithm of Angluin and others to the setting of timed systems, more precisely systems that can be described by a timed automaton [1], i.e., a finite automaton equipped with clocks that constrain the times of occurrences of actions. One motivation is to develop techniques for creating abstract timed models of hardware components, device drivers, etc. for analysis of timed reactive systems. Since timed automata can not in general be determinized [1], we restrict consideration to the class of *event-recording automata* [2]. These are timed automata that, for every action a, use a clock that records the time of the last occurrence of a. Event-recording automata can be determinized, and are sufficiently expressive to model many interesting timed systems; for instance, they are as powerful as timed transition systems [16, 2], another popular model for timed systems. We assume that an inference algorithm observes a system by checking whether certain actions can be performed at certain moments in time, and that it is able to control and record precisely the timing of the occurrence of each action.

In previous work, authors of this paper [11] have presented algorithms for the restricted class of event-recording automata that have most one outgoing transition per action; under this restriction, the time of an action occurrence depends only on the (untimed) past sequence of actions, and not on their timing. In this paper, we consider the full class of event-recording automata. This is much more challenging, since it must be inferred how the timing of an action occurrence may depend on timing of previous actions. One way to address this problem is to infer an explicit representation of the region graph, as in [12]; this can be done using techniques for untimed regular inference, but at the cost of a high blow-up. In this paper, we avoid the region graph, but must then develop techniques for inferring guards on transitions of a timed automaton.

A novel feature of our algorithm (in comparison with other regular inference algorithms) is that it contains two constructions: the first construction generates a so-called *timed decision tree* from the answers to the membership queries that have been performed so far. The conditions (or guards) that distinguish between branches in this tree should correspond to guard in the inferred automaton. A guard is therefore introduced only if there is a pair of observations with different outcomes, where this guard is the only means to distinguish between them. The algorithm includes a systematic search for such pairs of observations. The second construction consist in folding the timed decision tree into an event-recording automaton by appropriate merging of nodes, in an analogous manner as in algorithms for (untimed) regular inference.

Previous work on inference of timed systems include work of authors of this paper [11, 12]. Several papers are concerned with finding a definition of timed languages which is suitable as a basis for inference algorithms. There are several works that define determinizable classes of timed automata (e.g., [2, 24]) and right-congruences of timed languages (e.g., [21, 17, 25]), motivated by testing and verification.

The paper is structured as follows. After preliminaries and a definition of event-recording automata in the next section, the construction of timed decision trees is described in Section 3 and 4, followed by a technique for folding them into automata in Section 5. An illustrating example is given in Section 6, and finally Section 7 concludes the paper. The proofs are omitted due to lack of space but will appear in the full version of this paper.

2 Preliminaries

We write $\mathbb{R}^{\geq 0}$ for the set of nonnegative real numbers, and $\mathbb{N}$ for the set of natural numbers. Let Σ be a finite alphabet of size $|\Sigma|$. A *timed word* over Σ is a finite sequence $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ of symbols $a_i \in \Sigma$ paired with nonnegative real numbers $t_i \in \mathbb{R}^{\geq 0}$ such that the sequence $t_1 t_2 \ldots t_n$ of time-stamps is nondecreasing. We use λ to denote the empty word.

An event-recording automaton contains for every symbol $a \in \Sigma$ a clock x_a, called the *event-recording clock* of a. Intuitively, x_a records the time elapsed since the last occurrence of the symbol a. We write C_Σ for the set $\{x_a \mid a \in \Sigma\}$ of event-recording clocks. A *clock valuation* γ is a mapping from C_Σ to $\mathbb{R}^{\geq 0}$.

Throughout the paper, we will use an alternative, equivalent, representation of timed words, namely clocked words. A *clocked word* over Σ is a sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2) \ldots (a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ that are paired with clock valuations which satisfies

- $\gamma_1(x_a) = \gamma_1(x_b)$ for all $a, b \in \Sigma$, and
- $\gamma_i(x_a) = \gamma_i(x_{a_{i-1}}) + \gamma_{i-1}(x_a)$ whenever $1 < i \leq n$ and $a \neq a_{i-1}$.

Each timed word $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ can be naturally transformed into the clocked word $(a_1, \gamma_1)(a_2, \gamma_2) \ldots (a_n, \gamma_n)$ where for each i with $1 \leq i \leq n$,

- $\gamma_i(x_a) = t_i$ if $a_j \neq a$ for $1 \leq j < i$,
- $\gamma_i(x_a) = t_i - t_j$ if there is a j with $1 \leq j < i$ and $a_j = a$, such that $a_k \neq a$ for $j < k < i$.

A *timed language* over Σ is a set of clocked words over Σ.

A *clock constraint* is a conjunction of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$, and $n \in \mathbb{N}$. We use $\gamma \models \varphi$ to denote that the clock valuation γ satisfies the clock constraint φ. A clock constraint is *K-bounded* if it contains no constant larger than K. A *clock guard* is a clock constraint whose conjuncts are only of the form $x \sim n$ (for $x \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$), i.e., comparison between clocks is not permitted. The set of clock guards is denoted by G_Σ.

