10-2011

An efficient algorithm for learning event-recording automata

Shang-Wei LIN

Étienne ANDRÉ

Jin Song DONG

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Yang LIU

Citation

LIN, Shang-Wei; ANDRÉ, Étienne; DONG, Jin Song; SUN, Jun; and LIU, Yang. An efficient algorithm for learning event-recording automata. (2011). *Proceedings of the 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14*. 463-472.
Available at: https://ink.library.smu.edu.sg/sis_research/5026

An Efficient Algorithm for Learning Event-Recording Automata[⋆]

Shang-Wei Lin[1], Étienne André[1], Jin Song Dong[1], Jun Sun[2], and Yang Liu[1]

[1] School of Computing, National University of Singapore
{linsw,andre,dongjs,liuyang}@comp.nus.edu.sg
[2] Singapore University of Technology and Design
{sunjun}@sutd.edu.sg

Abstract. In inference of untimed regular languages, given an unknown language to be inferred, an automaton is constructed to accept the unknown language from answers to a set of membership queries each of which asks whether a string is contained in the unknown language. One of the most well-known regular inference algorithms is the L^* algorithm, proposed by Angluin in 1987, which can learn a minimal deterministic finite automaton (DFA) to accept the unknown language. In this work, we propose an efficient polynomial time learning algorithm, TL^*, for timed regular language accepted by event-recording automata. Given an unknown timed regular language, TL^* first learns a DFA accepting the untimed version of the timed language, and then passively refines the DFA by adding time constraints. We prove the correctness, termination, and minimality of the proposed TL^* algorithm.

1 Introduction

In formal verification such as model checking [4,13], system models and properties are assumed to be a priori during the verification process. However, modeling a system appropriately is not an easy task because if the model is too abstract, it may not describe the exact behavior of the system; if the model is too detailed, it suffers from the state space explosion problem. Thus an automatic inference or construction of abstract model is very helpful for system development.

In 1987, Angluin [3] proposed the L^* learning algorithm for inference of regular languages. Given an unknown language U to be inferred, L^* learns a minimal deterministic finite automaton (DFA) to accept U from answers to a set of membership queries each of which asks whether a string is contained in U.

After the L^* algorithm was proposed, it is widely used in several research fields. The most impressive one is that Cobleigh et al. [5] used the L^* algorithm to automatically generate the assumptions needed in assume-guarantee reasoning (AGR), which can alleviate the state explosion problem of model checking. Another interesting work is that Lin and Hsiung proposed a compositional synthesis

[⋆] This research is supported by the research grant MOE2009-T2-1-072 (Advanced Model Checking Systems) in School of Computing, National University of Singapore.

framework, CAGS [10], based on the L* algorithm to automatically eliminate all behavior violating the user-given properties.

However, there were almost no extensions of the learning algorithm to inference timed regular languages until 2004, Grinchtein et al. [7,8] proposed a learning algorithm for event-recording automata [2] based on L*. Grinchtein's learning algorithm, TL$^*_{sg}$, uses region construction to actively guess all possible time constraints for each untimed word. That is, each original membership query of an untimed word in L* gives rise to several membership queries of timed words with possible time constraints, which increases the number of membership queries exponentially with the largest constant appearing in the time constraints.

In this work, we propose an efficient *polynomial time* learning algorithm TL* for timed regular languages accepted by event-recording automata. *Event-recording automata* (ERA) [2] are a determinizable subclass of timed automata [1] such that a timed language accepted by an ERA can be classified into finite number of classes. Given a timed regular language U_T accepted by ERA, TL* first learns a DFA M accepting U (the untimed version of U_T) and then passively refines M by adding time constraints. Thus the number of membership queries required by TL* is much smaller than that of Grinchtein's algorithm. We prove that the TL* algorithm will correctly learn an ERA accepting the unknown language U_T after a finite number of iterations. Further, we also prove the minimality of our TL* algorithm, i.e., the number of locations of the ERA learned by TL* is minimal.

