Inference of Timed Transition Systems

Olga Grinchtein [1] Bengt Jonsson [2] Martin Leucker [3]

IT Department
Uppsala University
Uppsala, Sweden

Abstract

We extend Angluin's algorithm for on-line learning of regular languages to the setting of *timed transition systems*. More specifically, we describe a procedure for inferring systems that can be described by *event-recording automata* by asking a sequence of membership queries (does the system accept a given timed word?) and equivalence queries (is a hypothesized description equivalent to the correct one?). In the inferred description, states are identified by sequences of symbols together with timing information. The number of membership queries is polynomially in the region graph and in the biggest constant of the automaton to learn.

Key words: model inference, model learning, timed systems

1 Introduction

Research during the last decades have developed powerful techniques for using *models of reactive systems* in specification, automated verification (e.g., [9]), test case generation (e.g., [12,25]), implementation (e.g., [17]), 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In practice, however, often no formal specification is available, or becomes outdated as the system evolves over time.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 infer 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., [10,20]). However, peripheral hardware components, library modules, or third-party software systems do not allow static analysis. In practice, such systems must be analyzed

[1] Email: Olga.Grinchtein@it.uu.se
[2] Email: Bengt.Jonsson@it.uu.se
[3] Email: Martin.Leucker@it.uu.se
This author is supported by the European Research Training Network "Games".

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

by observing their external behavior. In fact, techniques for constructing models by analysis of externally observable behavior (black-box techniques) can be used in many situations.

- To create models of hardware components, library modules, that are part of a larger system which, e.g., is to be formally verified or analyzed.
- For regression testing, a model of an earlier version of an implemented system can be used to create a good test suite and test oracle for testing subsequent versions. This has been demonstrated, e.g., by Hungar et al. [16,21]).
- Black-box techniques, such as adaptive model checking [15], have been developed to check correctness properties, even when source code or formal models are not available.
- Tools that analyze the source code statically depend heavily on the implementation language used. Black-box techniques are easier to adapt to modules written in different languages.

The inference of models from observations of system behavior can be seen as a learning problem. For finite-state reactive systems, it means to construct a (deterministic) finite automaton from the answers to a finite set of *membership queries*, each of which asks whether a certain word is accepted by the automaton or not. There are several techniques (e.g., [4,13,22,24,5]) which use essentially the same basic principles; they differ in how membership queries may be chosen and in exactly how an automaton is constructed from the answers. The techniques guarantee that a correct automaton will be constructed if "enough" information is obtained. In order to check this, Angluin and others also allow *equivalence queries* that ask whether a hypothesized automaton accepts the correct language; such a query is answered either by *yes* or by a counterexample on which the hypothesis and the correct language disagree.

In [14], we extended the learning algorithm of Angluin and others to the setting of timed systems. We studied (a subclass of) event-recording automata (ERAs). These are timed automata [2] 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 [18,3], another popular model for timed systems.

For the approach presented in [14], however, we further restricted event-recording automata to be event-deterministic in the sense that each state has at most one outgoing transition per action (i.e., the automaton obtained by removing the clock constraints is deterministic). Under this restriction, timing constraints for the occurrence of an action depend only on the past sequence of actions, and not on their relative timing.

The chosen approach was based on the idea to reuse the techniques of learning regular systems instead of learning timed systems directly. Therefore, we established a characterization of timed languages accepted by DERAs in

terms of regular word languages. Such a regular word language can be understood as a symbolic representation of the timed language. As it is a regular language, methods like Angluin's algorithm can be used to estimate this symbolic language, provided symbolic queries can be answered. To achieve this, we described how symbolic words and timed words are related, and, more important, how to learn a symbolic word by several queries of timed words.

In this paper, we extend our previous results to the full class of event-recording automata (ERA). While we reuse the prosperous scheme developed in [14], the details are different. We work out a characterization in terms of a (symbolic) regular language for the language of ERAs. Furthermore, we show that each symbolic word can be identified by a *single* timed word. Thus, one query in Angluin's algorithm relates to a single timed query.

