Theoretical Computer Science 411 (2010) 4029–4054

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Learning of event-recording automata

Olga Grinchtein [a,*], Bengt Jonsson [a], Martin Leucker [b]

[a] *Department of Information Technology, Uppsala University, Uppsala, Sweden*

[b] *Institut für Informatik, TU München, München, Germany*

ARTICLE INFO

Article history:
Received 28 January 2009
Received in revised form 23 May 2010
Accepted 20 July 2010
Communicated by J. Esparza

Keywords:
Model inference
Model learning
Timed systems

ABSTRACT

In regular inference, a regular language is inferred from answers to a finite set of membership queries, each of which asks whether the language contains a certain word. One of the most well-known regular inference algorithms is the L^* algorithm due to Dana Angluin. However, there are almost no extensions of these algorithms to the setting of timed systems. We extend Angluin's algorithm for on-line learning of regular languages to the setting of timed systems. Since timed automata can freely use an arbitrary number of clocks, we restrict our attention to systems that can be described by deterministic event-recording automata (DERAs). We present three algorithms, TL^*_{sg}, TL^*_{nsg} and TL^*_s, for inference of DERAs. In TL^*_{sg} and TL^*_{nsg}, we further restrict event-recording automata to be event-deterministic in the sense that each state has at most one outgoing transition per action; learning such an automaton becomes significantly more tractable. The algorithm TL^*_{nsg} builds on TL^*_{sg}, by attempts to construct a smaller (in number of locations) automaton. Finally, TL^*_s is a learning algorithm for a full class of deterministic event-recording automata, which infers a so called *simple* DERA, which is similar in spirit to the region graph.

1. Introduction

Research during the last decades have developed powerful techniques for using *models of reactive systems* in specification, automated verification (e.g., [13]), test case generation (e.g., [16,30]), implementation (e.g., [22]), 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In practice, however, often no formal specification is available, or becomes outdated as the system evolves over time.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., [12,25]). 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. 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. [21,24]).

* Corresponding author. Tel.: +46 739082140.
E-mail addresses: olgag@it.uu.se, olga.grinchtein@ericsson.com (O. Grinchtein), bengt@it.uu.se (B. Jonsson), leucker@in.tum.de (M. Leucker).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.07.008

- Black-box techniques, such as adaptive model checking [19], 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 construction 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., [4,18,27,29,5]) 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. Techniques for learning finite automata have been successfully used for regression testing [21] and model checking [19] of finite-state systems for which no model or source code is available.

In this paper, we extend the techniques for automata learning developed by Angluin and others to the setting of timed systems. One longer-term goal is to develop techniques for creating abstract timed models of hardware components, device drivers, etc. for analysis of timed reactive systems; there are many other analogous applications. It is not an easy challenge, and we will therefore in this first work make some idealizing assumptions. We assume that a learning algorithm observes a system by checking whether certain actions can be performed at certain moments in time, and that the learner is able to control and record precisely the timing of the occurrence of each action. We consider systems that can be described by a timed automaton [1], i.e., a finite automaton equipped with clocks that constrain the possible absolute times of occurrences of actions. There are some properties of timed automata that make the design of learning algorithms difficult: the set of clocks is not known *a priori*, and they cannot in general be determinized [1]. We therefore restrict consideration to a 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 [23,2], another popular model for timed systems.

Although event-recording automata overcome some obstacles of timed automata, they still suffer from problems. One problem is that it is not clear how to generalize Nerode's right congruence, another is that in general they do not have canonical forms. Therefore we work with classes of event-recording automata which have canonical forms and can be understood as finite automata over a symbolic alphabet.

We present three algorithms, TL^*_{sg}, TL^*_{nsg} and TL^*_s, for learning deterministic event-recording automata.

In algorithms TL^*_{sg} and TL^*_{nsg}, we further restrict 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; learning such an automaton becomes significantly more tractable, and allows us to adapt the learning algorithm of Angluin to the timed setting.

TL^*_{sg} learns a so-called *sharply guarded* event-deterministic event-recording automaton. We show that every deterministic event-recording automaton can be transformed into a unique sharply guarded one with at most double exponentially more locations. We show that if the size of the untimed alphabet is fixed, then the number of membership queries of TL^*_{sg} is a polynomial in the size of the biggest constant appearing in guards, in the number n of locations of the sharply guarded event-deterministic event-recording automaton, in the size of the timed alphabet and in the length of the longest counterexample. The number of equivalence queries is at most n.

The algorithm TL^*_{nsg} addresses the problem of learning a smaller, not necessarily sharply guarded version of an event-deterministic event-recording automaton. It achieves this goal by *unifying* the queried information when it is "similar" which results in merging states in the constructed automaton. The number of needed queries exceeds those of TL^*_{sg} in the worst case; however, in practice it can be expected that it behaves better than TL^*_{sg}.

TL^*_s is a learning algorithm for the full class of deterministic event-recording automata. While we reuse the prosperous scheme developed in TL^*_{sg}, the details are different. We work out a characterization in terms of a (symbolic) regular language for the language of DERAs. 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. TL^*_s 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 TL^*_s is a polynomial in the size of the biggest constant appearing in guards, in the number n of locations of the simple deterministic event-recording automaton, in the size of the untimed alphabet and in the length of the longest counterexample. The number of equivalence queries is at most n.

Related work. In another work [17,20], two of the authors of this paper have developed a completely different algorithm for learning deterministic event-recording automata. The algorithm differs from TL^*_s in that the constructed automaton need not be a simple one. The transformation of an event-recording automaton to a corresponding simple automaton often increases its size significantly; a reason for this increase is that each transition is divided up into many "smaller" transitions, at least one transition for each possible integer value of the clocks of the automaton. The algorithm presented in [17,20] attempts to avoid this division of transitions by constructing an automaton, whose guards are "similar" to the guards of the event-recording automaton that is being learned. The main problems are then to synthesize "natural" guards on transitions, and to construct locations (control states). To address these problems, the algorithm does not re-use the structure of the L^* algorithm; instead of the observation table used in L^*, it uses a new data structure, called a timed decision tree, to organize results of queries. Its theoretical worst-case complexity is significantly higher than that of TL^*_s.

The only other work on learning of timed systems we are aware is by Verwer et al. [34], who present an algorithm for learning of timed automata with one clock which is reset at every transition. Their setting differs from ours in that the learner does not choose the words to be used in membership queries, but is given a sample of accepted and unaccepted timed words from which an automaton is to be constructed. The algorithm constructs a prefix tree from the sample of timed words and then tries to merge nodes of this tree pairwise to form an automaton. If the resulting automaton does not agree with the sample then the last merge is undone and a new merge is attempted. The algorithm does not construct timed automata in a systematic way, and it is hard to generalize the algorithm to timed automata with more than one clock.

Previous work on learning of infinite-state systems include the work by Berg et al. [8], who consider Mealy machines extended with state variables that can assume values from a potentially unbounded domain. These values can be passed as parameters in input and output messages, and can be used in tests for equality between state variables and/or message parameters. Their technique first uses the L^* algorithm to generate a finite-state Mealy machine for the case that the values are taken from a finite data domain. This finite-state Mealy machine is then transformed into a symbolic form, representing the desired infinite-state Mealy machine.

A problem related to automata learning (or regular inference) is that of conformance testing, where one is given an automaton specification of the intended behavior of an implementation, and would like to derive a test suite which checks that an implementation conforms to such a specification. In previous work [7], we showed that if a set of input words form a conformance test suite for a finite state machine, then that state machine can be inferred from this set of input words using automata learning techniques. Conversely, if a finite state machine is uniquely inferred from a set of input words which are supplied in membership queries, then this set of input words forms a conformance test suite for the state machine. Springintveld et al. [32] introduces an algorithm which generates a conformance test suite for timed automata. The algorithm constructs grid automata, which only contain states in which every clock value is from the set of integer multiples of 2^{-n} for some sufficiently large natural number n. Then the Vasilevskii–Chow algorithm [11,33] for generating conformance test suites for finite state machines, is applied to the grid automaton to generate a conformance test suite for the timed automaton.

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., [2,31]) and right-congruences of timed languages (e.g., [28,26,35]), motivated by testing and verification.

Structure of the paper. The paper is structured as follows. After preliminaries in the next section, we define deterministic event-recording automata (DERAs) in Section 3. In Section 4 we describe the L^* algorithm for learning DFAs. In Sections 5 and 7, we present the algorithms TL^*_{sg} and, respectively, TL^*_{nsg} for learning event-deterministic DERAs (EDERAs). In Section 6 we describe the algorithm TL^*_s suitable for learning general DERAs. Section 8 presents conclusions and directions for future research.

2. Preliminaries

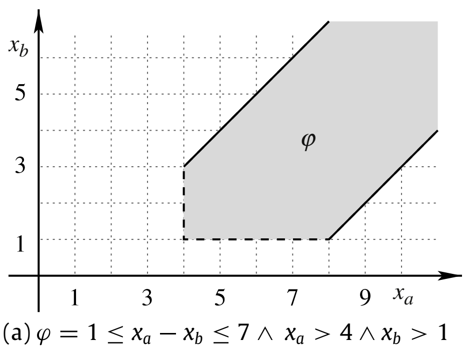
We write $\mathbb{R}^{\geq 0}$ for the set of nonnegative real numbers, and $\mathbb{N}$ for the set of natural numbers. Let Σ be a finite alphabet of size $|\Sigma|$. A *timed word* over Σ is a finite sequence $w_t = (a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ of symbols $a_i \in \Sigma$ that are paired with nonnegative real numbers t_i such that the sequence $t_1 t_2 \ldots t_n$ of time-stamps is nondecreasing. Each time-stamp t_i denotes the time of occurrence of the symbol a_i, measured from some common "initial moment". 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. We write C_Σ for the set $\{x_a \mid a \in \Sigma\}$ of event-recording clocks.

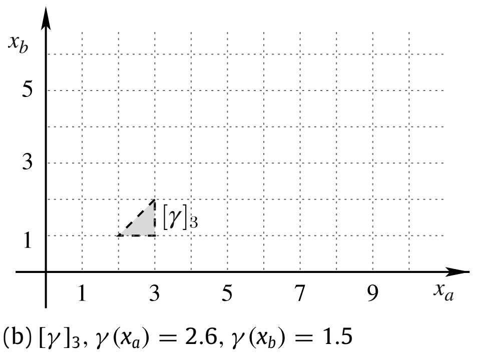
A *clock valuation* γ is a mapping from C_Σ to $\mathbb{R}^{\geq 0}$. For $a \in \Sigma$, we define $\gamma[x_a \mapsto 0]$ to be the clock valuation γ' such that $\gamma'(x_a) = 0$ and $\gamma'(x_b) = \gamma(x_b)$ for all $b \neq a$, $b \in \Sigma$. For $t \in \mathbb{R}^{\geq 0}$, we define $\gamma + t$ to be the clock valuation γ' such that $\gamma'(x_a) = \gamma(x_a) + t$ for all $a \in \Sigma$.

Throughout the paper, we will use an alternative, equivalent representation of timed words, namely clocked words. A *clocked word* w_c 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 for all $a \in \Sigma$ satisfies

- $\gamma_1(x_a) = \gamma_1(x_b)$ for all $a, b \in \Sigma$, and
- $\gamma_i(x_a) = \gamma_{i-1}(x_a) + \gamma_i(x_{a_{i-1}})$ whenever $1 < i \leq n$ and $a \neq a_{i-1}$. The quantity $\gamma_i(x_{a_{i-1}})$ is the time elapsed between the occurrence of a_{i-1} and a_i, and so can be arbitrary.



(a) $\varphi = 1 \le x_a - x_b \le 7 \wedge x_a > 4 \wedge x_b > 1$



(b) $[\gamma]_3$, $\gamma(x_a) = 2.6$, $\gamma(x_b) = 1.5$

Fig. 1. Clock constraint and region.

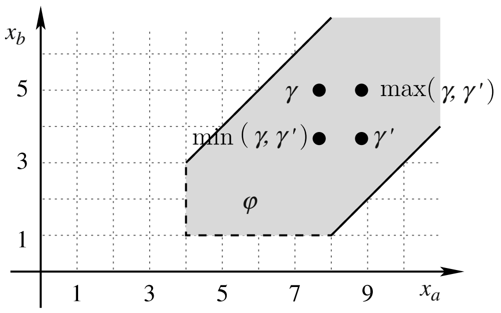


Fig. 2. Illustration of Proposition 1.

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

- $\gamma_i(x_a) = t_i$ if $a_j \ne a$ for $1 \le j < i$,
- $\gamma_i(x_a) = t_i - t_j$ if there is a j with $1 \le j < i$ such that $a_j = a$, and furthermore $a_k \ne a$ for $j < k < i$ (i.e., a_j is the most recent occurrence of a).

A *clock constraint* is a conjunction of atomic constraints of the form $x_a \sim n$, called a *clock bound*, or $x_a - x_b \sim n$, called a *difference bound*, for $x_a, x_b \in C_\Sigma$, $\sim \in \{<, \le, \ge >\}$, and $n \in \mathbb{N}$. A clock constraint is called *non-strict* if only $\sim \in \{\le, \ge\}$ is used, and, similarly, it is called *strict* if only $\sim \in \{<, >\}$ is used. For example, $\varphi = x_a - x_b \ge 1 \wedge x_a - x_b \le 7 \wedge x_a > 4 \wedge x_b > 1$ is a clock constraint, which is neither strict nor non-strict. We identify an empty conjunction with *true*.

We use $\gamma \models \varphi$ to denote that the clock valuation γ satisfies the clock constraint φ, defined in the usual manner. A clock constraint φ *identifies* a $|\Sigma|$-dimensional *polyhedron* $[\![\varphi]\!] \subseteq (\mathbb{R}^{\ge 0})^{|\Sigma|}$ viz. the vectors of real numbers satisfying the constraint. In Fig. 1(a), a clock constraint and the 2-dimensional polyhedron it identifies are shown.

For each clock constraint φ there are in general several other clock constraints that are equivalent to φ in the sense that they identify the same polyhedron. If φ is satisfiable, there is among these a unique *canonical* clock constraint, denoted by $Can(\varphi)$, obtained by closing φ under all consequences of pairs of conjuncts in φ, i.e.,

- from two difference bounds, such as $x_a - x_b \le 2$ and $x_b - x_c < 3$, we derive a new difference bound, viz. $x_a - x_c < 5$, and
- from a difference bound and a clock bound, such as $x_a - x_b \le 2$ and $x_a \ge 3$, we derive a new clock bound, viz. $x_b \ge 1$,
- from an upper and a lower clock bound, such as $x_a \le 3$ and $x_b > 2$, we derive a new difference bound, viz. $x_a - x_b < 1$,

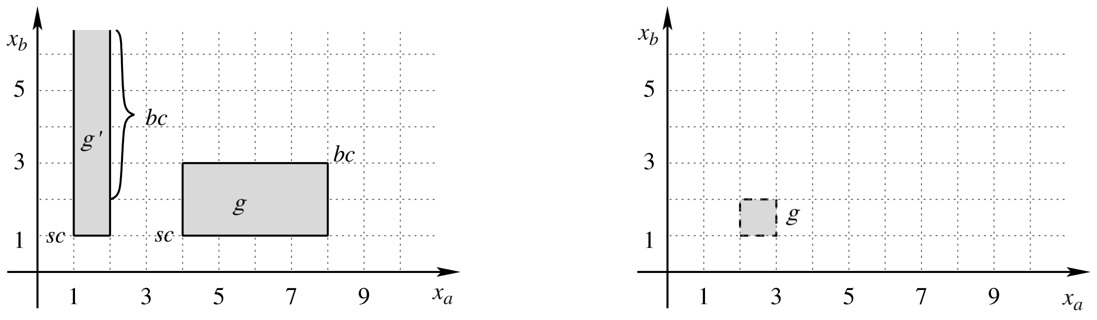
until saturation, and thereafter keeping the tightest bounds for each clock and each clock difference. If the canonical form contains inconsistent constraints, or requires some clock to be negative, then the clock constraint is unsatisfiable. The canonical form for an unsatisfiable clock constraint is defined to be *false* [14].

Clock constraints satisfy an important closure property:

Proposition 1. *For a clock constraint φ and two clock valuations γ, γ', if $\gamma \models \varphi$ and $\gamma' \models \varphi$, then $\min(\gamma, \gamma') \models \varphi$ and $\max(\gamma, \gamma') \models \varphi$, where $\min(\gamma, \gamma')$ is defined by $\min(\gamma, \gamma')(x_a) = \min(\gamma(x_a), \gamma'(x_a))$ for all $a \in \Sigma$, and analogously for $\max(\gamma, \gamma')$ (see Fig. 2).*

Proof. That $\min(\gamma, \gamma')$ satisfies a clock bound of form $x_a \sim n$ follows from the fact that $\min(\gamma, \gamma')(x_a)$ is either $\gamma(x_a)$ or $\gamma'(x_a)$, and that both γ and γ' satisfy $x_a \sim n$. To see that $\min(\gamma, \gamma')$ satisfies a difference bound of form $x_a - x_b \ge n$, assume that $\min(\gamma, \gamma')(x_a)$ is $\gamma(x_a)$. Then

$$\min(\gamma, \gamma')(x_a) - \min(\gamma, \gamma')(x_b) = \gamma(x_a) - \min(\gamma, \gamma')(x_b) \ge \gamma(x_a) - \gamma(x_b),$$



(a) $g = 4 \leq x_a \leq 8 \wedge 1 \leq x_b \leq 3 g' = 1 \leq x_a \leq 2 \wedge x_b \geq 1$ (b) $g = 2 < x_a < 3 \wedge 1 < x_b < 2$

Fig. 3. Guard and simple guard.

and by assumption $\gamma(x_a) - \gamma(x_b) \geq n$. The proof is analogous for difference bounds of form $x_a - x_b \leq n$ and for strict difference bounds. □

A *clock guard* is a conjunction of atomic constraints of the form $x_a \sim n$, for $x_a \in C_\Sigma$, $\sim \in \{<, \leq, \geq, >\}$, and $n \in \mathbb{N}$, i.e., comparison between clocks is not permitted. A clock guard is called *non-strict* if only $\sim \in \{\leq, \geq\}$ is used and *strict* if only $\sim \in \{<, >\}$ is used. For example, $x_a \geq 4 \wedge x_a \leq 8 \wedge x_b \geq 1 \wedge x_b \leq 3$ is a clock guard, which is non-strict.