This paper is organized as follows: Section 2 gives preliminary knowledge and introduces the L* algorithm. The proposed efficient learning algorithm, TL*, is described in Section 3. The conclusion and future work are given in Section 4.

2 Preliminaries

We give some background knowledge about timed languages and event-recording automata in Section 2.1 and introduce the L* algorithm in Section 2.2.

2.1 Timed Languages and Event-Recording Automata

Let Σ be a finite alphabet. A *timed word* over Σ is a finite sequence $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \dots, n\}$ that are paired with nonnegative real numbers $t_i \in \mathbb{R}^+$ such that the sequence $\overline{t} = t_1 t_2 \dots t_n$ of time-stamps is nondecreasing. For every symbol $a \in \Sigma$, we use x_a to denote the *event-recording clock* of a [2]. Intuitively, x_a records the time elapsed since the last occurrence of the symbol a. We use C_Σ to denote the set of event-recording clocks over Σ, i.e., $C_\Sigma = \{x_a \mid a \in \Sigma\}$. A *clock valuation* $\gamma : C_\Sigma \mapsto \mathbb{R}^+$ assigns a nonnegative real number to an event-recording clock.

A *clocked word* over Σ is a finite sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \dots, n\}$ that are paired with clock valuations γ_i such that $\gamma_1(x_a) = \gamma_1(x_b)$ for all $a, b \in \Sigma$ and $\gamma_i(x_a) = \gamma_{i-1}(x_a) + \gamma_i(x_{a_{i-1}})$ when $1 < i \le n$ and $a \ne a_{i-1}$. Each timed word $w_t = (a_1, t_1)(a_2, t_2)\dots(a_n, t_n)$ can be naturally transformed into a clocked word $cw(w_t) = (a_1, \gamma_1)(a_2, \gamma_2)\dots(a_n, \gamma_n)$

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

A *clock guard* g is a conjunction of constraints of the form $x_a \sim n$ for $x_a \in C_\Sigma$, $n \in \mathbb{N}$, and $\sim \in \{<, \leq, >, \geq\}$. A clock guard g identifies a hypercube zone $[\![g]\!] \subseteq (\mathbb{R}^+)^{|\Sigma|}$. We use G_Σ to denote the set of clock guards over C_Σ. A *guarded word* is a sequence $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ where $a_i \in \Sigma$ for $i \in \{1, 2, \ldots, n\}$ and $g_i \in G_\Sigma$ is a clock guard. 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 $\gamma_i \models g_i$ for all $i \in \{1, 2, \ldots, n\}$.

Definition 1. ***(Event-Recording Automata)*** *[2]. 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 $\delta :\subseteq L \times \Sigma \times G_\Sigma \mapsto 2^L$. An ERA is* deterministic *if $\delta(l, a, g)$ is a singleton set when it is defined, and when both $\delta(l, a, g_1)$ and $\delta(l, a, g_2)$ are both defined then $[\![g_1]\!] \cap [\![g_2]\!] = \emptyset$, where $l \in L$, $a \in \Sigma$, and $g_1, g_2 \in G_\Sigma$. A deterministic ERA is* complete *if for all $l \in L$ and for all $a \in \Sigma$, $\delta(l, a, g_i)$ is defined for all $i \in \{1, 2, \ldots, n\}$ such that $[\![g_1]\!] \cup [\![g_2]\!] \cup \ldots \cup [\![g_n]\!] = [\![true]\!]$. A guarded word $w_g = (a_1, g_1)(a_2, g_2) \ldots (a_n, g_n)$ is* accepted *by an ERA $D = (\Sigma, L, l_0, \delta, L^f)$ if $l_i = \delta(l_{i-1}, a_i, g_i)$ is defined for all $i \in \{1, 2, \ldots, n\}$ and $l_n \in L^f$. The* language *accepted by D, denoted by $\mathcal{L}(D)$, is the set of guarded words accepted by D.*