A *constrained word* is a sequence $w_\varphi = (a_1, \varphi_1)(a_2, \varphi_2) \ldots (a_n, \varphi_n)$ of symbols $a_i \in \Sigma$ that are paired with clock constraints. A *guarded word* is a sequence $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ of symbols $a_i \in \Sigma$ that are paired with clock guards. For a clocked word $w_c = (a_1, \gamma_1)(a_2, \gamma_2) \ldots (a_n, \gamma_n)$ we use $w_c \models w_g$ to denote that $\gamma_i \models g_i$ for $1 \leq i \leq n$.

For a guarded word w_g, we introduce the *strongest postcondition* of w_g, denoted by $sp(w_g)$, as the constraint on clock values that are induced by w_g on any following occurrence of a symbol. Postcondition computation is central in tools for symbolic verification of timed automata [6, 5], and can be done inductively as follows:

- $sp(\lambda) = \bigwedge_{a,b \in \Sigma} x_a = x_b$,
- $sp(w_g(a, g)) = ((sp(w_g) \wedge g)[x_a \mapsto 0]) \uparrow$,

where for clock constraint φ and clock x,

- $\varphi[x \mapsto 0]$ is the condition $x = 0 \wedge \exists x.\varphi$,
- $\varphi \uparrow$ is the condition $\exists d.\varphi'$, where d ranges over $\mathbb{R}^{\geq 0}$ and where φ' is obtained from φ by replacing each clock y by $y - d$.

Both operations can be expressed as corresponding operations on clock constraints.

Definition 1. *An* event-recording automaton *(ERA) over Σ is a tuple $\mathcal{A} = \langle L, L_0, L^f, E \rangle$ consisting of a finite set L of* locations*, a set $L_0 \subseteq L$ of* start locations*, a set L^f of accepting locations, and a finite set E of edges. Each edge is a quadruple (l, l', a, g) with a source location $l \in L$, a target location $l' \in L$, an input symbol $a \in \Sigma$, and a clock guard $g \in G_\Sigma$.*

A *run* of the event-recording automaton $\mathcal{A}$ over the clocked word
$w_c = (a_1, \gamma_1) \;\ldots\; (a_n, \gamma_n)$ is a finite sequence

$$l_0 \xrightarrow{e_1} l_1 \xrightarrow{e_2} l_2 \cdots \xrightarrow{e_{n-1}} l_{n-1} \xrightarrow{e_n} l_n$$

of locations $l_i \in L$ and edges $e_i = (l_{i-1}, l_i, a_i, g_i) \in E$ such that $l_0 \in L_0$ and $\gamma_i \models g_i$ for all $1 \leq i \leq n$. The run is *accepting* if $l_n \in L_f$. The clocked word w_c is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over w_c, otherwise w_c is *rejected*. The timed language $\mathcal{L}(\mathcal{A})$ defined by $\mathcal{A}$ consists of all clocked words accepted by $\mathcal{A}$.

We call an ERA *time-deterministic* (TDERA) iff it has a single start location and for each location l and input symbol a, the guards $g_1, \ldots, g_n$ of edges from l labeled a are total and mutually exclusive, i.e., $g_1 \vee \ldots \vee g_n \equiv$ *true* and $g_i \wedge g_j \equiv$ *false* whenever $1 \leq i < j \leq n$.

3 Overview of Our Inference Algorithm

In this section, we define timed decision trees and give an overview of our inference algorithm. Our algorithm is designed to infer a TDERA which defines the same timed language as a given TDERA $\mathcal{A}$. In this algorithm a so called *Learner*, who initially knows nothing about $\mathcal{A}$, is trying to learn $\mathcal{L}(\mathcal{A})$ by asking queries to a *Teacher*, who knows $\mathcal{A}$. There are two kinds of queries:

- A *membership query* consists in asking whether a clocked word w_c over Σ is in $\mathcal{L}(\mathcal{A})$.
- An *equivalence query* consists in asking whether a hypothesized TDERA $\mathcal{H}$ is correct, i.e., whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{A})$. The *Teacher* will answer *yes* if $\mathcal{H}$ is correct, or else supply a counterexample, i.e, a clocked word w_c which is either in $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{H})$ or in $\mathcal{L}(\mathcal{H}) \setminus \mathcal{L}(\mathcal{A})$.

The typical behavior of a *Learner* is to start by asking a sequence of membership queries, and gradually build a hypothesized TDERA $\mathcal{H}$ using the obtained answers. When the *Learner* feels that she has built a hypothesis $\mathcal{H}$, she makes an equivalence query to find out whether $\mathcal{H}$ is correct. If the answer is *yes*, the *Learner* has succeeded, otherwise she uses the returned counterexample to revise $\mathcal{H}$ and perform subsequent membership queries until arriving at a new hypothesized TDERA, etc.

We assume that the upper bound K on the constants occurring in the guards of event-recording automaton is known a priori.