The set of clock guards is denoted by G_Σ. A clock guard g *identifies* a $|\Sigma|$-dimensional *hypercube* $[\![g]\!] \subseteq (\mathbb{R}^{\geq 0})^{|\Sigma|}$. In Fig. 3(a), two different clock guards g and g' and two 2-dimensional hypercubes — rectangles — they identify are shown: the bounded one for g and the partially unbounded one for g'. We use equalities in clock constraints and clock guards in the natural way, e.g., $x_a = n$ denotes $x_a \geq n \wedge x_a \leq n$. In timed automata, guards are restricted to being clock guards: one reason is that this is sufficient for many applications, another is allowing them to be general clock constraints would make analysis algorithms more complicated (see e.g., [10]).

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 timed word w_t we use $w_t \models w_g$ to denote that $CW(w_t) \models w_g$. A guarded word $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ is called a *guard refinement* of $a_1 a_2 \ldots a_n$, and $a_1 a_2 \ldots a_n$ is called the word *underlying* w_g. The word w *underlying* a timed word w_t is defined in a similar manner. A guarded word $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ is non-strict if g_i is non-strict for all $1 \leq i \leq n$.

A clock constraint or a clock guard is *K-bounded* if it contains no constant larger than K. A K-bounded *simple clock guard* is a clock guard whose conjuncts are only of the form $x_a = n, n' < x_a \wedge x_a < n' + 1$ or $x_a > K$, for $0 \leq n \leq K, 0 \leq n' \leq K - 1$, $x_a \in C_\Sigma$. In Fig. 3(b), an example of a simple clock guard is shown. A *K-bounded simple guarded word* w_g 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 K-bounded simple clock guards.

The *extracted guard* from a clock constraint φ, denoted *guard*(φ) is the conjunction of all clock bounds (i.e., conjuncts of form $x_a \sim n$) in $Can(\varphi)$. In simple words, *guard*(φ) identifies the smallest hypercube surrounding the polyhedron identified by φ. If φ is unsatisfiable, *guard*(φ) is defined as *false*. This intuition immediately leads to the following proposition.

Proposition 2. *Let φ be a clock constraint and g be a clock guard. Then $[\![\varphi \wedge g]\!] = [\![\varphi]\!]$ implies $[\![guard(\varphi)]\!] \subseteq [\![g]\!]$.* □

For the developments to come, we define several operations on clock constraints φ.

- We define the *reset* of a clock x_a in φ, denoted by $\varphi[x_a \mapsto 0]$, as $Can(\varphi')$, where φ' is obtained from $Can(\varphi)$ by removing all conjuncts involving x_a, and adding the conjunct $x_a \leq 0$.
- We define the *time elapsing* of φ, denoted by $\varphi \uparrow$, as $Can(\varphi')$, where φ' is obtained from $Can(\varphi)$ by removing all upper bounds on clocks [15].

It is a standard result that these operations mirror the corresponding operations on clock valuations, in the sense that

- $\gamma' \models \varphi[x_a \mapsto 0]$ iff $\gamma' = \gamma[x_a \mapsto 0]$ for some γ with $\gamma \models \varphi$, and
- $\gamma' \models \varphi \uparrow$ iff $\gamma' = \gamma + t$ for some γ with $\gamma \models \varphi$ and $t \in \mathbb{R}^{\geq 0}$.

Following [15], we introduce the $K^<$-approximation $\langle\langle \varphi \rangle\rangle_K^<$ of the clock constraint φ as the constraint φ' obtained from $Can(\varphi)$ by

- removing all constraints of the form $x_a \sim n$ and $x_a - x_b \sim n$, whenever $\sim \in \{<, \leq\}$ and $n > K$, and
- replacing all constraints of the form $x_a \sim n$ and $x_a - x_b \sim n$ by $x_a > K$ and $x_a - x_b > K$, respectively, whenever $\sim \in \{>, \geq\}$ and $n > K$.

Note that the $K^<$-approximation of a canonical clock constraint is in general not canonical.

We introduce the $K^{\leq}$-approximation $\langle\langle g \rangle\rangle_K^{\leq}$ of the clock guard g as the clock guard obtained by

- removing all constraints of form $x_a \leq n$ and $x_a < n$, whenever $n > K$, and
- replacing all constraints of form $x_a \geq n$ and $x_a > n$ by $x_a \geq K$, whenever $n \geq K$.

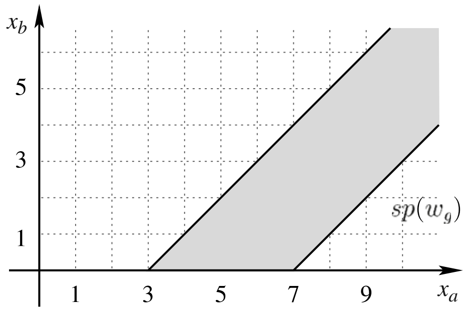


Fig. 4. $w_g = (a, true)(b, 2 \leq x_a \leq 4)(b, 1 \leq x_b \leq 3)$.

For a constraint φ and guarded word w_g, we introduce the *strongest postcondition* of w_g with respect to φ, denoted by $sp(\varphi, w_g)$. Postcondition computation is central in symbolic verification techniques for timed automata [6,9], and can be done inductively as follows:

- $sp(\varphi, \lambda) = \varphi$,
- $sp(\varphi, w_g(a, g)) = ((sp(\varphi, w_g) \wedge g)[x_a \mapsto 0])\uparrow$.

We often omit the first argument in the postcondition, implicitly assuming it to be the initial constraint $\varphi_0 = \bigwedge_{a,b\in\Sigma} x_a = x_b$ (or *true* if Σ has only one symbol), i.e., $sp(w_g) = sp(\varphi_0, w_g)$. Intuitively, $sp(w_g)$ is the constraint on clock valuations that is induced by w_g on any following occurrence of a clock valuation, i.e., $\gamma \models sp(w_g)$ if and only if there is a clocked word $w_c(a, \gamma)$ such that $w_c \models w_g$. We remark that the polyhedron identified by the strongest postcondition is a convex set [15], and that $sp(w_g)$ is non-strict if w_g is non-strict.

In Fig. 4, an example of the strongest postcondition for the guarded word $w_g = (a, true)(b, 2 \leq x_a \leq 4)(b, 1 \leq x_b \leq 3)$ is shown. Intuitively, taking a resets clock x_a. The two subsequent b-actions can only be taken between $2 + 1$ and $4 + 3$ time units later and do not reset x_a. As b is the last action taken in the word, there is no constraint on x_b.

For a guarded word w_g, we also introduce the $K^<$-approximated postcondition $sp^<_K(w_g)$, defined by

- $sp^<_K(\lambda) = \bigwedge_{a,b\in\Sigma} x_a = x_b$
- $sp^<_K(w_g(a, g)) = \langle\langle sp(sp^<_K(w_g), (a, g))\rangle\rangle^<_K$.

Given a natural number K, we define the *region equivalence* $\sim_K$ on the set of clock valuations by $\gamma \sim_K \gamma'$ if

- for all $x_a \in C_\Sigma$, either
  . $\gamma(x_a)$ and $\gamma'(x_a)$ are both greater than K, or
  . $\lfloor\gamma(x_a)\rfloor = \lfloor\gamma'(x_a)\rfloor$ and $fract(\gamma(x_a)) = 0$ iff $fract(\gamma'(x_a)) = 0$,

  and
- for all $x_a, x_b \in C_\Sigma$ with $\gamma(x_a) \leq K$ and $\gamma(x_b) \leq K$,
  $fract(\gamma(x_a)) \leq fract(\gamma(x_b))$ iff $fract(\gamma'(x_a)) \leq fract(\gamma'(x_b))$.

A *region* is an equivalence class of clock valuations induced by $\sim_K$. We denote by $[\gamma]_K$ the region of γ. In Fig. 1(b), an example of a region is depicted.

A clock constraint identifies the union of a set of regions, therefore region equivalence induces a natural equivalence on clock constraints. For two clock constraints φ and φ', define $\varphi \approx_K \varphi'$, if for each clock valuation γ with $\gamma \vDash \varphi$ there is a clock valuation γ' with $\gamma' \vDash \varphi'$ such that $\gamma \sim_K \gamma'$, and vice versa.

An important property of region equivalence is that it is preserved by reset and time elapsing operations. If $\gamma \sim_K \gamma'$ then $\gamma[x_a \mapsto 0] \sim_K \gamma'[x_a \mapsto 0]$, and for each $t \in \mathbb{R}^{\geq 0}$ there is a $t' \in \mathbb{R}^{\geq 0}$ such that $\gamma + t \sim_K \gamma' + t'$ [36]. If we combine this fact with the fact that the reset and time elapsing operations on constraints mirror the same operations on clock valuations, we infer that the relation $\approx_K$ on clock constraints is preserved by reset and time elapsing. Thus, if $\varphi \approx_K \varphi'$ then $\varphi[x_a \mapsto 0] \approx_K \varphi'[x_a \mapsto 0]$, and $\varphi\uparrow \approx_K \varphi'\uparrow$. The relation $\approx_K$ is also preserved by approximation, i.e., $\varphi \approx_K \langle\langle\varphi\rangle\rangle^<_K$. A corollary of these facts is the following lemma.

Lemma 3. *Let w_g be a guarded word. Then*

$$sp^<_K(w_g) \approx_K sp(w_g).$$

Proof. By induction on the length of w_g. For the base case, where w_g is empty, the proof is immediate. For the inductive step, assume that $sp^<_K(w_g) \approx_K sp(w_g)$ and consider the guarded word $w_g(a, g)$. By definition, $sp^<_K(w_g(a, g)) = \langle\langle sp(sp^<_K(w_g), (a, g))\rangle\rangle^<_K$. From $sp^<_K(w_g) \approx_K sp(w_g)$ we infer, using that $\approx_K$ is preserved by reset, time elapsing, and conjunction with clock guards, that $sp(sp^<_K(w_g), (a, g)) \approx_K sp(w_g(a, g))$. Since $\approx_K$ is preserved by approximation, we infer that $\langle\langle sp(sp^<_K(w_g), (a, g))\rangle\rangle^<_K \approx_K sp(sp^<_K(w_g), (a, g)) \approx_K sp(w_g(a, g))$, i.e., that $sp^<_K(w_g(a, g)) \approx_K sp(w_g(a, g))$. □

For every satisfiable non-strict clock guard g, we define its *K-smallest corner*, denoted by $sc_K(g)$, as the set of clock valuations γ that satisfy $\gamma(x_a) = n$ for all x_a such that the lower bound for x_a in g is of the form $n \leq x_a$ with $n < K$, and that satisfies $\gamma(x_a) \geq K$ whenever the lower bound for x_a in g is of the form $n \leq x_a$ with $n \geq K$. Similarly, we define the *biggest corner* of g, denoted $bc_K(g)$ as the set of valuations γ that are maximal in the dimensions where $[\![g]\!]$ has an upper bound and exceeds K in the others. In Fig. 3(a), for example, if $K = 8$, the biggest corner of guard g contains the only valuation γ with $\gamma(x_a) = 8$ and $\gamma(x_b) = 3$, while for g' if $K = 2$, the biggest corner contains all valuations γ with $\gamma(x_a) = 2$ and $\gamma(x_b) > 2$.

3. Event-recording automata

In this section, we introduce event-recording automata, which are the subject of the learning algorithms in this paper. We also introduce the further restricted class of event-deterministic event-recording automata, which the algorithms TL^*_{sg} and TL^*_{nsg} are designed to learn. In the treatment, we will repeatedly make use of standard deterministic finite automata.

A *deterministic finite automaton* (DFA) $\mathcal{A} = (\Gamma, Q, q_0, \delta, Q^f)$ over the alphabet Γ consists of a finite set of *states* Q, and *initial state* q_0, a partial *transition function* $\delta : Q \times \Gamma \to Q$, and a set of *final states* $Q^f \subseteq Q$. A *run* of $\mathcal{A}$ over the word $w = a_1 a_2 \ldots a_n$ is a finite sequence

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$$

of states $q_i \in Q$ such that q_0 is the initial state and $\delta(q_{i-1}, a_i)$ is defined for $1 \leq i \leq n$, with $\delta(q_{i-1}, a_i) = q_i$. In this case, we write $\delta(q_0, w) = q_n$, thereby extending the definition of δ to words in the natural way. The run is called *accepting* if $q_n \in Q^f$. The language $\mathcal{L}(\mathcal{A})$ comprises all words $a_1 a_2 \ldots a_n$ over which an accepting run exists.

We are now ready to introduce the class of automata models whose objects we want to learn: deterministic event-recording automata.

Definition 4. An *event-recording automaton* (ERA) $D = (\Sigma, L, l_0, \delta, L^f)$ 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, and a *transition function* $\delta : L \times \Sigma \times G_\Sigma \to 2^L$, which is a partial function with finite support that for each location, input symbol and guard potentially prescribes a set of target locations. An ERA is *deterministic* iff

- $\delta(l, a, g)$ is a singleton set whenever it is defined, and
- whenever $\delta(l, a, g_1)$ and $\delta(l, a, g_2)$ are both defined then $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$. □

Thus, for a deterministic ERA, a location l might have two different a successors, which, however, have nonoverlapping guards. Due to the first restriction, we will consider δ to be of type $\delta : L \times \Sigma \times G_\Sigma \to L$, i.e., to map each triple in its domain to a single location rather than a set. An ERA is *K-bounded* if the guard g is K-bounded whenever $\delta(l, a, g)$ is defined.

In this paper, we only consider deterministic ERAs, or DERAs for short, which is no significant restriction in terms of expressiveness as every ERA can be transformed into a DERA accepting the same language. For details, see [2].

In order to define the language accepted by a DERA, we first understand it as a DFA, which accepts guarded words.

Given a DERA $D = (\Sigma, L, l_0, \delta, L^f)$, we define $dfa(D)$ to be the DFA $\mathcal{A}_D = (\Gamma, L, l_0, \delta', L^f,)$ over the alphabet $\Gamma = \Sigma \times G_\Sigma$ 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 DERAs over Σ to DFAs over $\Sigma \times G_\Sigma$ is injective, meaning that for each DFA $\mathcal{A}$ over $\Sigma \times G_\Sigma$, there is a unique (up to isomorphism) ERA over Σ, denoted $era(\mathcal{A})$, such that $dfa(era(\mathcal{A}))$ is isomorphic to $\mathcal{A}$.

The language $\mathcal{L}(D)$ accepted by a DERA 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 DERAs 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 $D_1 \equiv D_2$.

We introduce a restricted class of deterministic ERAs, which the algorithms TL^*_{sg} and TL^*_{nsg} are designed to learn. The restriction is that each state has at most one outgoing transition per action. This means that timing constraints for the occurrence of an action depend only on the past sequence of actions, and not on their relative timing.

Definition 5. An ERA $(\Sigma, L, l_0, \delta, L^f)$ is called *event-deterministic* (EDERA), if

- only non-strict guards are used,
- for every $l \in L$ and $a \in \Sigma$ there is at most one $g \in G_\Sigma$ such that $\delta(l, a, g)$ is defined, and
- every location is accepting. □

In case of an EDERA, its transition function $\delta : L \times \Sigma \times G_\Sigma \to L$ can be understood as two functions; $\eta : L \times \Sigma \to G_\Sigma$, which for a location and an input symbol prescribes a guard, and $\varrho : L \times \Sigma \to L$, which for a location and an input symbol prescribes a target location. Thus, we also use $D = (\Sigma, L, l_0, \varrho, \eta)$ to denote an EDERA, where L^f is omitted since $L^f = L$.

From the above definitions, we see that the language of an EDERA D can be characterized by a prefix-closed set of guarded words $(a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ in $\mathcal{L}(dfa(D))$ such that each $a_1 a_2 \ldots a_n$ is a word underlying at most one $w_g \in \mathcal{L}(dfa(D))$. Thus, we can loosely say that D imposes on each untimed word $a_1 a_2 \ldots a_n$ the timing constraints represented by the guards $g_1 g_2 \cdots g_n$.