We introduce the algorithm LSDERA for learning deterministic event-recording automata. LSDERA learns a so-called *simple* deterministic event-recording automaton. We show that every deterministic event-recording automaton can be transformed into a unique simple one with at most single exponentially more locations. Our transformation is based on ideas used to derive so-called *region graphs*. We show that the number of membership queries of LSDERA is polynomial in the size of the biggest constant appearing in guards and in the number n of locations of the simple deterministic event-recording automaton. The number of equivalence queries is at most n.

Besides [14], we are not aware of any other work on learning of timed systems or timed languages. However, several papers are concerned with finding a definition of timed languages which is suitable as a basis for learning. There are several works that define determinizable classes of timed automata (e.g., [3,26]) and right-congruences of timed languages (e.g., [23,19,27]), motivated by testing and verification.

The paper is structured as follows. After preliminaries in the next section, we define event-recording automata (DERA) in Section 3, as well as, our techniques for learning ERAs and their timing constraints. Section 5 gives a short example.

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 $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ of symbols $a_i \in \Sigma$ that are paired with nonnegative real numbers t_i such that the sequence $t_1 t_2 \dots t_n$ of time-stamps is nondecreasing. We use λ to denote the empty word. A *timed language* over Σ is a set of timed words over Σ.

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. If there is no preceding occurrence of a, then the value of clock x_a is undefined, denoted by $\perp$. We write C_Σ for

the set $\{x_a | a \in \Sigma\}$ of event-recording clocks.

A *clock valuation* γ is a mapping from C_Σ to $\mathbb{R}^{\geq 0} \cup \{\bot\}$. A *clock constraint* is a conjunction of atomic constraints of the form $x = \bot$, $x \sim n$ 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; we then use the convention that $\bot$ satisfies only clock constraint $x = \bot$. A clock constraint is *K-bounded* if it contains no constant larger than K. Sometimes, when convenient, we identify all values greater than K and denote them by ∞. A clock constraint φ *identifies* a $|\Sigma|$-dimensional *polyhedron* $[\![\varphi]\!] \subseteq (\mathbb{R}^{\geq 0})^{|\Sigma|}$ viz. the vectors of real numbers satisfying the constraint. A *clock guard* is a clock constraint whose conjuncts are only of the form $x = \bot$ or $x \sim n$ (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. A clock guard g *identifies* a $|\Sigma|$-dimensional *hypercube* $[\![g]\!] \subseteq (\mathbb{R}^{\geq 0})^{|\Sigma|}$. A *simple clock guard* is a clock constraint whose conjunctions are only of the form $x = \bot$, $x = n$, $n < x < n + 1$ or $x > K$ (for $x \in C_\Sigma$). A *region constraint* is a clock constraint of the form $\bigwedge_{x \in C_\Sigma} c(x) \wedge \bigwedge_{x,y \in C_\Sigma} d(x, y)$ where $c(x)$ is of the form $x = n$, $n < x < n + 1$, or $x > K$, and, $d(x, y)$ is of the form $x - y = n$ or $n < x - y < n + 1$.

Clock constraints can efficiently and uniquely be represented using difference bound matrices (DBMs, [11]). Furthermore, DBMs allow efficient operations on clock constraints like intersection, checking equality etc.