Let us represent the information gained by the *Learner* at any point during the learning process as a partial mapping *Obs* from clocked words to $\{+, -\}$, where $+$ stands for *accepted* and $-$ for *rejected*. The domain $Dom(Obs)$ of *Obs* is the set of clocked words for which membership queries have been performed, or which have been given as counterexamples in equivalence queries. An inference algorithm should prescribe how to pose additional membership queries and thereafter transform *Obs* into a TDERA $\mathcal{H} = \langle L, L_0, L^f, E \rangle$, which is *conformant* with *Obs*, in the sense that any string $w_c \in Dom(Obs)$ is accepted by $\mathcal{H}$ if $Obs(w_c) = +$, and rejected by $\mathcal{H}$ if $Obs(w_c) = -$.

Timed Decision Trees. Our algorithm first organizes *Obs* into a timed decision tree.

Definition 2. *A* timed decision tree *(TDT) is a prefix-closed set N of guarded words, such that for any node $w_g \in N$ and symbol a, the set of nodes $w_g(a, g_1), \ldots, w_g(a, g_n)$ in N that extend w_g by the symbol a is either empty, or have the properties that $g_1 \vee \cdots \vee g_n = true$ and $g_i \wedge g_j = false$ whenever $i \neq j$.*

$Obs(\lambda) = +$
$Obs((b, 0)) = +$
$Obs((a, 0)) = -$
$Obs((a, 0)(a, 0)) = +$
$Obs((a, 0)(b, 0)) = +$
$Obs((a, 0)(b, 0.5)) = -$
$Obs((a, 1.5)(b, 1.5)) = -$

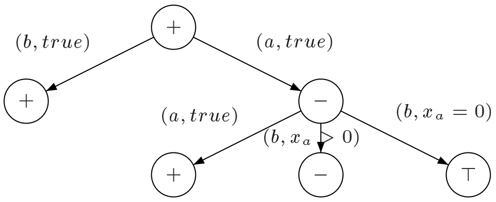


Fig. 1. A timed decision tree N.

To transfer the information in Obs to a timed decision tree, we extend the definition of Obs to guarded words, as follows.

$$Obs(w_g) = \begin{cases} \top & \text{if there are } w_c, w_c' \in Dom(Obs) \text{ with } w_c \vDash w_g \text{ and} \\ & \quad w_c' \vDash w_g \text{ such that } Obs(w_c) \neq Obs(w_c') \\ Obs(w_c) & \text{if there is } w_c \in Dom(Obs) \text{ with } w_c \vDash w_g \text{ such that} \\ & \quad Obs(w_c') = Obs(w_c) \text{ for all } w_c' \in Dom(Obs) \text{ with} \\ & \quad w_c' \vDash w_g \\ \bot & \text{if there is no } w_c \in Dom(Obs) \text{ with } w_c \vDash w_g \end{cases}$$

By this definition, the function Obs is overloaded so that it can also be regarded as a labeling of nodes in a timed decision tree. A node is labeled by $+$ (or $-$) if every observation that "leads to" the node is accepted (or rejected). A node is labeled by $\top$, and called *inconsistent*, if there are both accepted and rejected observations that lead to the node. If there are no observations that lead to the node, the node is labeled by $\bot$. A timed decision tree N is *consistent* w.r.t. Obs if it has no inconsistent nodes.

Definition 3. *A timed decision tree N* reflects *the partial mapping Obs if*

- *for any $w_c \in Dom(Obs)$ there is a $w_g \in N$ such that $w_c \vDash w_g$,*
- *N has no leaf w_g with $Obs(w_g) = \bot$.*

Example 1. Figure 1 shows an example of a set of observations[1] and a timed decision tree that reflects it[2]. □

Overview of Algorithm. From Obs, our algorithm constructs a timed decision tree which reflects Obs. Since each run of a TDERA corresponds to a guarded word, our ambition is that the tree will be a prefix of the unfolding of the learned TDERA. A main problem when constructing the tree is to find suitable guards

[1] To simplify the presentation we use timed words as a short representation of clocked words.

[2] In the figures we label the edges of a timed decision tree with pairs of the form (a, g) where $a \in \Sigma$ and $g \in G$. The guarded word of a node is the sequence of all labels on the path reaching the node.

that occur in edges of the TDERA. The *Learner* does not know *a priori* which guards to use in the final TDERA. Therefore, guards will be introduced "by need", when the information in *Obs* shows that they are necessary. Initially, the guarded words in the tree will use only *true* in guards. Whenever there is an inconsistent node w_g in the tree, it will be split into (at least) two guarded words, by splitting some guarded action in w_g according to whether a particular atomic clock guard is true or false. The appropriate clock guard is generated by posing additional membership queries in order to find a pair of observations with different outcomes (i.e., one accepted and one rejected), such that this guard is the only means to distinguish between them.