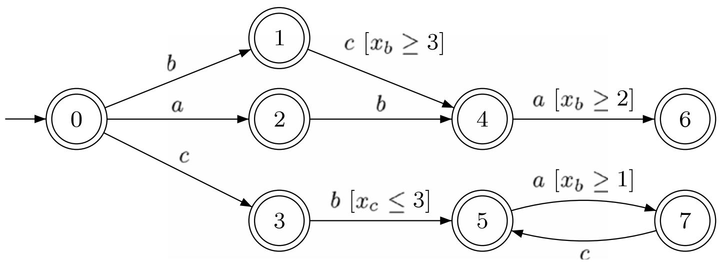


Fig. 5. An event-recording automaton.

Example 6. The event-recording automaton shown in Fig. 5 over the alphabet $\{a, b, c\}$ uses three event-recording clocks, x_a, x_b, and x_c. It is event deterministic, as all guards are non-strict and no location has two outgoing edges labelled with the same action. Location 0 is the initial location of the automaton. The clock constraint $x_b \geq 3$ that is associated with the edge from location 1 to 4 ensures that the action c can only be taken at least three time units after taking the transition from 0 to 1. This also implies that the time difference between the first b and the subsequent a is greater than or equal to 3. □

4. The L^* algorithm for learning DFAs

In this section, we shortly review the L^* algorithm, due to Angluin [4] for learning a regular (untimed) language, $\mathcal{L}(\mathcal{A}) \subseteq \Gamma^*$, accepted by a minimal deterministic finite automaton (DFA) $\mathcal{A} = (\Gamma, Q, q_0, \delta, Q^f)$. 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, which is a word 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 $T(u) : V \mapsto \{+, -\}$. The interpretation of T is that for $u \in (U \cup U\Gamma)$ and $v \in V$ we have $T(u)(v) = +$ if $uv \in \mathcal{L}(\mathcal{A})$ and $T(u)(v) = -$ if $uv \notin \mathcal{L}(\mathcal{A})$. In [4], each function $T(u)$ is called a *row*. Thus two rows, $T(u)$ and $T(u')$, are equal, denoted $T(u) = T(u')$, if $T(u)(v) = T(u')(v)$ for all $v \in V$. Table T is

- *closed*, if for each $u \in U$ and $a \in \Gamma$ there is a $u' \in U$ such that $T(ua) = T(u')$, and
- *consistent*, if, for each $u, u' \in U$, $T(u) = T(u')$ implies $T(ua) = T(u'a)$.

If T is not closed, we find $u' \in U\Gamma$ such that $T(u) \neq T(u')$ for all $u \in U$. Then we move u' to U and ask membership queries for every $u'av$ where $a \in \Gamma$ and $v \in V$. If T is not consistent, we find $u, u' \in U$, $a \in \Gamma$ and $v \in V$ such that $T(u) = T(u')$ and $T(ua)(v) \neq T(u'a)(v)$. Then we add av to V and ask membership queries for every uav where $u \in U \cup U\Gamma$. Checks whether T is closed and consistent can be done in any ordering. When T is closed and consistent the *Learner* constructs a hypothesized DFA $\mathcal{H} = (\Gamma, L, l_0, \delta, L^f)$, where

- $L = \{T(u) \mid u \in U\}$ is the set of distinct rows,
- l_0 is the row $T(\lambda)$,
- δ is defined by $\delta(T(u), a) = T(ua)$, and
- $L^f = \{T(u) \mid u \in U \text{ and } T(u)(\lambda) = +\}$ is the set of rows which are accepting without adding a suffix,

and submits $\mathcal{H}$ in an equivalence query. If the answer is *yes*, the learning procedure is completed. Otherwise the returned counterexample w is processed by adding every prefix of w (including w) to U, and subsequent membership queries are performed in order to make the table closed and consistent, after which a new hypothesized DFA is constructed, etc.

The L^* algorithm constructs $\mathcal{A}$ after asking $O(kn^2m)$ membership queries and at most n equivalence queries, where n is the number of states in $\mathcal{A}$, k is the size of the alphabet and m is the length of the longest counterexample [4]. 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 is the number of columns.

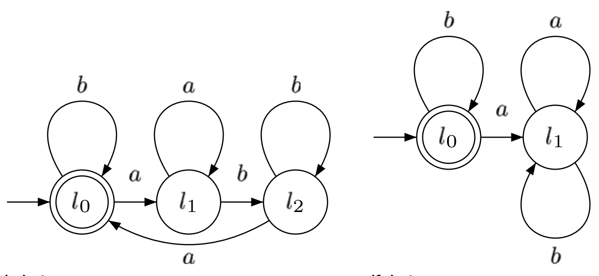
A description of the L^* algorithm is given as Algorithms 1 and 2, using Java-style pseudocode. Since membership queries and equivalence queries can be implemented in different ways and also differ in timed and untimed settings, we introduce the interface *Teacher* which contains two functions that are responsible for membership and equivalence queries (see Algorithm 1). Angluin's algorithm is given as function *Learner* of class L^* (see lines 10–21 in Algorithm 2). The function *Learner* first constructs an initial table by calling the function *initialize* and then constructs hypothesized automata until the answer to an equivalence query is *yes*. Since each hypothesized automaton has to be constructed from a closed and consistent table, function *Learner* checks these properties by calling functions *isClosed* and *isConsistent*. If the table is not consistent, the function *add_column* is called, which adds a distinguishing suffix to V. If the table is not closed, the function *move_row* is called which moves the corresponding row ua to U. When a hypothesized automaton is constructed, an equivalence query is performed and if a counterexample is obtained the function *process_counterexample* is called.

Algorithm 1 Interface of Teacher

```
interface Teacher{
   Function membership_query($u$)
   Function equivalence_query($\mathcal{H}$)
}
```

Algorithm 2 Pseudo code for Angluin's Learning Algorithm

```
class L*{

Teacher teacher
Alphabet $\Gamma$

Constructor $L^*(t,\Sigma)$
  $teacher = t$
  $\Gamma = \Sigma$

Function Learner()
  initialize $(U, V, T)$
  repeat
        while not(isClosed($(U, V, T)$)) or not(isConsistent($(U, V, T)$))
            if not(isConsistent($(U, V, T)$)) then add_column()
            if not(isClosed($(U, V, T)$)) then move_row()
        Construct hypothesized automaton $\mathcal{H}$
        teacher.equivalence_query($\mathcal{H}$)
        if the answer to equivalence query is a counterexample $u$ then
            process_counterexample(u)
  until the answer to equivalence query is 'yes' to the hypothesis $\mathcal{H}$
  return $\mathcal{H}$.

Function initialize($U, V, T$)
  $U := \{\lambda\}, V := \{\lambda\}$
  $T(\lambda)(\lambda)$=teacher.membership_query($\lambda$)
  for every $a \in \Gamma$
      $T(a)(\lambda)$=teacher.membership_query($a$)

Function isClosed()
  if for each $u \in U$ and $a \in \Gamma$ there is $u' \in U$ with $T(ua) = T(u')$
      return true
  else
      return false

Function isConsistent()
  if for each $a \in \Gamma$ and $u, u' \in U$ such that $T(u) = T(u')$ we have $T(ua) = T(u'a)$
      return true
  else
      return false

Function add_column()
  Find $a \in \Gamma$, $v \in V$ and $u, u' \in U$ such that $T(u) = T(u')$ and $T(ua)(v) \neq T(u'a)(v)$
  Add $av$ to $V$
  for every $u \in U \cup U\Gamma$
      $T(u)(av)$=teacher.membership_query($uav$)

Function move_row()
  Find $u \in U$, $a \in \Gamma$ such that $T(ua) \neq T(u')$ for all $u' \in U$
  Move $ua$ to $U$
  for every $a' \in \Gamma$ and $v \in V$
      $T(uaa')(v)$=teacher.membership_query($uaa'v$)

Function process_counterexample(u)
  Add every prefix $u'$ of $u$ to $U$
  for every $a \in \Gamma$, $v \in V$ and prefix $u'$ of $u$
      $T(u')(v)$=teacher.membership_query($u'v$)
      $T(u'a)(v)$=teacher.membership_query($u'av$)
}
```



(a) An automaton $\mathcal{A}$ (b) An automaton $\mathcal{H}$

Fig. 6. An automaton to be learned and a hypothesized automaton.

T_1	λ
λ	+
a	-
b	+

(a) Table T_1

T_2	λ
λ	+
a	-
b	+
aa	-
ab	-

(b) Table T_2

T_3	λ
λ	+
a	-
ab	-
aba	+
b	+
aa	-
abb	-
abaa	-
abab	+

(c) Table T_3

T_4	λ	*a*
λ	+	-
a	-	-
ab	-	+
aba	+	-
b	+	-
aa	-	-
abb	-	+
abaa	-	-
abab	+	-

(d) Table T_4

Fig. 7. Observation tables.

Let us consider an example of the L^* algorithm. Let $\mathcal{A}$ be the DFA shown in Fig. 6(a). Initially, the *Learner* asks membership queries for λ, a and b. The initial observation table T_1 is shown in Fig. 7(a), where $U = V = \{\lambda\}$. This observation table is consistent, but not closed, since $T(a) \neq T(\lambda)$. The *Learner* moves the prefix a to U and then asks membership queries for aa and ab to construct the observation table T_2 shown in Fig. 7(b). This observation table is closed and consistent. The *Learner* constructs the hypothesized automaton $\mathcal{H}$ shown in Fig. 6(b) and asks an equivalence query to the *Teacher*. Assume that the *Teacher* replies with the counterexample aba, which is in $\mathcal{L}(\mathcal{A})$ but not accepted by $\mathcal{H}$. To process the counterexample aba, the *Learner* adds ab and aba to U and asks membership queries for abb, $abaa$ and $abab$ to construct the observation table T_3 shown in Fig. 7(c). This observation table is not consistent since $T(a) = T(ab)$ but $T(aa) \neq T(aba)$. Then the *Learner* adds a to E and asks membership queries for ba, aaa, $abba$, $abaaa$ and $ababa$ to construct the observation table T_4 shown in Fig. 7(d) . This observation table is closed and consistent. The *Learner* constructs the automaton shown in Fig. 6(a) and asks an equivalence query to the *Teacher*. The *Teacher* replies *yes* and L^* terminates.

As we will see later, the general scheme of this algorithm stays the same in our algorithms for learning timed languages. However, the initialization of the table, queries, closedness and consistency checks become different.

5. Learning event-deterministic ERAs

In this section, we present the algorithm TL^*_{sg} for learning EDERAs, obtained by adapting the L^* algorithm. A central idea in the L^* algorithm is to let each state be identified by the words that reach it from the initial state (such words are called *access strings* in [5]). States are congruent if, according to the queries submitted so far, the same continuations of their access strings are accepted. This idea is naturally based on the properties of Nerode's right congruence (given a language L, two words $u, v \in \Sigma^*$ are equivalent if for all $w \in \Sigma^*$ we have $uw \in L$ iff $vw \in L$) which implies that there is a unique minimal DFA accepting L. In other words, for DFAs, every state can be characterized by the set of words accepted by the DFA when considering this state as an initial state, and every string leads to a state in a unique way.

For timed languages, it is not obvious how to generalize Nerode's right congruence.[1] In general there is no unique minimal DERA which is equivalent to a given DERA. As an example, consider Fig. 5, assuming for a moment that the c-transition from location 7 to 5 is missing. Then the language of the automaton does not change when changing the transition from 1 into 4 to 1 into 5, although the language accepted from 4 is different from the one from 5. Furthermore, if we do not modify the automaton in Fig. 5 we can reach location 4 by two guarded words: $(b, true)(c, x_b \geq 3)$ as well as $(a, true)(b, true)$. Although they lead to the same location, they admit different continuations of event-clock words: action a can be performed with $x_b = 2$ after $(a, true)(b, true)$ but not after $(b, true)(c, x_b \geq 3)$. The complication is that each guarded word imposes a postcondition, which constrains the values of clocks that are possible at the occurrence of future actions.

Our approach to overcoming the problem that DERAs have no canonical form is to define a subclass of EDERAs which do have a canonical form, and which furthermore can be understood as a DFA over $\Sigma \times G_\Sigma$ where G_Σ is the set of clock

[1] See [28] for a study on right-congruences on timed languages.

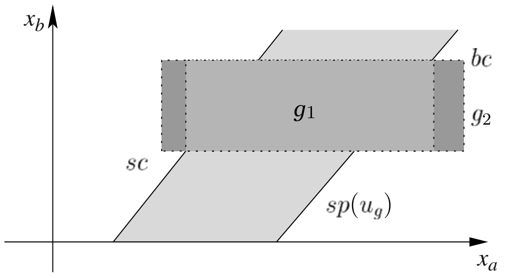


Fig. 8. An illustration of Definition 7.

guards. We can then use Angluin's algorithm to learn this DFA, and thereafter interpret the result as an EDERA. In the next section, we define this canonical form, called *sharply guarded* EDERA, and prove that any EDERA can be transformed to this canonical form. We can therefore use Angluin's algorithm to learn a DFA over $\Sigma \times G_\Sigma$. A problem is that membership queries will ask whether a guarded word is accepted by the DFA, whereas the EDERA to be learned answers only queries for timed words. We therefore extend the *Learner* in Angluin's algorithm by an *Assistant*, whose role is to answer a membership query for a guarded word, posed by the *Learner*, by asking several membership queries for timed words to the (timed) *Teacher*. We describe the operation of the *Assistant* in Section 5.2. Thereafter, in Section 5.3 we present the complete algorithm for learning EDERAs.

5.1. Sharply guarded EDERAs

Motivated by the previous discussion, in this section we define a class of EDERAs that admit a natural definition of right congruences.

Definition 7. A K-bounded EDERA D is *sharply guarded* if for all guarded words $w_g(a, g) \in \mathcal{L}(dfa(D))$, we have that $sp_K^<(w_g) \wedge g$ is satisfiable and

$$g = \left\langle\!\!\left\langle \bigwedge \{g' \in G_\Sigma \mid [\![sp_K^<(w_g) \wedge g]\!] = [\![sp_K^<(w_g) \wedge g']\!]\} \right\rangle\!\!\right\rangle_K^{\le}. \quad \square$$

Note that the conjunction is taken over all clock guards g', i.e., also those that are not K-bounded. Fig. 8 illustrates Definition 7 by a postcondition $sp(u_g) = x_a - x_b \ge 2 \wedge x_a - x_b \le 8$ together with a "sharp" guard $g_1 = x_a \ge 4 \wedge x_a \le 12 \wedge x_b \ge 2 \wedge x_b \le 4$ and a "non-sharp" guard $g_2 = x_a \ge 3 \wedge x_a \le 15 \wedge x_b \ge 2 \wedge x_b \le 4$. EDERA D such that $u_g(a, g_2) \in \mathcal{L}(dfa(D))$ and $K = 15$ is not sharply guarded, since $[\![sp_K^<(u_g) \wedge g_2]\!] = [\![sp_K^<(u_g) \wedge g_1]\!]$, where $[\![g_1]\!] \subset [\![g_2]\!]$. As shown in Fig. 8, there is no guard g' such that $[\![g']\!] \subset [\![g_1]\!]$ and $[\![sp(u_g) \wedge g']\!] = [\![sp(u_g) \wedge g_2]\!]$. Intuitively, an EDERA D is sharply guarded if the outgoing transitions from a location have guards which cannot be strengthened without changing the timing conditions under which the next symbol a will be accepted. Thus the upper and lower bounds on clock values in a clock valuation γ constraining the occurrence of a do not depend implicitly on the postcondition of the previous sequence of transitions taken by D. Thus we avoid the complications induced by postconditions described in the beginning of this section.

The following lemma gives a simpler characterization of being sharply guarded. Moreover, it shows that the use of approximation in Definition 7 does not affect the definition, but gives an a priori bound on the size of sharply guarded automata that accept a given timed language.

Lemma 8. *Let g be a non-strict K-bounded clock guard, and let φ be a K-bounded clock constraint. Then the following clock guards are equal:*

(a) $\langle\langle \bigwedge \{g' \in G_\Sigma \mid [\![\varphi \wedge g]\!] = [\![\varphi \wedge g']\!]\} \rangle\rangle_K^{\le}$,
(b) $\langle\langle \bigwedge \{g' \in G_\Sigma \mid [\![\varphi \wedge g]\!] \subseteq [\![g']\!]\} \rangle\rangle_K^{\le}$,
(c) $\langle\langle guard(\varphi \wedge g) \rangle\rangle_K^{\le}$.

In the following, we will mostly use characterization (c) when reasoning about sharply guarded EDERAs.

Proof. We first prove that (a) and (b) are equal. We observe that $[\![\varphi \wedge g]\!] \subseteq [\![g']\!]$ implies $[\![\varphi \wedge g]\!] \subseteq [\![\varphi \wedge g']\!]$, and that $[\![\varphi \wedge g]\!] = [\![\varphi \wedge g']\!]$ implies $[\![\varphi \wedge g]\!] \subseteq [\![g']\!]$. It therefore suffices to prove that we get the same result when using $[\![\varphi \wedge g]\!] \subseteq [\![\varphi \wedge g']\!]$ as when using $[\![\varphi \wedge g]\!] = [\![\varphi \wedge g']\!]$ as the condition on g' in the large conjunction. This follows by observing that for each guard g' such that $[\![\varphi \wedge g]\!] \subseteq [\![\varphi \wedge g']\!]$, there is a guard g'' such that $[\![\varphi \wedge g]\!] = [\![\varphi \wedge g'']\!]$, namely $g'' = g \wedge g'$, and that the conjunction of these g'' is the same as the conjunction of all g'.