A *clocked word* w_c is a sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2) \dots (a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ that are paired with event-clock valuations. Each timed word $w_t = (a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ can be naturally transformed into a clocked word $CW(w_t) = (a_1, \gamma_1)(a_2, \gamma_2) \dots (a_n, \gamma_n)$ where for each i with $1 \leq i \leq n$,

- $\gamma_i(x_a) = \bot$ 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 *guarded word* w_g is a sequence $w_g = (a_1, g_1)(a_2, g_2) \dots (a_n, g_n)$ of symbols $a_i \in \Sigma$ that are paired with clock guards. We require that each g_i may only reference defined clocks, i.e., clocks x_a such that a is among $a_1 a_2 \dots a_{i-1}$. Note that we identify an empty conjunction with *true*. For a clocked word $w_c = (a_1, \gamma_1)(a_2, \gamma_2) \dots (a_n, \gamma_n)$ we use $w_c \models w_g$ to denote that $\gamma_i \models g_i$ for $1 \leq i \leq n$. For a timed word w_t we use $w_t \models w_g$ to denote that $CW(w_t) \models w_g$.

$\varphi \uparrow$ 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 $y - d$.

A guarded word $w_g = (a_1, g_1)(a_2, g_2) \dots (a_n, g_n)$ is called a *guard refinement* of $a_1 a_2 \dots a_n$, and $a_1 a_2 \dots a_n$ is called the word *underlying* w_g. The word w underlying a timed word w_t is defined in a similar manner.

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 [8,6], and can be

done inductively as follows:

- $sp(\lambda) = true$,
- $sp(w_g(a,g)) = ((sp(w_g) \wedge g)[x_a \mapsto 0]) \uparrow$ if all clocks referenced in g are defined by w_g, otherwise $sp(w_g(a,g)) = false$,

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 $y - d$.

A *deterministic finite automaton* (DFA) $\mathcal{A} = \langle \Gamma, L, l_0, L^f, \delta \rangle$ over the alphabet Γ consists of states L, initial state l_0, a set $L^f \subseteq L$ of accepting states and a partial transition function $\delta : L \times \Gamma \to L$. A *run* of $\mathcal{A}$ over the word $w = a_1 a_2 \dots a_n$ is a finite sequence $l_0 \xrightarrow{a_1} l_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} l_n$ of states $l_i \in L$ such that l_0 is the initial state and $\delta(l_{i-1}, a_i)$ is defined for $1 \leq i \leq n$, with $\delta(l_{i-1}, a_i) = l_i$. In this case, we write $\delta(l_0, w) = l_n$, thereby extending the definition of δ in the natural way. The language $\mathcal{L}(\mathcal{A})$ comprises all words $a_1 a_2 \dots a_n$ over which an accepting run exists, where a run is accepting iff $\delta(l_0, w) \in L^f$.

3 Event-recording automata

Definition 3.1 *An* event-recording automaton *(ERA)* $D = \langle \Sigma, L, l_0, L^f, \delta \rangle$ *consists of a finite* input alphabet Σ*, a finite set* L *of* locations*, an* initial location $l_0 \in L$*, a set* L^f *of accepting locations, a* transition function $\delta : L \times \Sigma \times G \to L$*, which is a partial function that for each location, input symbol and guard potentially prescribes a target location.*

We call an ERA *time-deterministic* iff $\delta(l, a, g_1) = l_1$ and $\delta(l, a, g_2) = l_2$ implies $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$ or $l_1 = l_2$ Thus, while a location l might have two different a successors, these can be distinguished by the guard.

Theorem 3.2 ([3]) *Every ERA can be transformed into an equivalent time-deterministic ERA.*

Therefore we will concentrate on time-deterministic ERAs, or TDERAs for short, in the following.

In order to define the language accepted by a TDERA, we first understand it as a DFA.

Given a TDERA $D = \langle \Sigma, L, l_0, L^f, \delta \rangle$, we define $dfa(D)$ to be the DFA $\mathcal{A}_D = \langle \Gamma, L, l_0, L^f, \delta' \rangle$ over the alphabet $\Gamma = \Sigma \times G$ where $\delta' : L \times \Gamma \to L$ is defined by $\delta'(l, (a, g)) = \delta(l, a, g)$ if and only if $\delta(l, a, g)$ is defined, otherwise $\delta'(l, (a, g))$ is undefined. Note that D and $dfa(D)$ have the same number of locations/states. Further, note that this mapping from TDERAs over Σ to DFAs over $\Sigma \times G$ is injective, meaning that for each DFAs $\mathcal{A}$ over $\Sigma \times G$, there is a unique (up to isomorphism) ERA over Σ, denoted $tdera(\mathcal{A})$, such

that $dfa(tdera(\mathcal{A}))$ is isomorphic to $\mathcal{A}$.