In order to fold the tree into an automaton, the inference algorithm also maintains a prefixed closed set U of nodes in the tree, which will be used to represent states in the TDERA. We let $succ(U)$ be the set of children of nodes in U which are not themselves in U. These nodes are used to construct transitions between nodes in U. The construction of U is based on a relation $\preceq_{un}$ between nodes of the tree which states which tree nodes may be merged when constructing an automaton. Intuitively $v_g \preceq_{un} u_g$ if the information about future behavior from the node v_g is contained in the information about future behavior from the node u_g, so that the node v_g can be merged with the node u_g when constructing an automaton.

When the tree satisfies certain properties the *Learner* constructs a hypothesized TDERA by merging each node in V with the corresponding node in U, and asks the *Teacher* an equivalence query. If a counterexample is returned, the *Learner* adds the counterexample to the timed decision tree, and the algorithm iterates. The algorithm terminates when a TDERA is found, for which the *Teacher* returns no counterexample.

In the next section we describe how to split inconsistent nodes in a timed decision tree, and in Section 5 we describe how to fold a timed decision tree into a TDERA by merging nodes.

4 Splitting Inconsistent Nodes

In this section, we describe our procedure for splitting inconsistent nodes in a timed decision tree. If $Obs(w_g) = \top$, then there are two clocked words $w_c, w'_c \in Dom(Obs)$ with $w_c \vDash w_g$ and $w'_c \vDash w_g$, such that $Obs(w_c) \neq Obs(w'_c)$. That is, w_c and w'_c both correspond to the same node, but one is accepted and one is rejected. In this section, we present our procedure for splitting some parent of w_g into two nodes using a suitable atomic clock guard that separates w_c and w'_c. More precisely, if $w_g = (a_1, g_1) \dots (a_n, g_n)$, then we must find an atomic guard g and a position i in w_g, and then split node $(a_1, g_1) \dots (a_i, g_i)$ into $(a_1, g_1) \dots (a_i, (g_i \wedge g))$ and $(a_1, g_1) \dots (a_i, (g_i \wedge \neg g))$.

4.1 Finding Critical Pairs

For two nonnegative real numbers p, p', define $p \approx p'$ if either

- $p > K$ and $p' > K$, or
- both p and p' are integers with $p = p'$, or
- both p and p' are non-integer with $\lfloor p \rfloor = \lfloor p' \rfloor$ (i.e., they have the same integer parts).

For two clock valuations γ and γ', define $\gamma \approx \gamma'$ if $\gamma(x_a) \approx \gamma'(x_a)$ for all event-recording clocks x_a. For two clocked words $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ and $w'_c = (a'_1, \gamma'_1) \dots (a'_n, \gamma'_n)$, define $w_c \approx w'_c$ if $a_i = a'_i$ and $\gamma_i \approx \gamma'_i$ for all $1 \leq i \leq n$.

Definition 4. *The clocked word $w'_c = (a_1, \gamma'_1) \dots (a_n, \gamma'_n)$ is* adjacent *to the clocked word $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ if $w_c \not\approx w'_c$ and for all i with $1 \leq i \leq n$ and symbols $a \in \Sigma$,*

- *whenever $\gamma'_i(x_a)$ is an integer, then $\gamma_i(x_a) = \gamma'_i(x_a)$,*
- *whenever $\gamma_i(x_a)$ is an integer, and $\gamma'_i(x_a)$ is not, then $|\gamma_i(x_a) - \gamma'_i(x_a)| < 1$,*
- *whenever $\gamma_i(x_a)$ is not an integer, then $\gamma_i(x_a) \approx \gamma'_i(x_a)$,*

Definition 5. *Let w_g be a guarded word. A* critical pair *in w_g is a pair of clocked words w_c, w'_c such that*

- *$Obs(w_c) \neq Obs(w'_c)$,*
- *$w_c \models w_g$ and $w'_c \models w_g$,*
- *w'_c is adjacent to w_c,*
- *there is no $w''_c \models w_g$ such that w'_c is adjacent to w''_c, and w''_c is adjacent to w_c.*

Intuitively, one clocked word in a critical pair is accepted and one is rejected, both correspond to the same inconsistent node, and they are "as adjacent as possible". The point of critical pairs is that they allow a natural construction of guards that split some parents of inconsistent nodes.

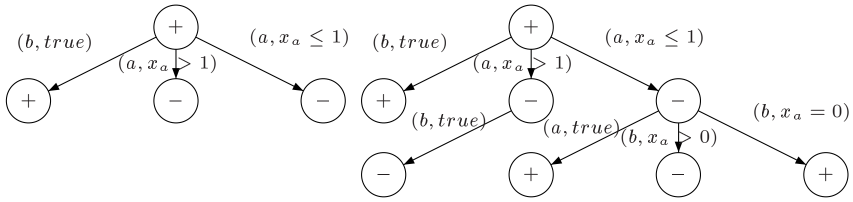
Theorem 1. *If there is a node w_g in the timed decision tree N such that $Obs(w_g) = \top$, then a critical pair in w_g can be found by $O(m|\Sigma| \log K)$ membership queries, where m is the length of w_g.*

Proof. We use binary search to find two adjacent clocked words and then solve a constraint satisfaction problem to find a critical pair. ◻

4.2 Splitting the Tree

In order to resolve inconsistency in a timed decision tree, we use a separating guard that can be inferred from a critical pair. A node is split according to the separating guard and subtree of the node is reconstructed. In general we maintain the property that every guard in a timed decision tree is inferred from a critical pair.