Note that in an ERA, each event-recording clock $x_a \in C_\Sigma$ is implicitly and automatically reset when a transition with event a is taken, which gives a good characteristic that each non-deterministic ERA can be determinized by subset construction [2]. Fig. 1 (a) p. 467 gives a deterministic ERA $\mathcal{A}_1$ accepting the timed word $(a, t_1)(a, t_2)(a, t_3) \ldots$, where $t_{2i} = t_{2i-1} + 3$ and $t_{2i+1} = t_{2i} + 1$ for $i \in \mathbb{N}$. We can also use a clocked word $(a, \gamma_1)(a, \gamma_2)(a, \gamma_3) \ldots$ to represent the timed word such that $\gamma_{2i-1}(x_a) = 1$ and $\gamma_{2i}(x_a) = 3$ for $i \in \mathbb{N}$. Or we can use a guarded word $(a, g_1)(a, g_2)(a, g_3) \ldots$ to represent the timed word such that $g_{2i-1} = (x_a = 1)$ and $g_{2i} = (x_a = 3)$ for $i \in \mathbb{N}$. Thus $\mathcal{A}_1$ accepts the timed language $\mathcal{L}(\mathcal{A}_1) = ((a, x_a = 1)(a, x_a = 3))^*$.

2.2 The L* Algorithm

The L* algorithm [3] is a formal method to learn a minimal DFA (with the minimal number of locations) that accepts an unknown language U over an alphabet Σ. During the learning process, L* interacts with a *Minimal Adequate Teacher* (Teacher for short) to ask *membership* and *candidate queries*. A *membership query* for a string σ is a function $\mathcal{Q}_m$ such that if $\sigma \in U$, then $\mathcal{Q}_m(\sigma) = 1$; otherwise, $\mathcal{Q}_m(\sigma) = 0$. A *candidate query* for a DFA M is a function $\mathcal{Q}_c$ such that if $\mathcal{L}(M) = U$, then $\mathcal{Q}_c(M) = 1$; otherwise, $\mathcal{Q}_c(M) = 0$. The results of membership queries are stored in an *observation table* (S, E, T) where $S \subseteq \Sigma^*$ is a set of prefixes, $E \subseteq \Sigma^*$ is a set of suffixes, and $T : (S \cup S \cdot \Sigma) \times E \mapsto \{0, 1\}$ is a mapping function such that if $s \cdot e \in U$, then $T(s, e) = 1$; otherwise, i.e., $s \cdot e \notin U$, then $T(s, e) = 0$, where $s \in (S \cup S \cdot \Sigma)$ and $e \in E$. The L* algorithm categorizes strings based on Myhill-Nerode Congruence [9].

Algorithm 1. L* Algorithm

```
input  : $\Sigma$: alphabet
output: a DFA accepting the unknown language $U$
1  Let $S = E = \{\lambda\}$ ;
2  Update $T$ by $\mathcal{Q}_m(\lambda)$ and $\mathcal{Q}_m(\lambda \cdot \alpha)$, for all $\alpha \in \Sigma$ ;
3  while true do
4      while there exists $(s \cdot \alpha)$ such that $(s \cdot \alpha) \not\equiv s'$ for all $s' \in S$ do
5          $S \longleftarrow S \cup \{s \cdot \alpha\}$ ;
6          Update $T$ by $\mathcal{Q}_m((s \cdot \alpha) \cdot \beta)$, for all $\beta \in \Sigma$ ;
7      Construct candidate DFA $M$ from $(S, E, T)$ ;
8      if $\mathcal{Q}_c(M) = 1$ then return $M$ ;
9      else
10         $\sigma_{ce} \longleftarrow$ the counterexample given by Teacher ;
11         $E \longleftarrow E \cup \{v\}$ where $v = WS(\sigma_{ce})$ ;
12         Update $T$ by $\mathcal{Q}_m(s \cdot v)$ and $\mathcal{Q}_m(s \cdot \alpha \cdot v)$, for all $s \in S$ and $\alpha \in \Sigma$ ;
```