We then prove that (b) and (c) are equal. Since $[\![\varphi \wedge g]\!] \subseteq [\![Can(\varphi \wedge g)]\!] \subseteq [\![guard(Can(\varphi \wedge g))]\!]$ we infer $[\![\varphi \wedge g]\!] \subseteq [\![guard(Can(\varphi \wedge g))]\!]$, from which it follows that (b) is included in (c). Conversely, for any guard g' we have that $[\![\varphi \wedge g]\!] \subseteq [\![g']\!]$ implies that $[\![guard(Can(\varphi \wedge g))]\!] \subseteq [\![g']\!]$, from which the opposite inclusion follows. $\square$

Let us introduce the notation $tightguard_K(\varphi, g)$ for $\langle\langle\, guard(\varphi \wedge g) \,\rangle\rangle_K^{\le}$. Let us establish some basic facts about $tightguard_K(\varphi, g)$.

Proposition 9. *Let g be a non-strict K-bounded clock guard, and let φ be a K-bounded clock constraint. Then*

(1) $[\![\varphi \wedge g]\!] = [\![\varphi \wedge guard(\varphi \wedge g)]\!]$,
(2) $tightguard_K(\varphi, g) = tightguard_K(\varphi, tightguard_K(\varphi, g))$.

Proof. Let φ and g be as in the statement of the proposition.

(1) follows from $[\![guard(\varphi \wedge g)]\!] \subseteq [\![g]\!]$ and $[\![\varphi \wedge g]\!] \subseteq [\![guard(\varphi \wedge g)]\!]$.
(2) By the definition of $tightguard_K(\varphi, g)$, and using form (b) and form (c) of Lemma 8, it suffices to prove

$$\left\langle\!\left\langle \bigwedge\{g' \in G_\Sigma \mid [\![\varphi \wedge g]\!] \subseteq [\![g']\!]\}\right\rangle\!\right\rangle_K^{\leq} = \left\langle\!\left\langle \bigwedge\{g' \in G_\Sigma \mid [\![\varphi \wedge \langle\langle guard(\varphi \wedge g)\rangle\rangle_K^{\leq}]\!] \subseteq [\![g']\!]\}\right\rangle\!\right\rangle_K^{\leq}.$$

This follows by noting that the expression on the left-hand side of $\subseteq$ is the same in both expressions, by property (1), since g is non-strict and K-bounded. □

The following lemma shows that sharply guarded EDERA can also be defined in terms of $sp(w_g)$, or any other clock constraint φ such that $\varphi \approx_K sp_K^{<}(w_g)$. The reason to define sharply guarded EDERA in terms of $sp_K^{<}(w_g)$ is that in order to bound the size of sharply guarded EDERA, we need to use $K^{<}$-approximations of postconditions in the construction of a sharply guarded EDERA in Lemma 12.

Lemma 10. *Let g be a non-strict K-bounded clock guard. Let φ and φ' be clock constraints such that $\varphi \approx_K \varphi'$. Then*

$$tightguard_K(\varphi, g) = tightguard_K(\varphi', g).$$

Proof. A K-bounded clock guard is a union of a set of regions. Therefore, if g is a K-bounded clock guard, $\varphi \approx_K \varphi'$ implies $\varphi \wedge g \approx_K \varphi' \wedge g$, which implies $Can(\varphi \wedge g) \approx_K Can(\varphi' \wedge g)$, which implies

$$\langle\langle guard(\varphi \wedge g)\rangle\rangle_K^{\leq} = \langle\langle guard(\varphi' \wedge g)\rangle\rangle_K^{\leq},$$

from which the lemma follows by the definition of $tightguard_K(\varphi, g)$. □

By Definition 7, an EDERA is sharply guarded if the outgoing transitions from a location have guards which cannot be strengthened without changing the timing conditions under which the next symbol will be accepted. The following lemma shows that these guards can be characterized in terms of biggest and smallest corners.

Lemma 11. *If $w_g(a, g) \in \mathcal{L}(dfa(D))$, where D is a K-bounded sharply guarded EDERA, then*

(a) *there is a timed word $w_t \in \mathcal{L}(D)$ such that $CW(w_t) = w_c(a, \gamma)$, $w_c(a, \gamma) \models w_g(a, g)$ and $\gamma \in bc_K(g)$.*
(b) *there is a timed word $w_t \in \mathcal{L}(D)$ such that $CW(w_t) = w_c(a, \gamma)$, $w_c(a, \gamma) \models w_g(a, g)$ and $\gamma \in sc_K(g)$.*

Proof. We first prove (a). Since D is a sharply guarded EDERA, we have that $sp_K^{<}(w_g) \wedge g$ is satisfiable, hence $sp(w_g) \wedge g$ is satisfiable (since $sp(w_g) \approx_K sp_K^{<}(w_g)$ by Lemma 3 and g is K-bounded). The basic property of postconditions, that $\gamma \models sp(w_g)$ if and only if there is a clocked word $w_c(a, \gamma)$ such that $w_c \models w_g$, implies that we must prove that there is a clock valuation γ such that $\gamma \models sp(w_g)$ and $\gamma \in bc_K(g)$. Let a be any action in Σ. If $x_a \leq n$ is a conjunct in g for $n \leq K$, then by the definition of sharply guarded, using form (c) in Lemma 8, and the fact that $sp(w_g)$ is non-strict, it follows that there is a clock valuation γ_a such that $\gamma_a \models sp(w_g)$ and $\gamma_a(x_a) = n$. Similarly, if there is no conjunct of form $x_a \leq n$ in g for $n \leq K$, then there is a clock valuation γ_a such that $\gamma_a \models sp(w_g)$ and $\gamma_a(x_a) > K$. Since this holds for any a, by Proposition 1 it follows that there is a clock valuation γ such that $\gamma \models sp(w_g)$ and $\gamma \in bc_K(g)$.

The proof of (b) is analogous: we can infer that whenever $x_a \geq n$ is a conjunct in g for $n < K$, then there is a clock valuation γ_a such that $\gamma_a \models sp(w_g)$ and $\gamma_a(x_a) = n$. A slight difference occurs for the case where $x_a \geq K$ is a conjunct in g: here we use satisfiability of $sp(w_g) \wedge g$ to infer that there is a clock valuation γ_a such that $\gamma_a \models sp(w_g)$ and $\gamma_a(x_a) \geq K$. Since this holds for any a, by Proposition 1 it follows that there is a clock valuation γ such that $\gamma \models sp(w_g)$ and $\gamma \in sc_K(g)$. □

Every EDERA can be transformed into an equivalent EDERA that is sharply guarded using the zone-graph construction [15].

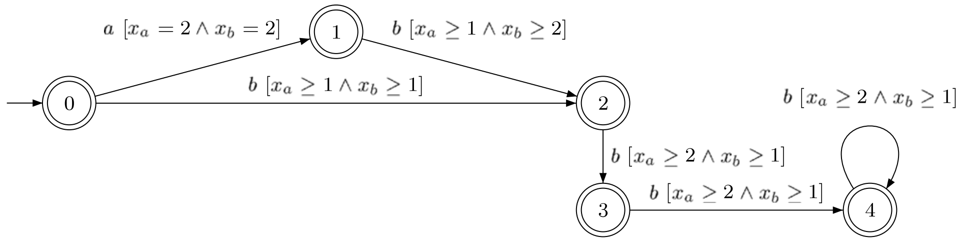
Lemma 12. *For every EDERA there is an equivalent EDERA that is sharply guarded.*

Proof. Let the EDERA $D = (\Sigma, L, l_0, \varrho, \eta)$ be K-bounded. We define an equivalent sharply guarded EDERA $D' = (\Sigma, L', l_0', \varrho', \eta')$ based on the so-called zone automaton for D. The set of locations of D' comprises pairs (l, φ) where $l \in L$ and φ is a K-bounded clock constraint. The intention is that φ is the $K^{<}$-approximated postcondition of any run from the initial location to (l, φ). The initial location l_0' of D' is $(l_0, sp(\lambda))$. For any location $l \in L$ and symbol a such that $\varrho(l, a)$ is defined and $\varphi \wedge \eta(l, a)$ is satisfiable, let $\varrho'((l, \varphi), a)$ be defined as $(\varrho(l, a), \varphi')$ where $\varphi' = \langle\langle sp(\varphi, (a, \eta(l, a)))\rangle\rangle_K^{<}$. We set $\eta'((l, \varphi), a) = tightguard_K(\varphi, \eta(l, a))$. By construction D' is event-deterministic.

We first show by induction over w_g' that whenever $(l, \varphi) = \delta'(l_0', w_g')$, where δ' is the transition function of $dfa(D')$ extended to words, i.e., D' has a run over w_g' to (l, φ), then $\varphi = sp_K^{<}(w_g')$. The base case $w_g' = \lambda$ is trivial. For the inductive step, let $w_g'(a, g) \in \mathcal{L}(dfa(D'))$, and let $\delta'(l_0', w_g') = (l, \varphi)$. By construction of D', $\delta'(l_0', w_g'(a, g)) = (\varrho(l, a), \varphi')$ where $\varphi' = \langle\langle sp(\varphi, (a, \eta(l, a)))\rangle\rangle_K^{<}$. We show that $\varphi' = \langle\langle sp(\varphi, (a, \eta'((l, \varphi), a)))\rangle\rangle_K^{<}$. Since $\eta'((l, \varphi), a) = \langle\langle guard(\varphi \wedge \eta(l, a))\rangle\rangle_K^{\leq}$, this follows from the equality

$$[\![\varphi \wedge g]\!] = [\![\varphi \wedge \langle\langle guard(Can(\varphi \wedge g))\rangle\rangle_K^{\leq}]\!],$$

which follows from (1) in Lemma 9 and the fact that g is a K-bounded clock guard.



$a\ [x_a = 2 \wedge x_b = 2]$
$b\ [x_a \geq 1 \wedge x_b \geq 2]$
$b\ [x_a \geq 1 \wedge x_b \geq 1]$
$b\ [x_a \geq 2 \wedge x_b \geq 1]$
$b\ [x_a \geq 2 \wedge x_b \geq 1]$
$b\ [x_a \geq 2 \wedge x_b \geq 1]$
0
1
2
3
4

Fig. 9. Sharply guarded automaton $\mathcal{H}'$.

The property proved in the previous paragraph, together with the observation in property (2) of Lemma 9, that $\eta'((l,\varphi),a) = tightguard_K(\varphi, \eta'((l,\varphi),a))$, implies by characterization (c) in Lemma 8 that D' is sharply guarded.

We then prove that D' is equivalent to D by induction. For every timed word w_t, we need to show that $w_t \in \mathcal{L}(D)$ iff $w_t \in \mathcal{L}(D')$. Due to the one-to-one correspondence between timed words and clocked words we present the proof in terms of clocked words. For λ, we start in l_0 in D and in $(l_0, sp(\lambda))$ in D'. So λ is accepted by both automata.

From the second paragraph of this proof, it follows that whenever $l = \delta(l_0, w_g)$ is defined, then there is a guarded word w'_g with the same underlying word as w_g such that $\delta'(l'_0, w'_g) = (l,\varphi)$ and such that $\varphi = sp^{<}_K(w_g) = sp^{<}_K(w'_g)$. Also the converse follows, i.e., that whenever $(l,\varphi) = \delta'(l'_0, w'_g)$ is defined, then there is a guarded word w_g with the same underlying word as w'_g, such that $\delta(l_0, w_g) = l$ and such that $\varphi = sp^{<}_K(w_g) = sp^{<}_K(w'_g)$. In both cases, it also follows for any alphabet symbol a that

$$[\![sp^{<}_K(w_g) \wedge \eta(l,a)]\!] = [\![sp^{<}_K(w_g) \wedge \eta'((l,\varphi),a)]\!].$$

To prove that $\mathcal{L}(D) \subseteq \mathcal{L}(D')$ we shall establish by induction over w_g that whenever $w_c \vDash w_g$ where $l = \delta(l_0, w_g)$ is defined, then there is a guarded word w'_g such that $w_c \vDash w'_g$ and $\delta'(l'_0, w'_g) = (l,\varphi)$ where $\varphi = sp^{<}_K(w_g) = sp^{<}_K(w'_g)$. The base case $w_g = \lambda$ is trivial. For the inductive step, let $w_c(a,\gamma) \vDash w_g(a,g)$ where $g = \eta(l,a)$. From the preceding paragraph, it follows that $w_c(a,\gamma) \vDash w'_g(a,g')$ where $g' = \eta'((l,\varphi),a)$. This concludes the inductive step.

The other inclusion $\mathcal{L}(D') \subseteq \mathcal{L}(D)$ follows in an analogous way. □

In Fig. 9 a sharply-guarded EDERA $\mathcal{H}'$ is shown, which is equivalent to the EDERA $\mathcal{H}$ shown in Fig. 11(b). The initial location $(0,\varphi_0)$, where $\varphi_0 = sp(\lambda)$, of $\mathcal{H}'$ corresponds to the location 0 in Fig. 9.

From the location 0 in $\mathcal{H}$ there is one a-transition and one b-transition. Therefore we construct an a-transition from the location $(0, sp(\lambda))$ to the location $(0,\varphi_1)$, where $\varphi_1 = \langle\langle sp(\varphi_0, (a, x_a = 2 \wedge x_b = 2))\rangle\rangle^{<}_K = (x_a + 2 = x_b)$, and a b-transition from location $(0,\varphi_0)$ to the location $(0,\varphi_2)$, where $\varphi_2 = \langle\langle sp(\varphi_0, (b, x_a \geq 1 \wedge x_b \geq 1))\rangle\rangle^{<}_K = x_b + 1 \leq x_a$. Since $\varphi_1 \wedge x_a = 2 \wedge x_b = 2$ (i.e., $x_a + 2 = x_b \wedge x_a = 2 \wedge x_b = 2$) is not satisfiable, from the location $(0,\varphi_1)$ we can construct only a b-transition, which leads to the location $(0, \langle\langle sp(\varphi_1, (b, x_a \geq 1 \wedge x_b \geq 1))\rangle\rangle^{<}_K)$, which happens to be $(0, x_b + 1 \leq x_a)$, i.e., $(0,\varphi_2)$ (location 2 in Fig. 9).

The guard for this transition is $tightguard_K(x_a + 2 = x_b, x_a \geq 1 \wedge x_b \geq 1) = x_a \geq 1 \wedge x_b \geq 2$. From location $(0,\varphi_2)$ we can construct only a b-transition, which leads to the location $(0,\varphi_3)$, where $\varphi_3 = \langle\langle sp(\varphi_2, (b, x_a \geq 1 \wedge x_b \geq 1))\rangle\rangle^{<}_K = x_b + 2 \leq x_a$, and has the guard $tightguard_K(\varphi_2, x_a \geq 1 \wedge x_b \geq 1) = x_a \geq 2 \wedge x_b \geq 1$. Again, from location $(0,\varphi_3)$ we can construct only a b-transition, which leads to the location $(0,\varphi_4)$, where $\varphi_4 = \langle\langle sp(\varphi_3, (b, x_a \geq 1 \wedge x_b \geq 1))\rangle\rangle^{<}_K = x_b + 2 < x_a$, and has the guard $tightguard_K(\varphi_3, x_a \geq 1 \wedge x_b \geq 1) = x_a \geq 2 \wedge x_b \geq 1$. The construction of $\mathcal{H}'$ terminates after adding a loop to the location $(0,\varphi_4)$.

The important property of sharply guarded EDERAs is that equivalence coincides with equivalence on the corresponding DFAs.

Definition 13. We call two sharply guarded EDERAs 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 14. *For two sharply guarded EDERAs D_1 and D_2, we have*

$$D_1 \equiv_t D_2 \quad \textit{iff} \quad D_1 \equiv_{dfa} D_2.$$

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 11 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 sharply guarded EDERAs.

Theorem 1. *For every EDERA there is a unique equivalent minimal sharply guarded EDERA (up to isomorphism).*

Proof. By Lemma 12, each EDERA D can be transformed into an equivalent EDERA D' that is sharply guarded. Let $\mathcal{A}_{min}$ be the unique minimal DFA which is equivalent to $dfa(D')$ (up to isomorphism). Since (as was remarked after Definition 7) whether or not a EDERA is sharply guarded depends only on $\mathcal{L}(dfa(D))$, we have that $D_{min} = era(\mathcal{A}_{min})$ is sharply guarded. By Lemma 14, D_{min} is the unique minimal sharply guarded EDERA (up to isomorphism) such that $D_{min} \equiv D'$, i.e., such that $D_{min} \equiv D$. □

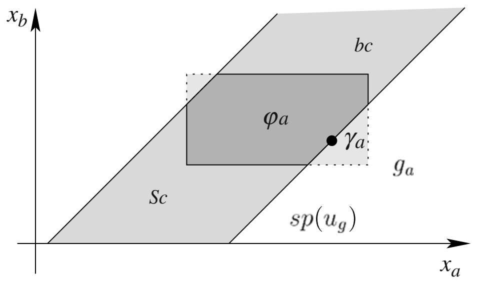
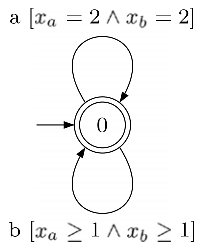


Fig. 10. An example for $sp(u_g)$, φ_a, and g_a.