Definition 6. *Let w_c, w'_c be a critical pair in w_g, where $w_c = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ and $w'_c = (a_1, \gamma'_1) \dots (a_n, \gamma'_n)$. The* separating guard *$x_a \leq k$ at position i is* inferred from *w_c, w'_c if*



(a) N_1 (b) N_2

Fig. 2. Introducing a split into the tree N.

- *$k \leq K$,*
- *$\gamma_i(x_a) = k$ and $\gamma'_i(x_a) > k$,*
- *$\gamma_j \approx \gamma'_j$ for all $j < i$.*

Similarly, from the dual case the separating guard $x_a \geq k$ can be inferred.

Example 2. Suppose that we have observations $u_c = (a, \gamma_1(x_a){=}0.8, \gamma_1(x_b){=}0.8)(b, \gamma_2(x_a){=}0.2, \gamma_2(x_b){=}1.0)$ and $w_c = (a, \gamma'_1(x_a){=}0.8, \gamma'_1(x_b){=}0.8)(b, \gamma'_2(x_a){=}0.3, \gamma'_2(x_b){=}1.1)$ such that $Obs(u_c) \neq Obs(w_c)$. The clocked word w_c is adjacent to u_c. We see that u_c, w_c is a critical pair and that the only guard that can be inferred is $x_b \leq 1$, which separates γ_2 and γ'_2. □

We show an example how a split is introduced to the tree. Recall the tree N shown in Figure 1. The node $(a, true)(b, x_a = 0)$ is inconsistent. In order to resolve inconsistency we need to find a critical pair. Suppose that the critical pair is $(a, \gamma(x_a) = 1, \gamma(x_b) = 1)(b, \gamma(x_a) = 0, \gamma(x_b) = 1)$, $(a, \gamma(x_a) = 1.5, \gamma(x_b) = 1.5)(b, \gamma(x_a) = 0, \gamma(x_b) = 1.5)$. We infer guard $x_a \leq 1$ from it. Then we construct tree N_1 shown in Figure 2(a) by splitting node $(a, true)$ into nodes $(a, x_a \leq 1)$ and $(a, x_a > 1)$, we also remove successors of node $(a, true)$ from the tree N. Then we copy subtree of node $(a, true)$ of N to the subtree of node $(a, x_a \leq 1)$ of N_1 and construct tree N_2 shown in Figure 2(b). The node $(a, x_a > 1)(a, true)$ is missing since there is no corresponding observation.

5 Constructing Automata from Timed Decision Trees

In this section, we consider how to fold a timed decision tree into an event recording automaton by merging nodes.

Let N be a timed decision tree. In order to fold it into a TDERA, we maintain a prefix-closed subset U of N, which will be used as states of the automaton. Given a set U, the set $succ(U)$ is defined as the children of nodes in U which are not themselves in U. An automaton is formed by merging each node in $succ(U)$ with some node in U. This is done on the basis of a relation between nodes in N which say how nodes may be merged.

Definition 7. *Let N be a timed decision tree. A node $v_g \in N$ is* unifiable *with a node $u_g \in N$, denoted $v_g \leq_{un} u_g$, if for every descendant of v_g of form $v_g w_g$ with $Obs(v_g w_g) \in \{+, -\}$ and clocked word w_c with $w_c \vDash w_g$, there is a descendant of u_g of form $u_g w'_g$ with $w_c \vDash w'_g$ and $Obs(u_g w'_g) = Obs(v_g w_g)$.*

That is, the node v_g is unifiable with node u_g if v_g can use the subtree of u_g to handle suffixes of clocked words. This means that the subtree at u_g must agree with the subtree at v_g on all descendants of v_g that are accepted or rejected.

The relation $v_g \leq_{un} u_g$ on nodes in N can be checked symbolically in the following way. Say that two guarded words w_g and w'_g *overlap* if there is a clocked word w_c such that both $w_c \vDash w_g$ and $w_c \vDash w'_g$. This can be checked by propagating the constraints imposed by prefixes of w_g and w'_g on later clock constraints. For a clock constraint φ and a guarded word w_g we define (recursively) the constrained word $\varphi \,; w_g$ to be

$$\begin{aligned} \varphi \,; \lambda &= \lambda \\ \varphi \,; (a, g)\ w_g &= (a, \varphi \wedge g)\big(\ ((\varphi \wedge g)[x_a \mapsto 0]) \uparrow ;\ w_g\ \big) \end{aligned}$$

If φ is a postcondition of v_g, then $\varphi \,; w_g$ exactly characterizes the set of clocked words w_c that may appear as suffixes of clocked words $u_c w_c$ with $v_c w_c \vDash v_g w_g$. It follows that w_g and w'_g overlap if for $true; w_g = (a_1, g_1) \ldots (a_n, g_n)$ and $true; w'_g = (a_1, g'_1) \ldots (a_n, g'_n)$, we have $g_i \wedge g'_i \neq false$ for all $i = 1, \ldots, n$. Now $v_g \leq_{un} u_g$ if for any descendant $v_g w_g$ of v_g with $Obs(v_g w_g) = \{+, -\}$,

- there is at least one descendant $u_g w'_g$ of u_g such that w_g and w'_g overlap, and
- for any descendant $u_g w'_g$ of u_g such that w_g and w'_g overlap, we have $Obs(v_g w_g) = Obs(u_g w'_g)$.