The language $\mathcal{L}(D)$ accepted by a TDERA D is defined to be the set of timed words w_t such that $w_t \models w_g$ for some guarded word $w_g \in \mathcal{L}(dfa(D))$. We call two TDERAs D_1 and D_2 equivalent iff $\mathcal{L}(D_1) = \mathcal{L}(D_2)$, and denote this by $D_1 \equiv_t D_2$, or just $D_1 \equiv D_2$. A TDERA is *K-bounded* if all its guards are K-bounded.

Definition 3.3 *A TDERA D is* simple *if for all guarded words $w_g(a,g) \in \mathcal{L}(dfa(D))$, we have that g is simple and $g \wedge sp(w_g)$ is satisfiable.*

We remark that whether or not a TDERA is simple depends only on $\mathcal{L}(dfa(D))$. A consequence of this definition is the following.

Lemma 3.4 *If $w_g(a,g) \in \mathcal{L}(dfa(D))$, where D is a simple TDERA, then there is a timed word $w_t(a,t) \in \mathcal{L}(D)$ such that $w_t(a,t) \models w_g(a,g)$.*

Proof. The claim follows easily from the definition of simple. □

Every TDERA can be transformed into an equivalent TDERA that is simple using the region-graph construction [1].

Lemma 3.5 *For every TDERA there is an equivalent TDERA that is simple.*

Proof. Let the TDERA $D = \langle \Sigma, L, l_0, L^f, \delta \rangle$ be K-bounded. We define an equivalent simple TDERA $D' = \langle \Sigma, L', l_0', L^{f'}, \delta' \rangle$ based on the so-called region automaton for D. We sketch the construction, details can be found in [1,7].

The set of locations of D' comprises pairs (l, φ) where $l \in L$ and φ is a K-bounded region constraint. However, (l, φ) has a slightly different interpretation than in the region graph construction. It should be understood as D is in location l and the time *starts* in some point given by φ. In other words, we think of $\varphi \uparrow$ rather than of φ.

To turn the region graph into an automaton, we have to add a transition function and final states. Intuitively, we proceed from (l, φ) to (l', φ') by an action a and a simple guard g, if D can proceed from l to l' by a, respecting some constraint $\hat{g}$ given in D. Then, φ' is the region obtained by constraining $\varphi \uparrow$ with g and resetting the clock for a. In other words, for every symbol a and simple guard g, let $\delta'((l, \varphi), a, g)$ be defined as (l', φ') if there exists a guard $\hat{g}$ that implies g and for which $\delta(l, a, \hat{g})$ is defined and is l', and $\varphi' = (\varphi \uparrow \wedge\, g)[x_a \mapsto 0]$, and $\varphi' \neq false$. Otherwise, it is undefined. The final states are given by: $(\delta(l, a, g), \varphi'') \in L^{f'}$ iff $\delta(l, a, g) \in L^f$.

It is routine to show that the part of the automaton reachable from the initial location $(l_0, true)$ is simple. □

The important property of simple TDERAs is that equivalence coincides with equivalence on the corresponding DFAs.