T	λ
λ	$+$
$(a, x_a = 2, x_b = 2)$	$+$
$(b, x_a \geq 1 \wedge x_b \geq 1)$	$+$

(a) Table T


a $[x_a = 2 \wedge x_b = 2]$
0
b $[x_a \geq 1 \wedge x_b \geq 1]$

(b) Automaton $\mathcal{H}$

Fig. 11. Table T and non-sharply guarded automaton $\mathcal{H}$.

5.2. Learning guarded words

Angluin's algorithm is designed to query (untimed) words rather than timed words. Before we can present the final learning algorithm for EDERAs, we must describe how the *Assistant* answers a membership query for a guarded word, posed by the *Learner*, by asking several membership queries for timed words to the (timed) *Teacher*.

To answer a membership query for a guarded word w_g, the *Assistant* first extracts the untimed word w underlying w_g. It thereafter determines the unique guard refinement w'_g of w that is accepted by $\mathcal{A}$ (if one exists) by posing several membership queries to the (timed) *Teacher*, in a way to be described below. Note that, as observed just before Example 6, each untimed word w has at most one guard refinement accepted by $\mathcal{A}$. Finally, the *Assistant* answers the query by *yes* iff w'_g equals w_g.

The guard refinement of w accepted by $\mathcal{A}$ will be determined inductively, by learning the guard under which an action a is accepted, provided that a sequence u of actions has occurred so far. Letting u range over successively longer prefixes of w, the *Assistant* can then learn the guard refinement w'_g of w. Let $u = a_1a_2 \dots a_n$, and assume that for $i = 1, \dots, n$, the *Assistant* has previously learned the guard g_i under which a_i is accepted, given that the sequence $a_1 \dots a_{i-1}$ has occurred so far. He can then compute the strongest postcondition $sp(u_g)$, where $u_g = (a_1, g_1) \dots (a_n, g_n)$. The *Assistant* must now determine the strongest guard g_a such that a is accepted after u_g precisely when $\varphi_a \equiv sp(u_g) \wedge g_a$ holds. Note that by Definition 5, there is a unique strongest g_a with this property. In the following, we assume that $sp(u_g)$ and φ_a are both in canonical form.

The guard g_a is determined by inquiring whether a set of clock valuations γ_a satisfy φ_a. Without loss of generality, the *Assistant* works only with integer valuations. For each γ_a that satisfies the postcondition $sp(u_g)$, he can make a membership query for some clocked word $w_c(a, \gamma_a)$, where w_c satisfies the guarded word u_g, since such a guarded word $w_c(a, \gamma_a)$ exists precisely when $\gamma_a \models sp(u_g)$. In other words, he can ask the (timed) *Teacher* for every integer point in the polyhedron $[\![sp(u_g)]\!]$ whether it is in $[\![\varphi_a]\!]$. A typical situation for two clocks is depicted in Fig. 10.

Let us now describe how clock valuations γ_a are chosen in membership queries in order to learn the guard g_a for a. As mentioned before, we assume that the *Assistant* knows the maximal constant K that can appear in any guard. This means that if a clock valuation γ with $\gamma(x_b) > K$ satisfies g_a, then clock x_b has no upper bound in g_a. Thus, by Lemma 11, the guard g_a can be uniquely determined by two clock valuations, one in its biggest corner $bc_K(g_a)$, and one in its smallest corner $sc_K(g_a)$.

Let us consider how to find a clock valuation in $bc_K(g_a)$. Suppose first that the *Assistant* knows some clock valuation γ_a that satisfies φ_a. The *Assistant* will then repeatedly increase the clock values in γ_a until γ_a is in $bc_K(g_a)$. This is done as follows. At any point during this process, let *Max* be the set of clocks, initially empty, for which the *Assistant* knows that they have reached a maximum, which is at most K, let *AboveK* be the set of clocks which have become more than K, and let $Unknown = C_\Sigma \setminus (Max \cup AboveK)$ be the clocks for which a maximum value is still searched. At each iteration, the *Assistant* finds the maximal $k \in \{1, \dots, K+1\}$ such that the valuation γ_a can be changed by increasing all clocks in *Unknown* by k, keeping the clocks in *Max* unchanged, and finding suitable values for the clocks in *AboveK* such that γ_a still satisfies φ_a. This can be done by a binary search using at most $\log K$ queries. The *Assistant* then lets γ_a be this new valuation. For all clocks x_b with $\gamma_a(x_b) \geq K + 1$, the *Assistant* concludes that x_b has no upper bound in φ_a. These clocks are moved over from *Unknown* to *AboveK*. If $\gamma_a(x_b) \leq K$ for some clocks $x_b \in$ *Unknown* then among these a clock (or several clocks) must be found that cannot be increased, which will be moved over from *Unknown* to *Max*.

Let us examine how to find a clock x_b in *Unknown* that cannot be increased, i.e., such that φ_a implies the constraint $x_b \leq \gamma_a(x_b)$. The idea is to increase each clock in turn by 1 and see whether the result still satisfies φ_a. The particularity to handle is that it may be possible to increase a clock x_b only together with other clocks, since $sp(u_g)$ must be satisfied (e.g., in Fig. 10 we see that if x_a is incremented in γ_a then x_b must also be incremented to stay in $sp(u_g)$). To define this in more detail, let us regard $sp(u_g)$ and g_a as fixed, and define for each integer valuation γ_a such that $\gamma_a \models \varphi_a$ the relation $\overset{\gamma_a}{\trianglelefteq}$ on the set C_Σ of clocks by

$$x_b \overset{\gamma_a}{\trianglelefteq} x_c \quad \text{if } sp(u_g) \text{ implies } x_b - x_c \leq \gamma_a(x_b) - \gamma_a(x_c)$$

(recall that we assume $sp(u_g)$ to be canonical). Thus, $x_b \overset{\gamma_a}{\trianglelefteq} x_c$ means that in order to keep $\gamma_a \vDash sp(u_g)$ while incrementing $\gamma_a(x_b)$ by 1 we should also increment $\gamma_a(x_c)$ by 1. For a valuation γ_a, define $dep_{\gamma_a}(x_b)$ as $\{x_c \; : \; x_b \overset{\gamma_a}{\trianglelefteq} x_c\}$. This means that in order to keep $\gamma_a \vDash sp(u_g)$ while incrementing $\gamma_a(x_b)$ by 1 we should also increment all clocks in $dep_{\gamma_a}(x_b)$ by 1.

Assume that $\gamma_a \models \varphi_a$. For a set C of clocks, define $\gamma_a[C \oplus k]$ as $\gamma_a(x_b) + k$ for $x_b \in C$ and $\gamma_a(x_b)$ otherwise. From the definition of dep_{γ_a}, we infer that $\gamma_a[dep_{\gamma_a}(x_b) \oplus 1] \models sp(u_g)$ for all x_b in *Unknown*. We claim that for each clock x_b in *Unknown* for which $\gamma_a[dep_{\gamma_a}(x_b) \oplus 1] \not\models \varphi_a$, the clock constraint φ_a must contain the conjunct $x_b \leq \gamma_a(x_b)$. To see this, we first note that $\gamma_a(x_b) \leq K$ and $\gamma_a[dep_{\gamma_a}(x_b) \oplus 1] \not\models \varphi_a$ together imply that there must be some x_c in $dep_{\gamma_a}(x_b)$ such that φ_a contains the conjunct $x_c \leq \gamma_a(x_c)$. We also note that $x_c \in dep_{\gamma_a}(x_b)$ means that $sp(u_g)$ contains the conjunct $x_b - x_c \leq \gamma_a(x_b) - \gamma_a(x_c)$. Hence, since φ_a is canonical, it contains the conjunct $x_b \leq \gamma_a(x_b)$.

To update *Max*, we thus move to *Max* all clocks such that $\gamma_a[dep_{\gamma_a}(x_b) \oplus 1] \not\models \varphi_a$. As an optimization, we can sometimes avoid to make one query for each clock in *Unknown* increased by 1 by first analysing the structure of the graph whose nodes are the clocks in *Unknown*, and whose edges are defined by the relation $\overset{\gamma_a}{\trianglelefteq}$. It is then sufficient to make at most one query for some clock in each strongly connected component of this graph, and use it as a query for each clock in the component.

After an iteration, another iteration is performed by finding a k to increase the clocks that remain in *Unknown*, and thereafter finding out which of these have reached their upper bounds. When *Unknown* $= \emptyset$, a valuation in $bc_K(g_a)$ has been found and the algorithm terminates.

Thus, all in all, determining the upper bound of a guard g_a needs at most $|C_\Sigma|$ binary searches, since in every loop at least one clock is moved to *Max*. Each uses at most $\log K + |C_\Sigma|$ membership queries. In an analogous way, we can find a minimal clock valuation that satisfies φ_a. The guard g_a is given by the K-approximation of the guard that has the minimal clock valuation as smallest corner and the maximal clock valuation as biggest corner, which can easily be formulated given these two points. Thus, the *Assistant* needs at most $2|C_\Sigma|(\log K + |C_\Sigma|)$ membership queries to learn a guard g_a, if initially it knows a valuation which satisfies φ_a.

Suppose now that the *Assistant* does not know a clock valuation γ_a that satisfies φ_a. In principle, φ_a and therefore g_a could specify exactly one valuation, meaning that the *Assistant* essentially might have to ask membership queries for all $\binom{|\Sigma|+K}{|\Sigma|}$ integer points that could be specified by φ_a. This is the number of non-increasing sequences of $|\Sigma| = |C_\Sigma|$ elements, where each element has values among 0 to K, since $sp(u_g)$ defines at least an ordering on the clocks.

Thus, the *Assistant* can answer a query for a guarded word w_g using at most $|w|\binom{|\Sigma|+K}{|\Sigma|}$ (timed) membership queries.

5.3. Algorithm TL^*_{sg}

Let us now turn to the problem of learning a timed language $\mathcal{L}(D)$ accepted by an EDERA D. We can assume without loss of generality that D is the unique minimal and sharply guarded EDERA that exists due to Theorem 1. Then D is uniquely determined by its symbolic language of $\mathcal{A} = dfa(D)$, which is a regular (word) language. In this setting, we assume

- to know an upper bound K on the constants occurring in guards of D,
- to have a *Teacher* who is able to answer two kinds of queries:
  - A *membership query* consists in asking whether a timed word w_t over Σ is in $\mathcal{L}(D)$.
  - An *equivalence query* consists in asking whether a hypothesized EDERA H is correct, i.e., whether $\mathcal{L}(H) = \mathcal{L}(D)$. The *Teacher* will answer *yes* if H is correct, or else supply a counterexample u, either in $\mathcal{L}(D) \setminus \mathcal{L}(H)$ or in $\mathcal{L}(H) \setminus \mathcal{L}(D)$.

Based on the observations in Section 5.1, our solution is to learn $\mathcal{L}(dfa(D))$, which is a regular language and can therefore be learned in principle using Angluin's learning algorithm. However, Angluin's algorithm is designed to query (untimed) words rather than timed words. Let us therefore extend the *Learner* in Angluin's algorithm by an *Assistant*, whose role is to answer a membership query for a guarded word, posed by the *Learner*, by asking several membership queries for timed words to the (timed) *Teacher*. This is described in Section 5.2. To complete the learning algorithm, we have to explain how the *Assistant* can answer equivalence queries to the *Learner*. Given a DFA $\mathcal{H}$, the *Assistant* can ask the (timed) *Teacher*, whether $era(\mathcal{H}) = D$. If so, the *Assistant* replies *yes* to the *Learner*. If not, the *Teacher* presents a timed word w_t that is in $\mathcal{L}(D)$ but not in $\mathcal{L}(era(\mathcal{H}))$ (or the other way round). For the word w underlying w_t, we can obtain its guard refinement w_g as described in the previous paragraph. Then w_g is in $\mathcal{L}(dfa(D))$ but not in $\mathcal{L}(\mathcal{H})$ (or the other way around). Thus, the *Assistant* can answer the equivalence query by w_g in this case.

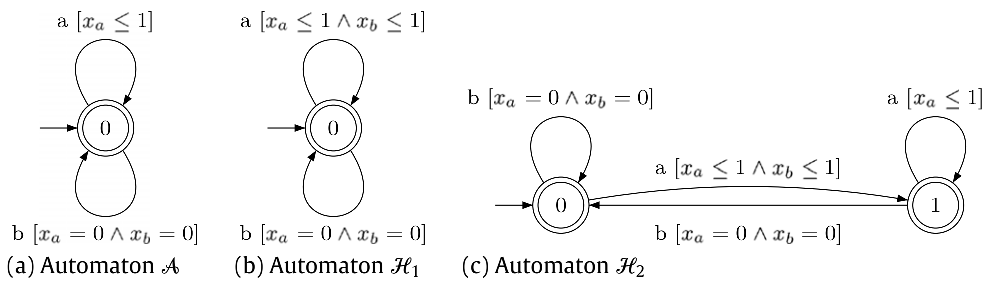

a $[x_a \leq 1]$
b $[x_a = 0 \wedge x_b = 0]$
(a) Automaton $\mathcal{A}$
a $[x_a \leq 1 \wedge x_b \leq 1]$
b $[x_a = 0 \wedge x_b = 0]$
(b) Automaton $\mathcal{H}_1$
b $[x_a = 0 \wedge x_b = 0]$
a $[x_a \leq 1 \wedge x_b \leq 1]$
b $[x_a = 0 \wedge x_b = 0]$
a $[x_a \leq 1]$
(c) Automaton $\mathcal{H}_2$

Fig. 12. EDERAs.

At this point, we should remark that it can be the case that the hypothesized automaton $\mathcal{H}$ which the algorithm constructs is not sharply guarded. This can happen if the observation table does not contain for each prefix u_g of the table and each symbol $a \in \Sigma$ at least one column labeled by a suffix of form (a, g) such that $u_g(a, g)$ is accepted. As an illustration, consider the hypothesized automaton $\mathcal{H}$ in Fig. 11(b), constructed from the table T shown in Fig. 11(a). Let us assume that $K = 4$ is given as an upper bound on constants in guards. The automaton $\mathcal{H}$ is non-sharply guarded, since after the guarded word $(a, x_a = 2 \wedge x_b = 2)(b, x_a \geq 1 \wedge x_b \geq 1) \in \mathcal{L}(dfa(\mathcal{H}))$ the postcondition implies $x_b \geq 3$, which means that after this guarded word, the guard on the following b-transition is not sharp. A so constructed non-sharply guarded automaton has always less locations than a corresponding sharply guarded automaton constructed from the same information.

Algorithm 3 Pseudo code for Assistant of TL^*_{sg}

```
class $TL^*_{sg}$Assistant implements Teacher{
  Teacher timedteacher

Constructor $TL^*_{sg}$Assistant(teacher)
  timedteacher = teacher

Function equivalence_query($\mathcal{H}$)
  timedteacher.equivalence_query($\mathcal{H}$)
  if the answer to equivalence query is a counterexample $w_t$
    Extract the word $w$ underlying $w_t$
    Learn guard refinement $w_g$ of $w$
    return $w_g$
  else
    return 'yes'

Function membership_query($w_g$)
  Extract underlying $w$ of $w_g$
  Learn guard refinement $w'_g$ of $w$
  if $w'_g = w_g$ then
    return 'yes'
  else
    return 'no'
}
```

Algorithm 4 Pseudo code for TL^*_{sg}

```
class $TL^*_{sg}$ extends L*{

Constructor $TL^*_{sg}$(timedteacher,$\Sigma$,K)
    $\Gamma = TL^*_{sg}$Alphabet($\Sigma$, K)
    teacher = $TL^*_{sg}$Assistant(timedteacher)
}
```

We call the algorithm outlined in the section TL^*_{sg}. More specifically, the algorithm for learning sharply guarded EDERA is as Algorithm 2, but extended with the Assistant shown in Algorithm 3.

Example 15. Let us illustrate the algorithm by showing how to learn the language of the automaton $\mathcal{A}$ depicted in Fig. 12(a). Initially, the algorithm asks membership queries for λ. It additionally asks membership queries to learn that (a, g) is accepted iff $g = x_a \leq 1 \wedge x_b \leq 1$ and (b, g) is accepted iff $g = x_a = 0 \wedge x_b = 0$. To follow the algorithm, we should also add rejected guarded words to the table: these are needed in order to find inconsistencies. In this example, we need to have only $(a, x_a \leq 1 \wedge x_b \geq 0)$ in the table, and in order to keep the table as small as possible we do not add other rejected guarded words.

T_1	λ
λ	+
$(a, x_a \leq 1, x_b \geq 0)$	-
$(a, x_a \leq 1 \wedge x_b \leq 1)$	+
$(b, x_a = 0 \wedge x_b = 0)$	+

(a) Table T_1

T_2	λ
λ	+
$(a, x_a \leq 1 \wedge x_b \leq 1)$	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)$	+
$(a, x_a \leq 1 \wedge x_b \geq 0)$	-
$(b, x_a = 0 \wedge x_b = 0)$	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(b, x_a = 0 \wedge x_b = 0)$	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)(a, x_a \leq 1 \wedge x_b \geq 0)$	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)(b, x_a = 0 \wedge x_b = 0)$	+

(b) Table T_2

Fig. 13. Tables T_1 and T_2.