Definition 2. ***Myhill-Nerode Congruence.*** *For any two strings $\sigma, \sigma' \in \Sigma^*$, we say they are* equivalent*, denoted by $\sigma \equiv \sigma'$, if $\sigma \cdot \rho \in U \Leftrightarrow \sigma' \cdot \rho \in U$, for all $\rho \in \Sigma^*$. Under the equivalence relation, we can say σ and σ' are the representing strings of each other, denoted by $\sigma = [\sigma']_r$ and $\sigma' = [\sigma]_r$.*

L* will always keep the observation table *closed* and *consistent.* An observation table is *closed* if for all $s \in S$ and $\alpha \in \Sigma$, there always exists $s' \in S$ such that $s \cdot \alpha \equiv s'$. An observation table is *consistent* if for every two elements $s, s' \in S$ such that $s \equiv s'$, then $(s \cdot \alpha) \equiv (s' \cdot \alpha)$ for all $\alpha \in \Sigma$. Once the table (S, E, T) is closed and consistent, the L* algorithm will construct a corresponding candidate DFA $C = (\Sigma_C, L_C, l_C^0, \delta_C, L_C^f)$ such that $\Sigma_C = \Sigma$, $L_C = S$, $l_C^0 = \{\lambda\}$, $\delta_C(s, \alpha) = [s \cdot \alpha]_r$ for $s \in S$ and $\alpha \in \Sigma$, and $L_C^f = \{s \in S \mid T(s, \lambda) = 1\}$.

Subsequently, L* makes a candidate query for C. If $\mathcal{L}(C) \neq U$, Teacher gives a counterexample σ_{ce} such that σ_{ce} is *positive* if $\sigma_{ce} \in \mathcal{L}(U) \setminus \mathcal{L}(C)$; *negative* if $\sigma_{ce} \in \mathcal{L}(C) \setminus \mathcal{L}(U)$. L* analyzes the counterexample σ_{ce} to find the witness suffix. A *witness suffix* is a string that when appended to two strings provides enough evidence for the two strings to be classified into two different equivalence classes under the Myhill-Nerode Congruence. Given an observation table (S, E, T) and a counterexample σ_{ce}, we define an *i-decomposition query* of σ_{ce}, denoted by $\mathcal{Q}_m^i(\sigma_{ce})$, as follows: $\mathcal{Q}_m^i(\sigma_{ce}) = \mathcal{Q}_m([u_i]_r \cdot v_i)$ where $\sigma_{ce} = u_i \cdot v_i$ with $|u_i| = i$, and $[u_i]_r$ is the representing string of u_i in S. The *witness suffix* of σ_{ce}, denoted by $WS(\sigma_{ce})$, is the suffix v_i of σ_{ce} such that $\mathcal{Q}_m^i(\sigma_{ce}) \neq \mathcal{Q}_m^0(\sigma_{ce})$. Once the witness suffix $WS(\sigma_{ce})$ is obtained, L* uses it to refine the candidate C until $\mathcal{L}(C) = \mathcal{L}(U)$. The pseudo-code of the L* algorithm is given in Algorithm 1.

Assume Σ is the alphabet of the unknown regular language U and the number of locations of the minimal DFA is n. The L* algorithm needs $n - 1$ candidate queries and $O(|\Sigma|n^2 + n \log m)$ membership queries to learn the minimal DFA, where m is the length of the longest counterexample returned by Teacher.

3 An Efficient Algorithm for Learning ERA

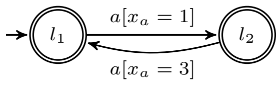
The intuition behind the L* algorithm is to classify untimed words into the minimal finite number of classes by performing membership queries, and each class can be represented by a location of a DFA. Because event-recording automata (ERA) are determinizable, a timed language (guarded words) accepted by an ERA can also be classified into a finite number of classes. The TL* algorithm tries to find the finite and minimal number of classes (locations).