Definition 3.6 *We call two simple TDERAs D_1 and D_2* dfa-equivalent*, denoted by $D_1 \equiv_{dfa} D_2$, iff $dfa(D_1)$ and $dfa(D_2)$ accept the same language (in the sense of DFAs).*

Lemma 3.7 *For two simple TDERAs D_1 and D_2, we have*

$$D_1 \equiv_t D_2 \text{ iff } D_1 \equiv_{dfa} D_2$$

Proof. The direction from right to left follows immediately, since $\mathcal{L}(D_i)$ is defined in terms of $\mathcal{L}(dfa(D_i))$. To prove the other direction, assume that $D_1 \not\equiv_{dfa} D_2$. Then there is a shortest w_g such that $w_g \in \mathcal{L}(dfa(D_1))$ but $w_g \notin \mathcal{L}(dfa(D_2))$ (or the other way around). By Lemma 3.4 this implies that there is a timed word w_t such that $w_t \in \mathcal{L}(D_1)$ but $w_t \notin \mathcal{L}(D_2)$, i.e., $D_1 \not\equiv_t D_2$. □

We can now prove the central property of simple TDERAs.

Theorem 3.8 *For every TDERA there is a unique equivalent minimal simple TDERA (up to isomorphism).*

Proof. By Lemma 3.5, each TDERA D can be translated into an equivalent TDERA D' that is simple. Let $\mathcal{A}_{min}$ be the unique minimal DFA which is equivalent to $dfa(D')$ (up to isomorphism). Since (as was remarked after Definition 3.3) whether or not a TDERA is simple depends only on $\mathcal{L}(dfa(D))$, we have that $D_{min} = tdera(\mathcal{A}_{min})$ is simple. By Lemma 3.7, D_{min} is the unique minimal simple TDERA (up to isomorphism) such that $D_{min} \equiv D'$, i.e., such that $D_{min} \equiv D$. □

4 Learning event-recording automata

Learning a DFA

Angluin's learning algorithm is designed for learning a regular (untimed) language, $\mathcal{L}(\mathcal{A}) \subseteq \Gamma^*$, accepted by a minimal deterministic finite automaton (DFA) $\mathcal{A}$ (when adapted to the case that $\mathcal{L}(\mathcal{A})$ is prefix-closed). 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 string $w \in \Gamma^*$ is in $\mathcal{L}(\mathcal{A})$.
- An *equivalence query* consists in asking whether a hypothesized DFA $\mathcal{H}$ 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 w, either in $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{H})$ or in $\mathcal{L}(\mathcal{H}) \setminus \mathcal{L}(\mathcal{A})$.

The *Learner* maintains a prefix-closed set $U \subseteq \Gamma^*$ of prefixes, which are candidates for identifying states, and a suffix-closed set $V \subseteq \Gamma^*$ of suffixes, which are used to distinguish such states. The sets U and V are increased when needed during the algorithm. The *Learner* makes membership queries for all words in $(U \cup U\Gamma)V$, and organizes the results into a *table* T which maps each $u \in (U \cup U\Gamma)$ to a mapping $T(u) : V \mapsto \{\text{accepted}, \text{not accepted}\}$. In [4], each function $T(u)$ is called a *row*. When T is *closed* (meaning that for each $u \in U$, $a \in \Gamma$ there is a $u' \in U$ such that $T(ua) = T(u')$) and *consistent* (meaning

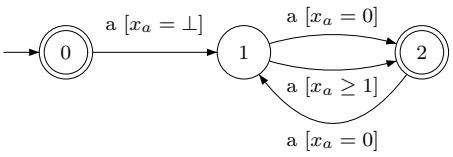

Fig. 1. Automaton A_1

that $T(u) = T(u')$ implies $T(ua) = T(u'a)$), then the *Learner* constructs a hypothesized DFA $\mathcal{H} = \langle \Gamma, L, l_0, \delta \rangle$, where $L = \{T(u) \mid u \in U\}$ is the set of distinct rows, l_0 is the row $T(\lambda)$, and δ is defined by $\delta(T(u), a) = T(ua)$, and submits $\mathcal{H}$ in an equivalence query. If the answer is *yes*, the learning procedure is completed, otherwise the returned counterexample is used to extend U and V, and perform subsequent membership queries until arriving at a new hypothesized DFA, etc.