T_3	λ	u_g
λ	+	-
$(a, x_a \leq 1 \wedge x_b \leq 1)$	+	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)$	+	+
$(a, x_a \leq 1 \wedge x_b \geq 0)$	-	-
$(b, x_a = 0 \wedge x_b = 0)$	+	-
$(a, x_a \leq 1 \wedge x_b \leq 1)(b, x_a = 0 \wedge x_b = 0)$	+	-
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)(a, x_a \leq 1 \wedge x_b \geq 0)$	+	+
$(a, x_a \leq 1 \wedge x_b \leq 1)(a, x_a \leq 1 \wedge x_b \geq 0)(b, x_a = 0 \wedge x_b = 0)$	+	-

Fig. 14. Table T_3, $u_g = (a, x_a \leq 1 \wedge x_b \geq 0)$.

This yields the initial observation table T_1 shown in Fig. 13(a). It is consistent and closed. Then the *Learner* constructs a hypothesized DERA $\mathcal{H}_1$ shown in Fig. 12(b) and submits $\mathcal{H}_1$ in an equivalence query. Assume that the counterexample $(a, 1.0)(a, 1.5)$ is returned. It is accepted by $\mathcal{A}$ but rejected by $\mathcal{H}_1$. The algorithm processes the counterexample and produces the observation table T_2 given in Fig. 13(b), which is not consistent. Following Angluin's algorithm we construct a closed and consistent table T_3 shown in Fig. 14. The sharply guarded EDERA $\mathcal{H}_2$ visualized in Fig. 12(c) corresponds to the observation table T_3 and accepts the same language as $\mathcal{A}$.

5.4. Complexity

In the L^* algorithm the number of membership queries is 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.

In our setting, a single membership query for a guarded word w_g might give rise to $|w|\binom{|\Sigma|+K}{|\Sigma|}$ membership queries to the (timed) *Teacher*. The alphabet of the DFA $dfa(D)$ is $\Sigma \times G$. Thus, the query complexity of TL^*_{sg} for a sharply guarded EDERA with n locations is

$$O\left(kn^2ml\binom{|\Sigma|+K}{|\Sigma|}\right)$$

where l is the length of the longest guarded word queried and k is the size of alphabet $\Sigma \times G$. The longest queried word is bounded by $O(m+n)$. If the *Teacher* always presents counterexamples of minimal length, then m is bounded by $O(n)$. The number of equivalence queries remains at most n. Note that, in general a (non-sharply guarded) EDERA D gives rise to a sharply guarded EDERA with double exponentially more locations. Since the number of regions is bounded by $|\Sigma|!2^{|\Sigma|}(2K+2)^{|\Sigma|}$, the number of locations in a sharply guarded EDERA is bounded by $n_1 2^{|\Sigma|!2^{|\Sigma|}(2K+2)^{|\Sigma|}}$, where n_1 is the number of locations in the EDERA. Thus, the query complexity of TL^*_{sg} for EDERA with n_1 locations is

$$O\left(kn_1^2 2^{2|\Sigma|!2^{|\Sigma|}(2K+2)^{|\Sigma|}}ml\binom{|\Sigma|+K}{|\Sigma|}\right).$$

6. Learning of DERA

Let us now turn our attention to learn the full class of deterministic event recording automata. The scheme for developing a learning algorithm is analogous to the scheme used for EDERAs in Section 5: we define a class of DERAs that admit a natural definition of right congruences, so that a DERA D in this class uniquely determines a language $\mathcal{L}(dfa(D))$. We show that each DERA can be transformed to this form. Then our solution is to learn $\mathcal{L}(dfa(D))$ using an assistant, whose role is to answer membership queries for guarded words by asking membership queries for timed words. In order to cope with the class of all DERAs, we need to find a different unique representation, and to change the task of the assistant.

6.1. Simple DERAs

Definition 16. A K-bounded DERA D is *simple* if all its guards are simple and if whenever $w_g(a, g)$ is a prefix of some word in $\mathcal{L}(dfa(D))$, then $sp^<_K(w_g) \wedge g$ is satisfiable. $\square$

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

Lemma 17. *If $w_g \in \mathcal{L}(dfa(D))$, where D is a simple DERA, then there is a timed word $w_t \in \mathcal{L}(D)$ such that $w_t \models w_g$.*

Proof. The lemma follows by induction from the fact that $sp^{<}_{K}(w'_g) \wedge g'$ is satisfiable iff $sp(w'_g) \wedge g'$ is satisfiable, whenever w'_g is a guarded word and g' is a K-bounded simple guard. □

We prove an important property of simple guarded words.

Lemma 18. *Let $w_g = (a_1, g_1) \dots (a_n, g_n)$ be a K-bounded simple guarded word, and let $w_t = (a_1, \gamma_1) \dots (a_n, \gamma_n)$ and $w'_t = (a_1, \gamma'_1) \dots (a_n, \gamma'_n)$ be two clocked words. If $w_t \vDash w_g$ and $w'_t \vDash w_g$ then $\gamma_i \sim_K \gamma'_i$ for $1 \le i \le n$.*

Proof. Since $\gamma_i \vDash g_i$ and $\gamma'_i \vDash g_i$, then for all $1 \le i \le n$ and $x_a \in C_\Sigma$ such that $\gamma_i(x_a) \le K$ and $\gamma'_i(x_a) \le K$ we have $\lfloor \gamma_i(x_a) \rfloor = \lfloor \gamma'_i(x_a) \rfloor$, and $fract(\gamma_i(x_a)) = 0$ iff $fract(\gamma'_i(x_a)) = 0$. We prove by induction over i that for $1 \le i \le n$ we have $fract(\gamma_i(x_{a_j})) - fract(\gamma_i(x_{a_k})) \ge 0$ iff $fract(\gamma'_i(x_{a_j})) - fract(\gamma'_i(x_{a_k})) \ge 0$ for all $a_j, a_k \in \Sigma$, whenever $\gamma_i(x_{a_j}) \le K$ and $\gamma_i(x_{a_k}) \le K$. For $i = 1$, this follows from $\gamma_1(x_{a_j}) = \gamma_1(x_{a_k})$ and $\gamma'_1(x_{a_j}) = \gamma'_1(x_{a_k})$. Assume that $\gamma_i \sim_K \gamma'_i$, that $\gamma_{i+1}(x_{a_j}) \le K$, that $\gamma_{i+1}(x_{a_k}) \le K$, and that $fract(\gamma_{i+1}(x_{a_j})) - fract(\gamma_{i+1}(x_{a_k})) \ge 0$. From $\gamma_i \sim_K \gamma'_i$, together with $\lfloor \gamma_{i+1}(x_{a_j}) \rfloor = \lfloor \gamma'_{i+1}(x_{a_j}) \rfloor$ and $\lfloor \gamma_{i+1}(x_{a_k}) \rfloor = \lfloor \gamma'_{i+1}(x_{a_k}) \rfloor$ we deduce $fract(\gamma'_{i+1}(x_{a_j})) - fract\ (\gamma'_{i+1}(x_{a_k})) \ge 0$. □

Every DERA can be transformed into an equivalent DERA that is simple using the region–graph construction [3].

Lemma 19. *For every K-bounded DERA there is an equivalent K-bounded DERA that is simple.*

Proof. Let the DERA $D = (\Sigma, L, l_0, L^f, \delta)$ be K-bounded. We define an equivalent simple DERA $D' = (\Sigma, L', l'_0, L^{f'}, \delta')$ by adapting the region graph construction for D.

The set of locations L' of D' comprises pairs (l, φ) where $l \in L$ and φ is a K-bounded clock constraint. The intention is that φ is the $K^{<}$-approximated postcondition of any run from the initial location to (l, φ). The initial location l'_0 of D' is $(l_0, sp(\lambda))$. We also introduce the location $l_e = (l_0, true)$ in L', where $l_e \notin L^{f'}$. For every symbol a and K-bounded simple guard g' for which there is a guard g such that $\delta(l, a, g)$ is defined and g' implies g, let $\delta'((l, \varphi), a, g')$ be defined as (l', φ') where $l' = \delta(l, a, g)$ and $\varphi' = \langle\langle (\varphi \wedge g')[x_a \mapsto 0]\uparrow \rangle\rangle^{<}_{K}$ if $\varphi' \ne false$, otherwise, $\delta'((l, \varphi), a, g') = l_e$. The final states are given by $(l, \varphi) \in L^{f'}$ iff $l \in L^f$.

To prove that D' is simple we need to show that if $w_g(a, g)$ is a prefix of some word in $\mathcal{L}(dfa(D'))$, then $sp^{<}_{K}(w_g) \wedge g$ is satisfiable. Let (l, φ) be the location reached from l'_0 on input of the guarded word w_g. By construction of D', it is the case that $\langle\langle (\varphi \wedge g)[x_a \mapsto 0]\uparrow \rangle\rangle^{<}_{K}$ is satisfiable, since $\delta'((l, \varphi), a, g)$ is not l_e. Hence also $sp^{<}_{K}(w_g) \wedge g$ is satisfiable.

We show that D' is equivalent to D.

Let $w_t \vDash w_g$ and $w_g \in \mathcal{L}(dfa(D))$. We show that there is $w'_g \in \mathcal{L}(dfa(D'))$ such that $w_t \vDash w'_g$. Let u_g be any prefix w_g and let $l = \delta(l_0, u_g)$. We prove by induction on the length of u_g that if u_t is a prefix of w_t with $u_t \vDash u_g$, then there is a simple guarded word u'_g such that $u_t \vDash u'_g$ and such that $\delta'((l_0, \varphi_0), u'_g) = (l, \varphi)$ for some φ. For the base case, if $u_g = \lambda$ then $u'_g = \lambda$ and $\delta'((l_0, \varphi_0), u'_g) = (l_0, \varphi_0)$. For the inductive step, assume this property for u_g, and let $u_g(a, g)$ be a prefix of w_g. Let $l' = \delta(l_0, u_g(a, g))$. Let $u_t(a, t)$ be the prefix of w_t with $u_t(a, t) \vDash u_g(a, g)$, and let $u'_g(a, g')$ be the unique simple guarded word with $u_t(a, t) \vDash u'_g(a, g')$. Then g' implies g, and by the construction of D' we infer that $\varphi \wedge g'$ is satisfiable, and that $\delta'((l, \varphi), a, g') = (l', \varphi')$ where $\varphi' = \langle\langle (\varphi \wedge g')[x_a \mapsto 0]\uparrow \rangle\rangle^{<}_{K}$, with $\varphi' \ne false$. This concludes the induction. From $w_g \in \mathcal{L}(dfa(D))$ we infer $\delta(l_0, w_g) \in L^f$. Let $l^f = \delta(l_0, w_g)$. By the just proved property, there is a simple guarded word w'_g such that $w_t \vDash w'_g$ and such that $\delta'((l_0, \varphi_0), w'_g) = (l^f, \varphi)$ for some φ. By the construction of D' we have $(l^f, \varphi) \in L^{f'}$, hence $w'_g \in \mathcal{L}(dfa(D'))$.

Let $w_t \vDash w'_g$ and $w'_g \in \mathcal{L}(dfa(D'))$. We show that there is $w_g \in \mathcal{L}(dfa(D))$ such that $w_t \vDash w_g$. Let u'_g be any prefix of w'_g, let $(l, \varphi) = \delta'((l_0, \varphi_0), u'_g)$, and let u_t be the prefix of w_t with $u_t \vDash u'_g$. We prove by induction on the length of u'_g that there is a guarded word u_g such that $\delta(l_0, u_g) = l$ and $u_t \vDash u_g$. For the base case, if $u'_g = \lambda$ then $u_g = \lambda$ and $\delta(l_0, u_g) = l_0$. For the inductive step, assume this property for u'_g, and let $u'_g(a, g')$ be a prefix of w'_g. Let $(l', \varphi') = \delta(l_0, u'_g(a, g'))$. Let $u_t(a, t)$ be the prefix of w_t with $u_t(a, t) \vDash u'_g(a, g')$. By the construction of D' there is a g such that g' implies g and $\delta(l, a, g) = l'$. Since g' implies g we infer $u_t(a, t) \vDash u_g(a, g)$. This concludes the induction. From $w'_g \in \mathcal{L}(dfa(D'))$ we infer $\delta'((l_0, \varphi_0), w'_g) = (l^f, \varphi)$ for some $l^f \in L^f$. By the just proved property, there is a guarded word w_g such that $w_t \vDash w_g$ and such that $\delta(l_0, w_g) = l^f$. Then $w_g \in \mathcal{L}(dfa(D))$. □

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

Definition 20. We call two simple DERAs D_1 and D_2 *dfa-equivalent*, denoted $D_1 \equiv_{dfa} D_2$, iff $dfa(D_1)$ and $dfa(D_2)$ accept the same language (in the sense of DFAs).

Now, exactly as in Section 5, we get counterparts for Lemma 14 and Theorem 1.

Lemma 21. *For two simple DERAs D_1 and D_2, we have*

$$D_1 \equiv_t D_2 \quad \textit{iff} \quad D_1 \equiv_{dfa} D_2.$$

We can now prove the central property of simple DERAs.

Theorem 2. *For every DERA there is a unique equivalent minimal simple DERA (up to isomorphism).*

Proof. The proof is analogous to the proof of Theorem 1. □

6.2. Algorithm TL^*_s

Given a timed language that is accepted by a DERA D, we can assume without loss of generality that D is the unique minimal and simple one that exists due to Theorem 2. Then D is uniquely determined by its symbolic language of $\mathcal{A} = dfa(D)$, which is a regular (word) language over $\Sigma \times G_s$, where G_s is a set of simple clock guards. Thus, we can learn $\mathcal{A}$ using Angluin's algorithm and return $era(\mathcal{A})$. 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. Moreover, it can be the case that the *Teacher* answers *yes* to an equivalence query for a hypothesized automaton $\mathcal{H}$ and $\mathcal{H}$ is smaller than $\mathcal{A}$.

Similar as in the previous section, we extend the *Learner* in Angluin's algorithm by a *Assistant*, whose role is to answer a membership query for a simple guarded word, posed by the *Learner*, by asking membership queries for timed words to the (timed) *Teacher*. Furthermore, he also has to answer equivalence queries, consulting the timed *Teacher*.

For a simple guarded word $w_g = (a_1, g_1) \dots (a_n, g_n)$ each simple guard g that extends w_g together with an action a defines exactly one region. Thus, if w_g is accepted, it is enough to check a in a single point in this region defined by g and the postcondition of w_g. In other words, it suffices to check an arbitrary timed word $w_t \models w_g$ to check whether w_g 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 for the simple DERA with n locations is $O(|\Sigma|^2 n^2 mK)$, where m is the length of the longest counterexample.

Algorithm 5 Pseudo code for Assistant of TL^*_s

```
class $TL^*_s$Assistant implements Teacher{
  Teacher timedteacher

Constructor $TL^*_s$Assistant(teacher)
      timedteacher = teacher

Function equivalence_query($\mathcal{H}$)
  timedteacher.equivalence_query($\mathcal{H}$)
  if the answer to equivalence query is a counterexample $w_t$
    Construct simple guarded word $w_g$ such that $w_t \models w_g$.
    return $w_g$
  else
    return 'yes'

Function membership_query($w_g$)
   if there is no $w_t$ such that $w_t \models w_g$ then
     return 'no'
   else
     Choose $w_t$ such that $w_t \models w_g$
     timedteacher.membership_query($w_t$)
     if answer to membership query is 'no'
       return 'no'
     else
       return 'yes'
}
```

Algorithm 6 Pseudo code for TL^*_s

```
class $TL^*_s$ extends L*{

Constructor $TL^*_s$(timedteacher,$\Sigma$,K)
    $\Gamma = TL^*_s$Alphabet($\Sigma$, K)
    teacher = $TL^*_s$Assistant(timedteacher)
}
```

Example 22. Let us illustrate the algorithm by showing how to learn the language of the automaton $\mathcal{A}$ depicted in Fig. 15. Initially, the algorithm asks membership queries for λ, $(a, x_a = 0)$, $(a, 0 < x_a < 1)$, $(a, x_a = 1)$ and $(a, x_a > 1)$. This yields the initial observation table T_1 shown in Fig. 16(a). It is consistent but not closed, since $row((a, x_a = 0))$ is distinct from $row(\lambda)$. Following Angluin's algorithm, we can construct a closed and consistent table T_2 shown in Fig. 16(b). Then the *Learner* constructs a hypothesized DERA $\mathcal{H}_1$ shown in Fig. 17 and submits $\mathcal{H}_1$ in an equivalence query. Assume that the counterexample $(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)$ is returned. It is rejected by $\mathcal{A}$ but accepted by $\mathcal{H}_1$. The algorithm

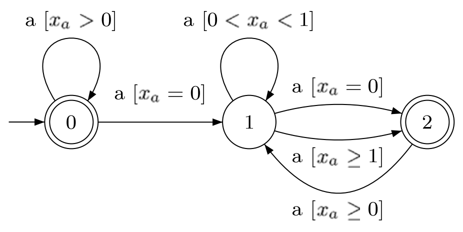


Fig. 15. Automaton $\mathcal{A}$.

T_1	λ
λ	+
$(a, x_a = 0)$	-
$(a, 0 < x_a < 1)$	+
$(a, x_a = 1)$	+
$(a, x_a > 1)$	+

(a)

T_2	λ
λ	+
$(a, x_a = 0)$	-
$(a, 0 < x_a < 1)$	+
$(a, x_a = 1)$	+
$(a, x_a > 1)$	+
$(a, x_a = 0)(a, x_a = 0)$	+
$(a, x_a = 0)(a, 0 < x_a < 1)$	-
$(a, x_a = 0)(a, x_a = 1)$	+
$(a, x_a = 0)(a, x_a > 1)$	+

(b)

Fig. 16. Tables T_1 and T_2.