3.1 The TL* Algorithm

Given a timed language U_T, the proposed TL* algorithm interacts with a timed Teacher to make two types of queries: the *timed membership* and *timed candidate queries*. A *timed membership query* for a guarded word w_g is a function $\mathcal{Q}_{m^T}$ such that $\mathcal{Q}_{m^T}(w_g) = 1$ if $w_g \in U_T$; otherwise $\mathcal{Q}_{m^T}(w_g) = 0$. A *timed candidate query* for an ERA M is a function $\mathcal{Q}_{c^T}$ such that $\mathcal{Q}_{c^T}(M) = 1$ if $\mathcal{L}(M) = U_T$; otherwise, $\mathcal{Q}_{c^T}(M) = 0$. TL* assumes Teacher can answer membership queries for guarded words (instead of timed words) and give counterexamples in guarded words for candidate queries. This is not a strong assumption since there are data structures such as DBM [6] to represent time symbolically.

Algorithm 2 gives the pseudo-code of the TL* algorithm. The idea behind TL* is to first learn a DFA M accepting $Untime(U_T)$, the untimed language with respect to U_T, and then to refine the untimed language by adding time constraints. Therefore, TL* consists of two phases, namely the *untimed learning* phase (Lines 1-3) and the *timed refinement* phase (Lines 7-22). Note that the splitting of zones in Line 10 can be done by DBM subtraction [11].

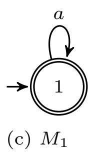
We use an example to illustrate the TL* algorithm. Suppose the timed language U_T to be learned is accepted by the ERA $\mathcal{A}_1$ as shown in Fig. 1 (a). In the untimed learning phase, L* is used to learn the DFA M_1, as shown in Fig. 1 (c), accepting the untimed language a^*, and the observation table (S, E, T) obtained by L* is shown in Fig. 1 (b). At this time, $\Sigma = \{a\}$, $S = \{\lambda\}$, and $E = \{\lambda\}$.



(a) ERA $\mathcal{A}_1$

	λ
λ	1 (s_0)
a	1

(b) T_1



(c) M_1

	λ
λ	1 (s_0)
$(a, true)$	1

(d) T_2

Fig. 1. Untimed Learning Phase

In the timed refinement phase, TL* first modifies the alphabet and the observation table into timed version, i.e., $\Sigma = \{(a, true)\}$, $S = \{(\lambda, true)\}$, and $E = \{(\lambda, true)\}$. The current timed observation table T_2 is shown in Fig. 1 (d). Then, TL* performs the timed candidate query for the first candidate ERA M_1. However, the answer to the candidate query is "no" with a negative counterexample $(a, x_a < 1) \in \mathcal{L}(M_1) \backslash \mathcal{L}(U_T)$. Because there is a prefix $(a, true)$ in the observation such that $[\![x_a < 1]\!] \subset [\![true]\!]$, the prefix $(a, true)$ is split into $(a, x_a < 1)$ and