In the case that $v_g \not\leq_{un} u_g$, there is one situation in which it is certain that v_g and u_g cannot be unified:

Definition 8. *The nodes u_g and v_g are* incompatible*, if they are distinguished by equivalent suffixes, i.e., if $Dom(Obs)$ contains clocked words $u_c w_c$ and $v_c w'_c$ such that*

- *$u_c \vDash u_g$ and $v_c \vDash v_g$,*
- *$w_c \approx w'_c$, and*
- *$Obs(u_c w_c) \neq Obs(v_c w'_c)$.*

The remaining case is when $v_g \not\leq_{un} u_g$, and u_g and v_g are not incompatible. In this case there are descendants $v_g w_g$ of v_g and $u_g w'_g$ of u_g such that w_g and w'_g overlap, but $Obs(v_g w_g) \neq Obs(u_g w'_g)$. Then we perform the following *unification* procedure. If $sp(v_g); w_g$ and $sp(u_g); w'_g$ overlap, we pose one or two more membership queries to check if v_g and u_g are incompatible. If $sp(v_g); w_g$ and $sp(u_g); w'_g$ do not overlap and $sp(v_g) = sp(u_g)$ we can make v_g unifiable with u_g by the following procedure. We remove all descendants nodes of v_g. If there is a clocked word $v_c(a_1, \gamma_1) \ldots (a_n, \gamma_n) \in Dom(Obs)$, where $v_c \vDash v_g$ and there is a node $w_g(a_1, g_1) \ldots (a_n, g_n)$ in the tree, then we add a node $v_g(a_1, g_1) \ldots (a_n, g_n)$ to the tree and ask a query if the node is labeled by $\perp$.

In order to fold a TDT N into a TDERA, two natural requirements should be fulfilled.

Definition 9. *Let N be a TDT, and let U be a prefix-closed subset of N. Then N is*

- U-complete *if for all $u_g \in U$ and $a \in \Sigma$ there is some $u_g(a, g) \in N$,*
- U-closed *if for all $v_g \in succ(U)$ there is $u_g \in U$ with $v_g \preceq_{un} u_g$.*

We can now describe our algorithm. Initially U contains the root $\{\lambda\}$. The *Learner* then asks membership queries, updates the tree, and maintains the set U, with the goal to make the tree consistent, U-complete, and U-closed. This is done by Algorithm 1.

Algorithm 1 Pseudo code for the construction consistent, U-closed and U-complete tree

Function construction(N,U)
  repeat
    while tree N is not consistent **do**
      Find inconsistent node $w_g \in N$.
      Perform splitting procedure which splits some prefix v_g of w_g into nodes v'_g and v''_g with new subtrees.
      if $v_g \in U \cup succ(U)$ **then** Add v'_g and v''_g to $succ(U)$.

    while tree N is not $U-$complete **do**
      Find $u_g \in U$ such that for some $a \in \Sigma$ there is no $g \in G_\Sigma$ with $u_g(a, g) \in N$.
      Add node $u_g(a, true)$ to N and to $succ(U)$.

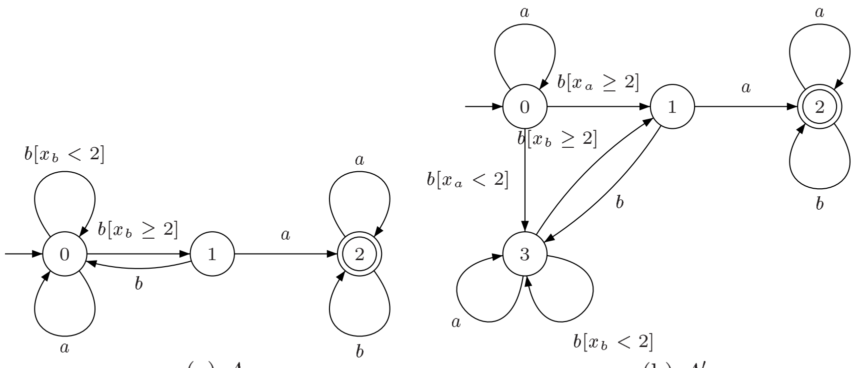
    while tree N is not $U-$closed **do**
      Find $v_g \in succ(U)$ such that $v_g \npreceq_{un} u_g$ for all $u_g \in U$.
      for each $u_g \in U$
        Perform unification procedure for v_g and u_g.
        if $v_g \preceq_{un} u_g$ **then continue_while**.
      Add v_g to U and all its successors to $succ(U)$.

  until tree N is consistent, $U-$closed and $U-$complete

From a consistent, U-complete, and U-closed timed decision tree we construct an event-recording automaton, called *merged automaton*, that agrees with the current set of observations *Obs*.