For Angluin's algorithm it is known that the number of membership queries can be bounded by $O(kn^2m)$, where n is the number of states, k is the size of the alphabet, and m is the length of the longest counterexample. The rough idea is that for each entry in the table T a query is needed, and $O(knm)$ is the number of rows, n the number of columns.

Learning a TDERA

Given a timed language that is accepted by a TDERA D, we can assume without loss of generality that D is the unique minimal and simple one that exists due to Theorem 3.8. Then D is uniquely determined by its symbolic language of $\mathcal{A} = \mathit{dfa}(D)$, which is a regular (word) language. Thus, we can learn $\mathcal{A}$ using Angluin's algorithm and return $\mathit{tdera}(\mathcal{A})$. However, $\mathcal{L}(\mathcal{A})$ is a language over simple guarded words, but the *Teacher* in the timed setting is supposed to deal with timed words rather than guarded words.

Let us therefore extend the *Learner* in Angluin's algorithm by an *Assistant*, whose role is to answer a membership query for a simple guarded word, posed by the *Learner*, by asking a membership query for timed word to the (timed) *Teacher*. Furthermore, it also has to answer equivalence queries, consulting the timed *Teacher*.

For a simple guarded word $w = (a_1, g_1) \dots (a_n, g_n)$ each simple guard g that extends w together with an action a defines exactly one region. Thus, if w is accepted, it is enough to check a in a single point in this region defined by g and the postcondition of w. In other words, it suffices to check an arbitrary timed word $w_t \models w$ to check whether w is in the symbolic language or not.

The number of successor regions that one region can have is $O(|\Sigma|K)$. Then the complexity of the algorithm is $O(|\Sigma|^2 n^2 mK)$.

T_1	λ
λ	1
$a(x_a = \bot)$	0

(a)

T_2	λ
λ	1
$(a, x_a = \bot)$	0
$(a, x_a = \bot)(a, x_a = 0)$	1
$(a, x_a = \bot)(a, 0 < x_a < 1)$	0
$(a, x_a = \bot)(a, x_a = 1)$	1
$(a, x_a = \bot)(a, x_a > 1)$	1

(b)

Table 1

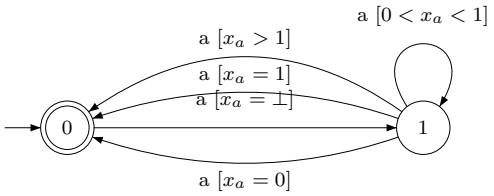

Fig. 2. Atomaton A_2

5 Example

Let us explain the algorithm by showing how to learn the language of the automaton A_1 depicted in Figure 1. Initially, the algorithm asks membership queries for λ and $(a, x_a = \bot)$. This yields the initial observation table T_1 shown in Table 1(a).[4] It is consistent but not closed, since $row((a, x_a = \bot))$ is distinct from $row(\lambda)$. Following Angluin's algorithm, we can construct a closed and consistent table T_2 shown in Table 1(b). Then the *Learner* constructs a hypothesized TDERA A_2 shown in Figure 2 and submits A_2 in an equivalence query. Assume that the counterexample $(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)$ is returned. It is accepted by A_1 but rejected by A_2. The algorithm processes the counterexample and finally produces the observation table T_3 given in Table 2. The automaton A_3 visualized in Figure 3 corresponds to the observation table T_3 and accepts the same language as A_1.

6 Conclusion

In this paper, we presented a technique for learning timed transitions systems based on their representation as event-recording automata (ERA). We show that the timed language of every ERA can uniquely be represented by a regular language of guarded words, using ideas of the so-called region graph

[4] 0 represents non-accepting while 1 represents accepting.