Algorithm 2. TL* Algorithm

```
input  : $\Sigma$: alphabet, $C_\Sigma$: the set of event-recording clocks
output: a deterministic ERA accepting the unknown timed language $U_T$
1  Use $L^*$ to learn a DFA $M$ accepting $Untime(U_T)$ ;
2  Let $(S, E, T)$ be the observation table during the L* learning process ;
3  Replace $\alpha$ by $(\alpha, true)$, $s$ by $(s, true)$, and $e$ by $(e, true)$ for each $\alpha \in \Sigma$, $s \in S$
   and $e \in E$;
4  while true do
5      if $Q_c^T(M) = 1$ then return $M$ ;
6      else
7          Let $(a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ be the counterexample given by Teacher ;
8          foreach $(a_i, g_i)$, $i \in \{1, 2, \ldots, n\}$ do
9              if $(a_i, g)$ is a substring of $p$ or $e$ for some $p \in S \cup (S \cdot \Sigma)$ and $e \in E$
               such that $[\![g_i]\!] \subset [\![g]\!]$ then
10                 Let $G = \{\hat{g_1}, \hat{g_2}, \ldots, \hat{g_m}\}$ obtained by $[\![g]\!] - [\![g_i]\!]$ ;
11                 $\Sigma \longleftarrow \Sigma \setminus \{(a_i, g)\} \cup \{(a_i, g_i), (a_i, \hat{g_1}), (a_i, \hat{g_2}), \ldots, (a_i, \hat{g_m})\}$ ;
12                 Split $p$ into $\{\hat{p_0}, \hat{p_1}, \hat{p_2}, \ldots, \hat{p_m}\}$ where $(a_i, g_i)$ is a substring of $\hat{p_0}$
                   and $(a_i, \hat{g_j})$ is a substring of $\hat{p_j}$ for all $j \in \{1, 2, \ldots, m\}$ ;
13                 Split $e$ into $\{\hat{e_0}, \hat{e_1}, \hat{e_2}, \ldots, \hat{e_m}\}$ where $(a_i, g_i)$ is a substring of $\hat{e_0}$
                   and $(a_i, \hat{g_j})$ is a substring of $\hat{e_j}$ for all $j \in \{1, 2, \ldots, m\}$ ;
14                 Update $T$ by $Q_{m^T}(\hat{p_j} \cdot \hat{e_j})$ for all $j \in \{0, 1, 2, \ldots, m\}$ ;
15         while there exists $(s \cdot \alpha)$ such that $s \cdot \alpha \not\equiv s'$ for all $s' \in S$ do
16             $S \longleftarrow S \cup \{s \cdot \alpha\}$ ;
17             Update $T$ by $Q_{m^T}((s \cdot \alpha) \cdot \beta)$ for all $\beta \in \Sigma$ ;
18         $v \longleftarrow WS((a_1, g_1)(a_2, g_2) \cdots (a_n, g_n))$ ;
19         if $|v| > 0$ then
20             $E \longleftarrow E \cup \{v\}$ ;
21             Update $T$ by $Q_{m^T}(s \cdot v)$ and $Q_{m^T}(s \cdot \alpha \cdot v)$ for all $s \in S$ and $\alpha \in \Sigma$ ;
22         Construct candidate $M$ from $(S, E, T)$ ;
```

$(a, x_a \geq 1)$, and the timed membership queries for $(a, x_a < 1)$ and $(a, x_a \geq 1)$ are performed, respectively. The current observation table T_3 is shown in Fig. 2 (a). However, T_3 is not closed because there is $(a, x_a < 1)$ with no $s \in S$ such that $s \equiv (a, x_a < 1)$, so $(a, x_a < 1)$ is added into S and the membership queries for $(a, x_a < 1)(a, x_a < 1)$ and $(a, x_a < 1)(a, x_a \geq 1)$ are performed, respectively. The closed observation table T_4 and its the corresponding ERA M_2 are shown in Fig. 2 (b) and (c), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a \geq 1)\}$, $S = \{(\lambda, true), (a, x_a < 1)\}$, and $E = \{(\lambda, true)\}$.

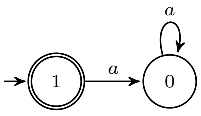
In the second iteration of the timed refinement phase, TL* performs the timed candidate query for M_2. However, the answer is still "no" with a positive counterexample $(a, x_a = 1) \in \mathcal{L}(U_T) \setminus \mathcal{L}(M_2)$. Because there are two prefixes $(a, x_a \geq 1)$ and $(a, x_a < 1)(x_a \geq 1)$ in the observation table (S, E, T) such that $[\![x_a = 1]\!] \subset [\![x_a \geq 1]\!]$, the prefix $(a, x_a \geq 1)$ is split into $(a, x_a = 1)$ and $(a, x_a > 1)$, and the prefix $(a, x_a < 1)(x_a \geq 1)$ is split into $(a, x_a < 1)(x_a = 1)$ and $(a, x_a < 1)(x_a > 1)$, respectively. The timed membership queries for the