Definition 10. *Let the TDT N with set $U \subseteq N$ be a consistent, U-complete, and U-closed timed decision tree. The* merged automaton *of N, denoted $N_{\preceq_{un}}$ is the TDERA $\langle U, \{\lambda\}, L^f, E \rangle$, where*

- $L^f = \{\ u_g \in U\ :\ Obs(u_g) = +\ \}$,
- $(u_g, u_g(a, g), a, g) \in E$ *if* $u_g(a, g) \in U$, *otherwise* $(u_g, u'_g, a, g) \in E$ *with* $u_g(a, g) \in succ(U)$, *where* u'_g *for each* $u_g(a, g)$ *is chosen as a unique node with* $u_g(a, g) \preceq_{un} u'_g$.



(a) A (b) A'

Fig. 3. Automata A and A'.

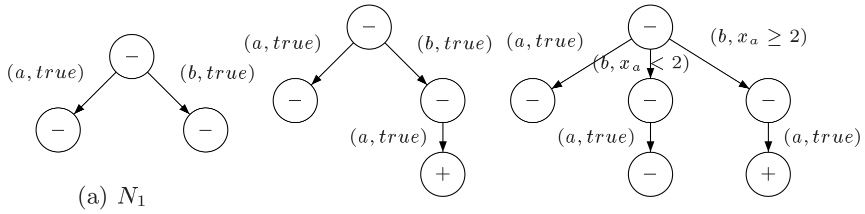
The locations of the merged automaton are nodes from U. For each $u_g(a, g)$ with $u_g \in U$, there is an edge from location u_g to location u'_g labeled with $a \in \Sigma$ and $g \in G_\Sigma$ if $u'_g = u_g(a, g)$ and $u_g(a, g) \in U$, or u'_g is a selected node in *succ*(U) which is unifiable with $u_g(a, g)$.

Theorem 2. *Let A be some TDERA which is equivalent to the one we are learning. Then the inference algorithm for learning a TDERA equivalent to A terminates after $O(|S|^l)$ equivalence queries, and $O(K^{4|\Sigma|(l+|A||S|^{|A|})}+|S|^{l+|A||S|^{|A|}}(l+|A||S|^{|A|})|\Sigma|\log K)$ membership queries, where $|S|$ is the maximal number of splits used in the transitions from a node in A, $|A|$ is the number of locations in A and l is the length of the longest counterexample.*

Let A' be a minimum size TDERA of Theorem 2. The output of our inference algorithm is an event-recording automaton A representing the timed language to be learned. The number of locations in A is less than $m2^n$, where n is the size of A' and m is the number of K-bounded clock constraints, which is $O(4^{|\Sigma|^2}K^{|\Sigma|^2})$. For comparison, in [12] authors of this paper present an algorithm that infers the region graph of some automaton representing the language to be learned. The number of equivalence queries is at most R and the number of membership queries is $O(|\Sigma|^2R^2lK)$, where R is the size of the region graph which is bounded by $2^n|\Sigma|!2^{|\Sigma|}K^{|\Sigma|}$ states, and l is the length of longest counterexample.

6 Example

Suppose the system to learn is the event-recording automaton A in Figure 3(a). We start by asking the queries λ, $(a, 0)$, and $(b, 0)$, and then construct the timed decision tree N_1, shown in Figure 4(a), with $U = \{\lambda\}$. The tree N_1 is U-complete, consistent and U-closed. The Learner construct hypothesized automaton A_1 which has one location and submits A_1 as equivalence query.

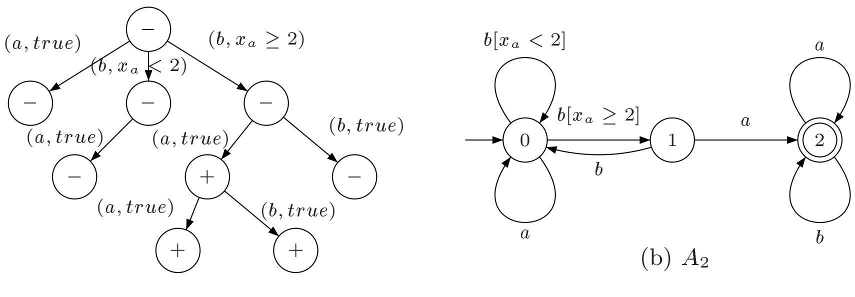


(a) N_1 (b) N_2 (c) N_3

Fig. 4. Trees N_1, N_2 and N_3.