T_3	λ	w_g
λ	1	0
$(a, x_a = \bot)$	0	0
$(a, x_a = \bot)(a, x_a = 0)$	1	1
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)$	1	1
$(a, x_a = \bot)(a, 0 < x_a < 1)$	0	0
$(a, x_a = \bot)(a, x_a = 1)$	1	0
$(a, x_a = \bot)(a, x_a > 1)$	1	0
$(a, x_a = \bot)(a, x_a = 0)(a, 0 < x_a < 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a > 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a > 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)(a, x_a = 1)$	0	0
$(a, x_a = \bot)(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)(a, x_a > 1)$	0	0

Table 2
$w_g = (a, x_a = 0)(a, x_a = 0)$

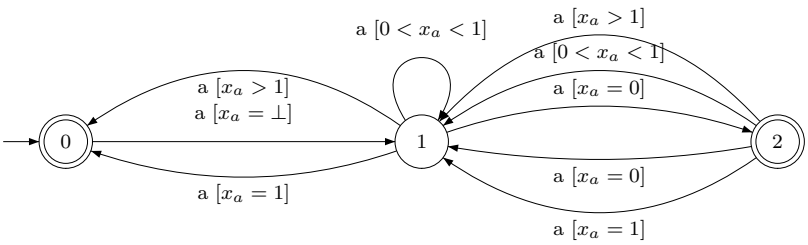

Fig. 3. Automaton A_3

construction. This allows us to adapt existing algorithms for learning regular languages to the timed setting. The main additional work is to learn the guards under which individual actions will be accepted.

The complexity of our learning algorithm is polynomial in the size of the region graph. In general, this can be exponentially larger than a minimal deterministic ERA automaton representing the same language. It would interesting to establish lower bounds of the learning problem for timed systems.

References

[1] R. Alur. Timed automata. In *Proc. 11th International Computer Aided Verification Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, 1999.

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

[3] R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[4] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[5] 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.

[6] 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 *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, 1996.

[7] P. Bouyer. Untameable timed automata. In H. Alt and M. Habib, editors, *Symp. on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer Verlag, 2003.

[8] 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. 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer-Verlag, 1998.

[9] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Dec. 1999.

[10] 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.

[11] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite-State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer Verlag, 1989.

[12] 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.

[13] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[14] O. Grinchtein, B. Jonsson, and M. Leucker. Learning of event-recording automata. In *Proceedings of the Joint Conferences FORMATS and FTRTFT*, Lecture Notes in Computer Science, Sept. 2004. to appear

[15] A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In J.-P. Katoen and P. Stevens, editors, *Proc. TACAS '02, 8^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 357–370. Springer Verlag, 2002.

[16] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In R.-D. Kutsche and H. Weber, editors, *Proc. FASE '02, 5^{th} Int. Conf. on Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 80–95. Springer Verlag, 2002.

[17] 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.

[18] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112:173–337, 1994.

[19] T. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In K. Larsen, S. Skuym, and G. Winskel, editors, *Proc. ICALP '98, 25^{th} International Colloquium on Automata, Lnaguages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer Verlag, 1998.

[20] 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 *Lecture Notes in Computer Science*, pages 131–147, Stanford, CA, 2000. Springer Verlag.

[21] H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *Proc. 15^{th} Int. Conf. on Computer Aided Verification*, 2003.

[22] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[23] O. Maler and A. Pnueli. On recognizable timed languages. In *Proc. FOSSACS04, Conf. on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer-Verlag, 2004. Available from `http://www-verimag.imag.fr/PEOPLE/Oded.Maler/`.

[24] R. Rivest and R. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

[25] 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.

[26] J. Springintveld and F. Vaandrager. Minimizable timed automata. In B. Jonsson and J. Parrow, editors, *Proc. FTRTFT'96, Formal Techniques in Real-Time and Fault-Tolerant Systems, Uppsala, Sweden*, volume 1135 of *Lecture Notes in Computer Science*, pages 130–147. Springer Verlag, 1996.

[27] 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, Formal Techniques in Real-Time and Fault-Tolerant Systems, Lübeck, Germany*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer Verlag, 1994.