	λ
λ	1 (s_0)
$(a, x_a < 1)$	0
$(a, x_a \geq 1)$	0

(a) T_3

	λ
λ	1 (s_0)
$(a, x_a < 1)$	0 (s_1)
$(a, x_a \geq 1)$	0
$(a, x_a < 1)(a, x_a < 1)$	0
$(a, x_a < 1)(a, x_a \geq 1)$	0

(b) T_4



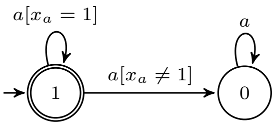
(c) M_2

Fig. 2. Timed Refinement 1

new prefixes are performed. The current closed observation table T_5 and its corresponding ERA M_3 are shown in Fig. 3 (a) and (b), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a > 1)\}$, $S = \{(\lambda, true), (a, x_a < 1)\}$, and $E = \{(\lambda, true)\}$.

	λ
λ	1 (s_0)
$(a, x_a < 1)$	0 (s_1)
$(a, x_a = 1)$	1
$(a, x_a > 1)$	0
$(a, x_a < 1)(a, x_a < 1)$	0
$(a, x_a < 1)(a, x_a = 1)$	0
$(a, x_a < 1)(a, x_a > 1)$	0

(a) T_5



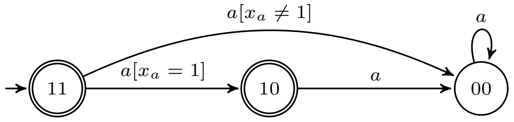
(b) M_3

Fig. 3. Timed Refinement 2

In the third iteration of the timed refinement phase, TL* performs the timed candidate query for the ERA M_3. However, the answers is still "no" with a negative counterexample $\pi = (a, x_a = 1)(a, x_a = 1) \in \mathcal{L}(M_3) \setminus \mathcal{L}(U_T)$. This time, no prefix or suffix in the observation table has to be split. TL* analyzes the counterexample as follows. $Q^0_{m^T}(\pi) = Q_{m^T}((a, x_a = 1)(a, x_a = 1)) = 0$. $Q^1_{m^T}(\pi) = Q^1_{m^T}([(a, x_a = 1)]_r(a, x_a = 1)) = Q_{m^T}((a, x_a = 1)) = 1 \neq Q^0_{m^T}(\pi)$. Thus, we have a witness suffix $v = (a, x_a = 1)$, and v is added into the set E. Then the membership queries for $s \cdot (a, x_a = 1)$ for all $s \in S$ are performed. The closed observation table T_7 and its corresponding ERA M_4 are shown in Fig. 4 (a) and (b), respectively. At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a > 1)\}$, $S = \{(\lambda, true), (a, x_a < 1), (a, x_a = 1)\}$, and $E = \{(\lambda, true), (a, x_a = 1)\}$.

	λ	$(a, x_a = 1)$
λ	1	1 (s_0)
$(a, x_a < 1)$	0	0 (s_1)
$(a, x_a = 1)$	1	0 (s_2)
$(a, x_a > 1)$	0	0
$(a, x_a < 1)(a, x_a < 1)$	0	0
$(a, x_a < 1)(a, x_a = 1)$	0	0
$(a, x_a < 1)(a, x_a > 1)$	0	0
$(a, x_a = 1)(a, x_a < 1)$	0	0
$(a, x_a = 1)(a, x_a = 1)$	0	0
$(a, x_a = 1)(a, x_a > 1)$	0	0