Assume that the counterexample $(b, 2)(a, 2.4)$ is returned. It is accepted by A but rejected by A_1. Then the counterexample is added to the tree N_1 and the tree N_2, shown in Figure 4(b), is constructed. The tree is not U-closed as $(b, true) \not\leq_{un} \lambda$. Since $sp(\lambda); (a, true)$ and $sp((b, true)); (a, true)$ overlap, we ask query $(b, 0)(a, 0)$ that is not accepted by A. The node $(b, true)(a, true)$ is now labeled $\top$, and thus N_2 is inconsistent. By binary search we find the critical pair $(b, \gamma_1(x_a) = 2, \gamma_1(x_b) = 2)(a, \gamma_2(x_a) = 2.4, \gamma_2(x_b) = 0.4)$, $(b, \gamma_1'(x_a) = 1.75, \gamma_1'(x_b) = 1.75)(a, \gamma_2'(x_a) = 2.1, \gamma_2'(x_b) = 0.35)$ from which we infer the inequalities $x_a \geq 2$ and $x_b \geq 2$.

At this point we (arbitrarily) choose inequality $x_a \geq 2$ and construct timed decision tree N_3, shown in Figure 4(c), by splitting node $(b, true)$. The tree N_3 is not U-closed as $(b, x_a \geq 2) \not\leq_{un} \lambda$. Since $sp(\lambda); (a, true)$ and $sp((b, x_a \geq 2)); (a, true)$ overlap, we ask the queries $(a, 2.4)$ and $(b, 2)(a, 4.4)$ that are rejected and accepted by A respectively. Then λ and $(b, x_a \geq 2)$ are incompatible. We add $(b, x_a \geq 2)$ and $(b, x_a \geq 2)(a, true)$ to U. In order to make the tree U-complete we ask queries $(b, 2)(b, 2)$, $(b, 2)(a, 2)(a, 2)$, and $(b, 2)(a, 2)(b, 2)$. The resulting timed decision tree N_4 is shown in Figure 5(a).



(a) N_4 (b) A_2

Fig. 5. Tree N_4 and automaton A_2.

The tree N_4 is U-complete, consistent, and U-closed. The Learner construct the hypothesized automaton A_2, shown in Figure 5(b).

The procedure continues and the Learner will construct three more automata before automaton A', shown in Figure 3(b), is finally created (due to space limitations these steps are omitted). Note that the automaton A' contains only one state more than the automaton A.

7 Conclusion

We have presented a technique for inference of timed systems that can be represented as event-recording automata. We introduced timed decision tree as a data structure for organizing the results of the membership and equivalence queries. The timed decision tree is folded into an event-recording automaton by a merging procedure that is based on a notion of unifiable nodes. This is the first inference algorithm for the full class of event-recording automata which avoids explicit use of the region graph. We believe that the algorithm can be extended to deterministic timed automata.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.
3. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
4. J. L. Balcázar, J. Díaz, and R. Gavaldá. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. Kluwer, 1997.
5. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *LNCS*, pages 232–243. Springer-Verlag, 1996.
6. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer-Verlag, 1998.
7. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, Dec. 1999.
8. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng. Bandera : Extracting finite-state models from java source code. In *Proc. 22nd Int. Conf. on Software Engineering*, June 2000.
9. J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29, 1997.
10. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
11. O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. In *Proceedings of the Joint Conferences FORMATS and FTRTFT*, volume 3253 of *LNCS*, Sept. 2004.

12. O. Grinchtein, B. Jonsson, and M. Leucker. Inference of timed transition systems. *ENCS*, 138(3):87–99, 2005.
13. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In *Proc. TACAS'02*, LNCS, 2002.
14. A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In R.-D. Kutsche and H. Weber, editors, *Proc. FASE02*, volume 2306 of *LNCS*, pages 80–95. Springer Verlag, 2002.
15. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Engineering*, 16(4):403–414, April 1990.
16. T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112:173–337, 1994.
17. T. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In K. Larsen, S. Skuym, and G. Winskel, editors, *ICALP98*, volume 1443 of *LNCS*, pages 580–591. Springer Verlag, 1998.
18. G. Holzmann. Logic verification of ANSI-C code with SPIN. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification: Proc.* 7^{th} *Int. SPIN Workshop*, volume 1885 of *LNCS*, pages 131–147, Stanford, CA, 2000. Springer Verlag.
19. H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *Proc. CAV'03*, 2003.
20. M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
21. O. Maler and A. Pnueli. On recognizable timed languages. In *Proc. FOSSACS04*, LNCS. Springer-Verlag, 2004.
22. R. Rivest and R. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.
23. M. Schmitt, M. Ebner, and J. Grabowski. Test generation with autolink and testcomposer. In *Proc. 2nd Workshop of the SDL Forum Society on SDL and MSC - SAM'2000*, June 2000.
24. J. Springintveld and F. Vaandrager. Minimizable timed automata. In B. Jonsson and J. Parrow, editors, *Proc. FTRTFT'96*, volume 1135 of *LNCS*, pages 130–147. Springer Verlag, 1996.
25. T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In H. Langmaack, W. P. de Roever, and J. Vytopil, editors, *Proc. FTRTFT'94*, volume 863 of *LNCS*, pages 694–715. Springer Verlag, 1994.