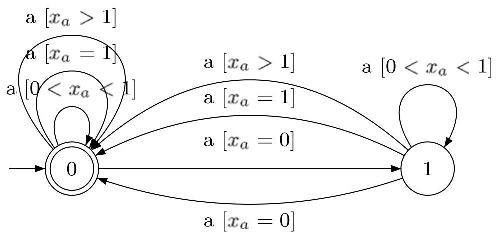


Fig. 17. Automaton $\mathcal{H}_1$.

T_3	λ	$(a, x_a > 1)$
λ	+	+
$(a, x_a = 0)$	-	+
$(a, x_a = 0)(a, x_a = 0)$	+	-
$(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)$	-	+
$(a, 0 < x_a < 1)$	+	+
$(a, x_a = 1)$	+	+
$(a, x_a > 1)$	+	+
$(a, x_a = 0)(a, 0 < x_a < 1)$	-	+
$(a, x_a = 0)(a, x_a = 1)$	+	-
$(a, x_a = 0)(a, x_a > 1)$	+	-
$(a, x_a = 0)(a, x_a = 0)(a, x_a = 0)$	-	+
$(a, x_a = 0)(a, x_a = 0)(a, x_a = 1)$	-	+
$(a, x_a = 0)(a, x_a = 0)(a, x_a > 1)$	-	+
$(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)(a, x_a = 0)$	+	-
$(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)(a, 0 < x_a < 1)$	-	+
$(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)(a, x_a = 1)$	+	-
$(a, x_a = 0)(a, x_a = 0)(a, 0 < x_a < 1)(a, x_a > 1)$	+	-

Fig. 18. Table T_3.

processes the counterexample and finally produces the observation table T_3 given in Fig. 18. The automaton $\mathcal{H}_2$ visualized in Fig. 19 corresponds to the observation table T_3 and accepts the same language as $\mathcal{A}$.

7. Learning non-sharply guarded EDERAs

Learning a sharply guarded EDERA allows to transfer Angluin's setting to the timed world. However, in practice, one might be interested in a smaller non-sharply guarded EDERA rather than its sharply guarded version. In this section, we describe how to learn a usually smaller, non-sharply guarded version. The idea is to identify states whose futures are "similar". While in the worst-case, more membership queries are needed, we hope that the algorithm converges faster in practice.

Let us illustrate the main idea underlying this section: Fig. 22 shows an EDERA $\mathcal{A}_2$ constructed by the algorithm TL^*_{sg}, while Fig. 21(a) shows an EDERA $\mathcal{A}_1$, which accepts the same language as $\mathcal{A}_2$, but has less locations than $\mathcal{A}_2$. In terms of TL^*_{sg}, locations 2 and 3 in $\mathcal{A}_2$ are different, since an a-transition from location 2 can be taken when $x_a = 1$ and $x_b = 3$, while

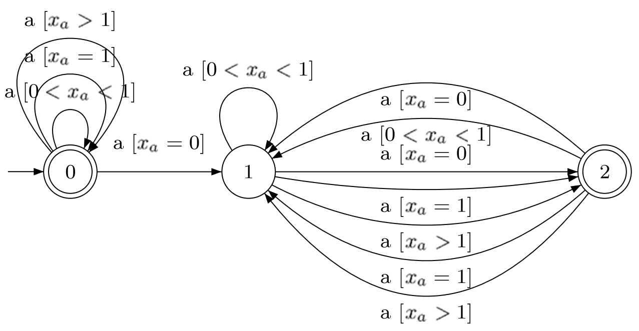


Fig. 19. Automaton $\mathcal{H}_2$.

from location 3 an a-transition cannot be taken when $x_a = 1$ and $x_b = 3$—reaching location 3 yields the postcondition $x_b - x_a = 3$. However, we can merge location 2, 3 and 4 if we add an a-transition from location 2 to location 2 with the guard $x_a = 1 \wedge x_b \leq 4$. In this section we show how to find guards which allow merging of locations, which are non-mergeable in terms of TL^*_{sg}. We develop our ideas in the setting of learning non-sharply guarded EDERAs, but a similar study could be carried out to learn non-simple DERAs.

7.1. *Learning based unification*

Let us now define a relationship on guarded words, which will be used to merge states whose futures are "similar", taking the postcondition into account.

Let $PG = \{\langle \varphi_1, (a_1, g_{11}) \dots (a_n, g_{1n})\rangle, \dots, \langle \varphi_k, (a_1, g_{k1}) \dots (a_n, g_{kn})\rangle\}$ be a set of k pairs of postconditions and guarded words with the same sequences of actions. We say that the guarded word $(a_1, \hat{g_1}) \dots (a_n, \hat{g_n})$ *unifies PG* if for all $j \in \{1, \dots, k\}$ and $i \in \{1, \dots, n\}$

$$g_{ji} \wedge sp(\varphi_j, (a_1, g_{j1}) \dots (a_{i-1}, g_{j(i-1)})) \equiv \hat{g}_i \wedge sp(\varphi_j, (a_1, \hat{g}_1) \dots (a_{i-1}, \hat{g}_{i-1})).$$

Then, the set PG is called *unifiable* and $(a_1, \hat{g_1}) \dots (a_n, \hat{g_n})$ is called a *unifier*. Intuitively, the guarded words with associated postconditions can be unified if there is a unifying, more liberal guarded word, which is equivalent to all guarded words in the context of the respective postconditions. Then, given a set of guarded words with postconditions among $\{\varphi_1, \dots, \varphi_k\}$, these guarded words can be considered to yield the same state, provided that the set of future guarded actions together with the respective postcondition is unifiable.

In the next example we show that if every pair in PG is unifiable it does not follow that PG is unifiable.

Example 23. Let

$$\begin{aligned}
g_1 &= (x_a \geq 5 \wedge x_a \leq 7 \wedge x_b \geq 5 \wedge x_b \leq 7),\\
\varphi_1 &= (x_a = x_b),\\
g_2 &= (x_a \geq 7 \wedge x_a \leq 9 \wedge x_b \geq 3 \wedge x_b \leq 5),\\
\varphi_2 &= (x_b = x_a - 4),\\
g_3 &= (x_a \geq 9 \wedge x_a \leq 11 \wedge x_b \geq 1 \wedge x_b \leq 3),\\
\varphi_3 &= (x_b = x_a - 8),
\end{aligned}$$

see Fig. 20(a). Let $PG = \{(\varphi_1, (a, g_1)), (\varphi_2, (a, g_2)), (\varphi_3, (a, g_3))\}$ and

$$\begin{aligned}
g_4 &= (x_a \geq 5 \wedge x_a \leq 9 \wedge x_b \geq 3 \wedge x_b \leq 7),\\
g_5 &= (x_a \geq 7 \wedge x_a \leq 11 \wedge x_b \geq 1 \wedge x_b \leq 5),\\
g_6 &= (x_a \geq 5 \wedge x_a \leq 11 \wedge x_b \geq 1 \wedge x_b \leq 7).
\end{aligned}$$

Then (a, g_4) is the strongest unifier for $\{(\varphi_1, (a, g_1)), (\varphi_2, (a, g_2))\}$, see Fig. 20(b), (a, g_5) is the strongest unifier for $\{(\varphi_2, (a, g_2)), (\varphi_3, (a, g_3))\}$, see Fig. 20(c) and (a, g_6) is the strongest unifier for $\{(\varphi_1, (a, g_1)), (\varphi_3, (a, g_3))\}$, see Fig. 20(d). Then the strongest possible unifier for PG should be g_6, but $\varphi_2 \wedge g_6 \not\equiv \varphi_2 \wedge g_2$. Then PG is not unifiable. □

It is easy to check, whether PG is unifiable, using the property that the guards in PG are tight in the sense of Definition 7. The basic idea in each step is to take the weakest upper and lower bounds for each variable. Assume the guard g_{ji} is given by its upper and lower bounds:

$$g_{ji} = \bigwedge_{a \in \Sigma} (x_a \leq c^{\leq}_{a,ji} \wedge x_a \geq c^{\geq}_{a,ji}).$$

For $i = 1, \dots, n$, define the candidate $\hat{g}_i$ as

$$\hat{g}_i = \bigwedge_a \left(x_a \leq \max_j \{c^{\leq}_{a,ji}\} \right) \wedge \bigwedge_a \left(x_a \geq \min_j \{c^{\geq}_{a,ji}\} \right)$$

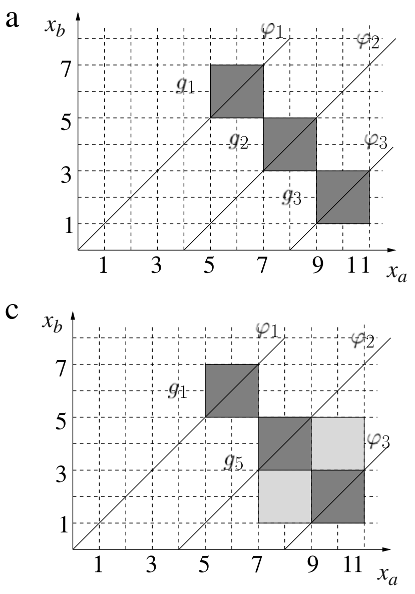
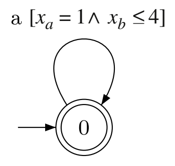


Fig. 20. Not unifiable $PG = \{(\varphi_1, (a, g_1)), (\varphi_2, (a, g_2)), (\varphi_3, (a, g_3))\}$.



T	a
u_1	$x_a = 1 \wedge x_b = 1$
u_2	$x_a = 1 \wedge x_b = 2$
u_3	$x_a = 1 \wedge x_b = 3$
u_4	$x_a = 1 \wedge x_b = 4$
u_5	$false$

(a) Automaton $\mathcal{A}_1$ (b) Table T

Fig. 21. A DERA to be learned and an observation table.

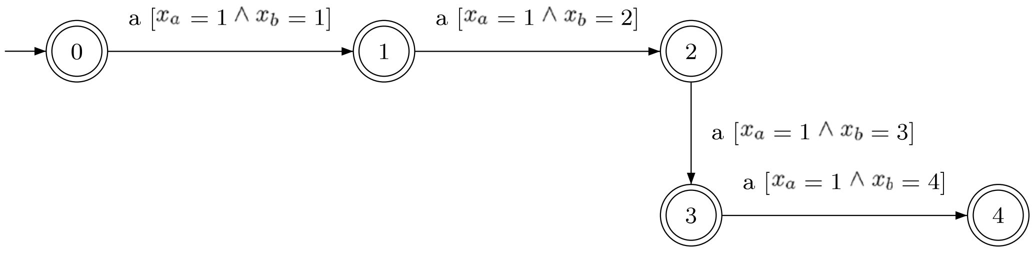

a $[x_a = 1 \wedge x_b = 1]$
a $[x_a = 1 \wedge x_b = 2]$
a $[x_a = 1 \wedge x_b = 3]$
a $[x_a = 1 \wedge x_b = 4]$
0
1
2
3
4

Fig. 22. Automaton $\mathcal{A}_2$.

and check whether the guarded word $(a_1, \hat{g}_1) \ldots (a_n, \hat{g}_n)$ obtained in this way is indeed a unifier. We represent *false* as the constraint $\bigwedge_{a \in \Sigma} x_a \leq 0 \wedge x_a \geq K + 1$. It can be shown that if *PG* is unifiable, then this candidate is the strongest possible unifier.

The learning algorithm using the idea of unified states works similar as the one for EDERAs. However, we employ a slightly different observation table. Let $\Gamma = \Sigma \times G_\Sigma$. Rows of the table are guarded words of a prefix-closed set $U \subseteq \Gamma^*$. Column labels are untimed words from a set $V \subseteq \Sigma^*$. The entries of the table are sequences of guards describing under which values the column label extends the row label. Thus, we define a *timed observation table* $T : U \cup U\Gamma \to (V \to G_\Sigma^*)$, where $T(u_g)(v) = g_1 \ldots g_n$ implies $|v| = n$. We require the initial observation table to be defined over $U = \{\lambda\}$ and $V = \Sigma$.

We define $u_g \in U \cup U\Gamma$ and $u'_g \in U \cup U\Gamma$ to be v-unifiable if $v = a_1 \ldots a_n \in V, T(u_g)(v) = g_1 \ldots g_n, T(u'_g)(v) = g'_1 \ldots g'_n$ and $\{(sp(u_g), (a_1, g_1) \ldots (a_n, g_n)), (sp(u'_g), (a_1, g'_1) \ldots (a_n, g'_n))\}$ is unifiable. We define $u_g \in U \cup U\Gamma$ and $u'_g \in U \cup U\Gamma$ to be unifiable if for every $v \in V$, u_g and u'_g are v-unifiable.

A timed observation table is *closed* if for every $u_g \in U\Gamma$ there is $u'_g \in U$ such that u_g and u'_g are unifiable. A timed observation table is *consistent* if for all $u_g, u'_g \in U$ whenever u_g and u'_g unifiable, and $u_g(a, g), u'_g(a, g') \in U \cup U\Gamma$ then $u_g(a, g)$ and $u'_g(a, g')$ are unifiable.

A *merging* of the timed observation table T consists of a partition Π of the guarded words $U \cup U\Gamma$, and an assignment of a clock guard $CG(\pi, a)$ to each block $\pi \in \Pi$ and action $a \in \Sigma$, such that for each block $\pi \in \Pi$ we have

- for each suffix $v = a_1 \dots a_n \in V$, the set $\{\langle sp(u_g), (a_1, g_1) \dots (a_n, g_n)\rangle \mid u_g \in \pi, T(u_g)(v) = g_1 \dots g_n\}$ is unifiable,
- if $u_g, u'_g \in \pi$ and $u_g(a, g), u'_g(a, g') \in U \cup U\Gamma$ then $u_g(a, g), u'_g(a, g') \in \pi'$ for some block π' in Π, and
- $(a, CG(\pi, a))$ is the unifier for $\{(sp(u_g), (a, g')) \mid u_g \in \pi, T(u_g)(a) = g'\}$ for each $a \in \Sigma$.

Intuitively, a merging defines a grouping of rows into blocks, each of which can potentially be understood as a state in a EDERA, together with a choice of clock guard for each action and block, which can be understood as a guard for the action in the EDERA. For each table there are in general several possible mergings, but the number of mergings is bounded, since the number of partitions is bounded, and since the number of possible unifiers $GC(\pi, a)$ is also bounded.

A *coarsest merging* of the timed observation table T is a merging with a minimal number of blocks. From a closed and consistent timed table we can get a lower bound on the number of blocks. It follows from Example 23 that in order to construct a coarsest merging we need to check whether all rows in a block are unifiable.

Given a merging (Π, GC) of a closed and consistent timed observation table T, one can construct the EDERA $\mathcal{H} = (\Sigma, L, l_0, \varrho, \eta)$, where

- $L = \Pi$ comprises the blocks of Π as locations,
- $l_0 = \pi \in \Pi$ with $\lambda \in \pi$ is the initial location,
- $\varrho(\pi, a) = \pi'$, if there are $u \in \pi$ and g such that $u \in U$, $u(a, g) \in \pi'$ and $T(u)((a, g)) \neq$ *false*, otherwise $\varrho(\pi, a)$ is undefined.
- η is defined by $\eta(\pi, a) = GC(\pi, a)$.

7.2. Algorithm TL^*_{nsg}

The algorithm TL^*_{nsg} for learning (non-sharply guarded) EDERAs is as TL^*_{sg}, except that the new notions of closed and consistent are used. If in Algorithm 2 the check for closeness and consistency compares two rows in a table on equality, the algorithm TL^*_{nsg} checks whether two rows are unifiable. Since the entries of a timed observation table are sequences of guards, we introduce the function *learn_guard*, which learns the guard refinement of some suffix of a word. One further modification is that the hypothesis is constructed as described in the previous paragraph, using the computed merging. The rest of the algorithm remains unchanged (see Algorithm 8).

Algorithm 7 Pseudo code for Assistant of TL^*_{nsg}

```
class $TL^*_{nsg}$Assistant extends $TL^*_{sg}$Assistant
    Teacher timedteacher

Constructor $TL^*_{nsg}$Assistant(teacher)
    timedteacher = teacher

Function learn_guard($u_g$,$w$)
    Extract underlying $u$ of $u_g$
    Learn guard refinement $u_g(a_1, g_1) \dots (a_n, g_n)$ of $uw$
    return $g_1 \dots g_n$
```

Lemma 24. *Let $\mathcal{A}$ be a smallest EDERA equivalent to the system that is to be learned. Let $|\mathcal{A}|$ be the number of locations in $\mathcal{A}$. Then the algorithm TL^*_{nsg} terminates and constructs an EDERA $\mathcal{A}'$ with $|\mathcal{A}|$ locations, which is equivalent to $\mathcal{A}$.*

Proof. We first prove that every coarsest merging constructed from a timed observation table has at most $|\mathcal{A}|$ blocks. Assume that the algorithm TL^*_{nsg} constructs the timed observation table $T : U \cup U\Gamma \to (V \to G^*)$. Assume that $u^1_g, \dots, u^n_g$ are all rows in $U \cup U\Gamma$ such that $v^1_g, \dots, v^n_g$ lead to the same location l in $\mathcal{A}$ and for each $1 \leq i \leq n$, the word underlying v^i_g is equal to the word underlying u^i_g. Since $w_t \vDash u^i_g$ iff $w_t \vDash v^i_g$, we infer $sp(v^i_g) = sp(u^i_g)$ for each $1 \leq i \leq n$. Let $a_1 \dots a_m \in V$ and $T(u^i_g)(a_1 \dots a_m) = g_{i1} \dots g_{im}$ for each $1 \leq i \leq n$. Let $v^1_g(a_1, g_1) \dots (a_m, g_m)$ lead to some location in $\mathcal{A}$. Since $w_t \vDash v^i_g(a_1, g_1) \dots (a_m, g_m)$ iff $w_t \vDash u^i_g(a_1, g_{i1}) \dots (a_m, g_{im})$, we infer that for each $1 \leq i \leq n$ and $1 \leq j \leq m-1$ we have $sp(v^i_g(a_1, g_1) \dots (a_j, g_j)) = sp(u^i_g(a_1, g_{i1}) \dots (a_j, g_{ij}))$ and both

$$sp(v^i_g) \wedge g_1 \equiv sp(u^i_g) \wedge g^i_1, \quad \text{and}$$
$$sp(v^i_g(a_1, g_1) \dots (a_j, g_j)) \wedge g_{j+1} \equiv sp(u^i_g(a_1, g_{i1}) \dots (a_j, g_{ij})) \wedge g_{i(j+1)}.$$

Algorithm 8 Pseudo code for TL^*_{nsg}

```
class $TL^*_{nsg}$ extends $\mathbf{L}^*${
  Alphabet $\Sigma$

Constructor $TL^*_{nsg}(timedteacher,\Sigma,K)$
      $\Gamma = TL^*_{nsg}$Alphabet$(\Sigma, K)$
      $this.\Sigma = \Sigma$
      $teacher = TL^*_{nsg}$Assistant$(timedteacher)$

Function initialize($(U, V, T)$)
    $U := \{\lambda\}, V := \Sigma$
    for every $a \in \Sigma$
        g=$teacher$.learn_guard($\lambda$, $a$)
        if $g \neq false$
            Add $(a, g)$ to $U\Gamma$
    for every $u_g \in U \cup U\Gamma$ and $a \in \Sigma$
        $T(u_g)(a)$=$teacher$.learn_guard($u_g$, $a$)

Function isClosed($(U, V, T)$)
    if for each $u_g \in U\Gamma$ there is $u'_g \in U$ such that $u_g$ and $u'_g$ are unifiable then
        return true
    else
        return false

Function isConsistent($(U, V, T)$)
    if for each $a \in \Sigma$ and $u_g, u'_g \in U$ such that $u_g(a, g), u'_g(a, g') \in U \cup U\Gamma$, and
    $u_g$ and $u'_g$ are unifiable we have $u_g(a, g)$ and $u'_g(a, g')$ are unifiable then
        return true
    else
        return false

Function add_column()
  Find $a \in \Sigma$, $v \in V$, $u_g, u'_g \in U$ and $u_g(a, g), u'_g(a, g') \in U \cup U\Gamma$ such that
$u_g$ and $u'_g$ are unifiable, but $u_g(a, g)$ and $u_g(a, g')$ are
not$v$-unifiable
  Add $av$ to $V$
  for every $u_g \in U \cup U\Gamma$
      $T(u_g)(av)$=$teacher$.learn_guard($u_g$,$av$)

Function move_row()
  Find $u_g \in U\Gamma$ such that for all $u'_g \in U$, $u_g$ and $u'_g$ are not unifiable
  Move $u_g$ to $U$
  for every $a \in \Sigma$
      $g = teacher$.learn_guard($u_g$,$a$)
      if $g \neq false$
          Add $u_g(a, g)$ to $U\Gamma$
          for every $v \in V$
              $T(u_g(a, g))(v)$=$teacher$.learn_guard($u_g(a, g)$,$v$)

Function process_counterexample(u_g)
  Add every prefix $u'_g$ of $u_g$ to $U$.
  for every $v \in V$ and prefix $u'_g$ of $u_g$
      $T(u'_g)(v)$=$teacher$.learn_guard($u'_g$,$v$)
  for every $a \in \Sigma$ and prefix $u'_g$ of $u_g$
      $g = teacher$.learn_guard($u'_g$,$a$)
      if $g \neq false$
          Add $u'_g(a, g)$ to $U\Gamma$
          for every $v \in V$
              $T(u'_g(a, g))(v)$=$teacher$.learn_guard($u'_g(a, g)$,$v$)
}
```

Then the set $\{(sp(u^i_g), (a_1, g_{i1}) \dots (a_m, g_{im})) | 1 \leq i \leq n\}$ is unifiable. Let $a'_1 \dots a'_k \in V$. We can use the same argument to show that for every $a \in \Sigma$, the set

$$\{(sp(u^i_g(a, g'_i)), (a'_1, g'_{i1}) \dots (a'_k, g'_{ik})) | 1 \leq i \leq n, u^i_g(a, g'_i) \in U \cup U\Gamma\}$$

is unifiable. Since $a_1 \dots a_n$ and $a'_1 \dots a'_k$ were chosen arbitrarily, we can conclude that there is a merging Π of T such that $u^1_g, \dots, u^n_g$ are in the same block. Thus a coarsest merging of T can contain at most $|\mathcal{A}|$ blocks.

It follows that there can be at most $|\mathcal{A}|$ rows in U such that no pair of them is unifiable. Then the number of calls to the function *move_row* in Algorithm 8 is bounded, and hence a closed table can be constructed. If for every $u_g, u'_g \in U$, u_g and

u'_g are not unifiable then the table is also consistent. Thus we need a bounded number of calls to the function *add_column* to make the timed observation table consistent.

Since the number of blocks in a coarsest merging is bounded by $|\mathcal{A}|$, and the number of automata of the same size is bounded, then the algorithm TL^*_{nsg} terminates and constructs an automaton $\mathcal{A}'$ with $|\mathcal{A}|$ locations. $\square$

Roughly, TL^*_{nsg} can be understood as TL^*_{sg} plus merging. Therefore, in the worst case, more steps and therefore queries are needed as in TL^*_{sg}. However, when a small non-sharply guarded EDERA represents a large sharply guarded EDERA, TL^*_{nsg} will terminate using less queries. Therefore, a better performance can be expected in practice.

Example 25. Let us the algorithm TL^*_{nsg} on a small example. Let the automaton $\mathcal{A}_1$ shown in Fig. 21(a) be the EDERA to learn. After a number of queries of the algorithm TL^*_{nsg}, we obtain the observation table T shown in Fig. 21(b), where the guarded words u_1 – u_5 are defined by

$$\begin{aligned}
u_1 &= \lambda\\
u_2 &= (a, x_a = 1 \wedge x_b = 1)\\
u_3 &= (a, x_a = 1 \wedge x_b = 1)(a, x_a = 1 \wedge x_b = 2)\\
u_4 &= (a, x_a = 1 \wedge x_b = 1)(a, x_a = 1 \wedge x_b = 2)(a, x_a = 1 \wedge x_b = 3)\\
u_5 &= u_4(a, x_a = 1 \wedge x_b = 4).
\end{aligned}$$

It turns out that all rows of T are unifiable. Define PG by

$$\begin{aligned}
PG = \{&\langle sp(u_1), (a, x_a = 1 \wedge x_b = 1)\rangle,\\
&\langle sp(u_2), (a, x_a = 1 \wedge x_b = 2)\rangle,\\
&\langle sp(u_3), (a, x_a = 1 \wedge x_b = 3)\rangle,\\
&\langle sp(u_4), (a, x_a = 1 \wedge x_b = 4)\rangle,\\
&\langle sp(u_5), (a, \mathit{false})\rangle\}
\end{aligned}$$

where *false* represents constraint $x_a \le 0 \wedge x_a \ge 5$. It can be checked that the guarded word $(a, x_a = 1 \wedge x_b \le 4)$ unifies PG. We will use the merging of the observation table T as the partition which consists of exactly one block, and equipping the action a with the guard $x_a = 1 \wedge x_b \le 4$. The automaton obtained from this mergings is the automaton $\mathcal{A}_1$ which consists of exactly one state. In contrast, the algorithm TL^*_{sg}, which does not employ unification, would construct the sharply guarded EDERA $\mathcal{A}_2$ shown in Fig. 22. The automaton $\mathcal{A}_2$ has 5 states, since table T has 5 different rows. $\square$

8. Conclusion

In this paper, we presented techniques for learning timed systems that can be represented as event-recording automata.

By considering the restricted class of event-deterministic automata, we can uniquely represent an automaton by a regular language of guarded words, and the learning algorithm can identify states by access strings that are untimed sequences of actions. 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 constructed automaton has the form of a zone graph, which, in general, can be doubly exponentially larger than a minimal DERA representing the same language, but for many practical systems the zone graph construction does not lead to a severe explosion, as exploited by tools for timed automata verification [6,9]. The resulting algorithm is called TL^*_{sg}. The query complexity of TL^*_{sg} for a sharply guarded EDERA with n locations is

$$O\left(kn^2ml\binom{|\Sigma| + K}{|\Sigma|}\right)$$

where l is the length of the longest guarded word queried and k is the size of alphabet $\Sigma \times G$. The query complexity of TL^*_{sg} for EDERA with n locations is

$$O\left(kn^2 2^{2|\Sigma|!2^{|\Sigma|}(2K+2)^{|\Sigma|}}ml\binom{|\Sigma| + K}{|\Sigma|}\right).$$

We also introduced the algorithm TL^*_{nsg} also for learning event-deterministic automata, which simultaneously reduces the size of the automaton learned so far, however, for the price of a larger worst-case complexity.

Without the restriction of event-determinism, the problem of learning guards is significantly less tractable. We present the algorithm TL^*_s that learns general DERA. The drawback of the algorithm that it constructs a DERA in spirit of a region graph. The query complexity of TL^*_s for the simple DERA with n locations is $O(|\Sigma|^2n^2mK)$. In verification for timed automata, it is a well-known fact, that despite theoretically lower worst-case complexity, algorithms based on region graphs perform less efficiently than algorithms based on zone graphs.

Together with [17], this paper describes initial efforts on learning algorithms for timed systems. Future work has to show the benefit of these algorithms in practical applications as well as to examine the most suitable application area for each algorithm.

Acknowledgements

This work was supported in part by the European Research Training Network GAMES, and Project 231167 CONNECT.

References

[1] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.
[2] R. Alur, L. Fix, T. Henzinger, Event-clock automata: a determinizable class of timed automata, Theoretical Computer Science 211 (1999) 253–273.
[3] R. Alur, Timed automata, in: Proc. 11th International Computer Aided Verification Conference, in: Lecture Notes in Computer Science, vol. 1633, Springer-Verlag, 1999, pp. 8–22.
[4] D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation 75 (1987) 87–106.
[5] J. L. Balcázar, J. Díaz, R. Gavaldá, Algorithms for learning finite automata from queries: a unified view, in: Advances in Algorithms, Languages, and Complexity, Kluwer, 1997, pp. 53–72.
[6] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, S. Yovine, Kronos: A model-checking tool for real-time systems, in: A.J. Hu, M.Y. Vardi (Eds.), Proc. 10th International Conference on Computer Aided Verification, Vancouver, Canada, in: Lecture Notes in Computer Science, vol. 1427, Springer-Verlag, 1998, pp. 546–550.
[7] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, B. Steffen, On the correspondence between conformance testing and regular inference, in: M. Cerioli (Ed.), Proc. FASE'05, 8th Int. Conf. on Fundamental Approaches to Software Engineering, in: Lecture Notes in Computer Science, vol. 3442, 2005, pp. 175–189.
[8] T. Berg, B. Jonsson, H. Raffelt, Regular inference for state machines using domains with equality tests, in: J.L. Fiadeiro, P. Inverardi (Eds.), Proc. FASE'08, 11th Int. Conf. on Fundamental Approaches to Software Engineering, in: Lecture Notes in Computer Science, vol. 4961, 2008, pp. 317–331.
[9] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, Wang Yi, UPPAAL: a tool suite for the automatic verification of real-time systems, in: R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III, in: Lecture Notes in Computer Science, vol. 1066, Springer-Verlag, 1996, pp. 232–243.
[10] J. Bengtsson, W. Yi, On clock difference constraints and termination in reachability analysis of timed automata, in: J.S. Dong, J. Woodcock (Eds.), Proc. ICFEM 2003, 5th Int. Conf. on Formal Engineering Methods, Singapore, in: Lecture Notes in Computer Science, vol. 2885, Springer Verlag, Nov. 2003, pp. 491–503.
[11] T.S. Chow, Testing software design modeled by finite-state machines, IEEE Transactions on Software Engineering, SE-4 3 (1978) 178–187.
[12] J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, H. Zheng, Bandera: extracting finite-state models from java source code, in: Proc. 22nd Int. Conf. on Software Engineering, June 2000.
[13] E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.
[14] D.L. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Automatic Verification Methods for Finite State Systems, 1989, pp. 197–212.
[15] C. Daws, S. Tripakis, Model checking of real-time reachability properties using abstractions, in: TACAS, 1998, pp. 313–329.
[16] J.-C. Fernandez, C. Jard, T. Jéron, C. Viho, An experiment in automatic generation of test suites for protocols with verification technology, Science of Computer Programming 29 (1997).
[17] O. Grinchtein, B. Jonsson, P. Pettersson, Inference of event-recording automata using timed decision trees, in: C. Baier, H. Hermanns (Eds.), Proc. CONCUR 2006, 17th Int. Conf. on Concurrency Theory, in: LNCS, vol. 4137, 2006, pp. 435–449.
[18] E.M. Gold, Language identification in the limit, Information and Control 10 (1967) 447–474.
[19] A. Groce, D. Peled, M. Yannakakis, Adaptive model checking, in: J.-P. Katoen, P. Stevens (Eds.), Proc. TACAS'02, 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 2280, Springer Verlag, 2002, pp. 357–370.
[20] O. Grinchtein, Learning of Timed Systems. Ph.D. Thesis, Uppsala University, 2008.
[21] A. Hagerer, H. Hungar, O. Niese, B. Steffen, Model generation by moderated regular extrapolation, in: R.-D. Kutsche, H. Weber (Eds.), Proc. FASE'02, 5th Int. Conf. on Fundamental Approaches to Software Engineering, in: Lecture Notes in Computer Science, vol. 2306, Springer Verlag, 2002, pp. 80–95.
[22] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, M.B. Trakhtenbrot, STATEMATE: a working environment for the development of complex reactive systems, IEEE Transactions on Software Engineering 16 (4) (1990) 403–414.
[23] T.A. Henzinger, Z. Manna, A. Pnueli, Temporal proof methodologies for timed transition systems, Information and Computation 112 (1994) 173–337.
[24] H. Hungar, O. Niese, B. Steffen, Domain-specific optimization in automata learning, in: Proc. 15th Int. Conf. on Computer Aided Verification, 2003.
[25] G.J. Holzmann, Logic verification of ANSI-C code with SPIN, in: K. Havelund, J. Penix, W. Visser (Eds.), SPIN Model Checking and Software Verification: Proc. 7th Int. SPIN Workshop, in: Lecture Notes in Computer Science, vol. 1885, Springer Verlag, Stanford, CA, 2000, pp. 31–147.
[26] T.A. Henzinger, J.-F. Raskin, P.-Y. Schobbens, The regular real-time languages, in: K.G. Larsen, S. Skuym, G. Winskel (Eds.), Proc. ICALP'98, 25th International Colloquium on Automata, Languages, and Programming, in: Lecture Notes in Computer Science, vol. 1443, Springer Verlag, 1998, pp. 580–591.
[27] M.J. Kearns, U.V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, 1994.
[28] O. Maler, A. Pnueli, On recognizable timed languages, in: Proc. FOSSACS04, Conf. on Foundations of Software Science and Computation Structures, in: Lecture Notes in Computer Science, Springer-Verlag, 2004.
[29] R.L. Rivest, R.E. Schapire, Inference of finite automata using homing sequences, Information and Computation 103 (1993) 299–347.
[30] M. Schmitt, M. Ebner, 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.
[31] J. Springintveld, F. Vaandrager, Minimizable timed automata, in: B. Jonsson, J. Parrow (Eds.), Proc. FTRTFT'96, Formal Techniques in Real-Time and Fault-Tolerant Systems, Uppsala, Sweden, in: Lecture Notes in Computer Science, vol. 1135, Springer Verlag, 1996, pp. 130–147.
[32] J. Springintveld, F. Vaandrager, P.R. D'Argenio, Testing timed automata, Theoretical Computer Science 254 (1–2) (2001) 225–257.
[33] M.P. Vasilevskii, Failure diagnosis of automata, Kibernetika 4 (1973) 98–108.
[34] S.E. Verwer, M.M. de Weerdt, C. Witteveen, Identifying an automaton model for timed data, in: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn), Benelearn, 2006, pp. 57–64.
[35] T. Wilke, Specifying timed state sequences in powerful decidable logics and timed automata, in: H Langmaack, W.P. de Roever, J. Vytopil (Eds.), Proc. FTRTFT'94, Formal Techniques in Real-Time and Fault-Tolerant Systems, Lübeck, Germany, in: Lecture Notes in Computer Science, vol. 863, Springer Verlag, 1994, pp. 694–715.
[36] S. Yovine, Model checking timed automata, in: European Educational Forum: School on Embedded Systems, 1996, pp. 114–152.