(b) T_7



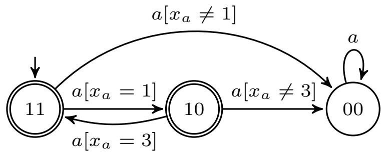
(c) M_4

Fig. 4. Timed Refinement 3

In the fourth iteration of the timed refinement phase, TL* performs the timed candidate query for the ERA M_4 again. However, the answer is still "no" with a positive counterexample $\pi = (a, x_a = 1)(a, x_a = 3) \in \mathcal{L}(U_T) \setminus \mathcal{L}(M_4)$. Three prefixes $(a, x_a > 1)$, $(a, x_a < 1)(a, x_a > 1)$, and $(a, x_a = 1)(a, x_a > 1)$ in the observation table T_7 have to be split, and the new split prefixes are shown in Fig. 5 (a). The timed membership queries for the new split prefixes concatenated with e for all $e \in E$ are performed. Then the TL* algorithm analyzes the counterexample. Since $Q^0_{m^T}(\pi) = Q^1_{m^T}(\pi) = Q^2_{m^T}(\pi)$, therefore there is no witness suffix for π. The closed observation table T_8 is shown in Fig. 5 (a), and it corresponding ERA M_5 is constructed as shown in Fig. 5 (b). At this time, $\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, 1 < x_a < 3), (a, x_a = 3), (a, x_a > 3)\}$, $E = \{(\lambda, true), (a, x_a < 1), (a, x_a = 1)\}$, and $E = \{(\lambda, true), (a, x_a = 1)\}$.

	λ	$(a, x_a = 1)$
λ	1	1 (s_0)
$(a, x_a < 1)$	0	0 (s_1)
$(a, x_a = 1)$	1	0 (s_2)
$(a, 1 < x_a < 3)$	0	0
$(a, x_a = 3)$	0	0
$(a, x_a > 3)$	0	0
$(a, x_a < 1)(a, x_a < 1)$	0	0
$(a, x_a < 1)(a, x_a = 1)$	0	0
$(a, x_a < 1)(a, 1 < x_a < 3)$	0	0
$(a, x_a < 1)(a, x_a = 3)$	0	0
$(a, x_a < 1)(a, x_a > 3)$	0	0
$(a, x_a = 1)(a, x_a < 1)$	0	0
$(a, x_a = 1)(a, x_a = 1)$	0	0
$(a, x_a = 1)(a, 1 < x_a < 3)$	0	0
$(a, x_a = 1)(a, x_a = 3)$	1	1
$(a, x_a = 1)(a, x_a > 3)$	0	0

(a) T_8



(b) M_5

Fig. 5. Timed Refinement 4

In the fifth iteration of the timed refinement, TL* performs the timed candidate query for M_5. This time, Teacher says that $\mathcal{L}(M_5) = U_T$, and the learning process of TL* is finished.

3.2 Analysis of the TL* Algorithm

Given a timed language U_T accepted by a deterministic ERA $\mathcal{A} = (\Sigma, L, l_0, \delta, L^f)$, TL* learns $Com(\mathcal{A})$ to accept U_T. In the learning process of TL*, each untimed word $(\alpha, true)$ for $\alpha \in \Sigma$ might be split into $|G_\mathcal{A}|$ timed words, where $G_\mathcal{A}$ is the set of clock zones partitioned by the clock guards appearing in $\mathcal{A}$. For example, the clock guards appearing in $\mathcal{A}_1$, as shown in Fig. 1 (a) p. 467, are $x_a = 1$ and $x_a = 3$, so $G_\mathcal{A} = \{x_a < 1, x_a = 1, 1 < x_a < 3, x_a = 3, x_a > 3\}$. Thus, each membership query of untimed word $(a, true)$ gives rise to $|G_\mathcal{A}|$ timed membership queries. Totally, TL* needs to perform $O(|\Sigma| \cdot |G_\mathcal{A}| \cdot |L|^2 + |L| \log |\pi|)$ membership queries to learn $Com(\mathcal{A})$, where π is the counterexample given by Teacher. By Theorem 1, TL* needs to perform $O(|L| + |\Sigma| \cdot |G_\mathcal{A}|)$ candidate queries.

Lemma 1. *Given a closed and consistent observation table (S, E, T), any deterministic ERA consistent with T must have at least $|S|$ locations.*

Proof. We first define a row in the observation table. If $p \in S \cup (S \cdot \Sigma)$ is a prefix (row) of the table, we use $row(p)$ to denote the function $f : E \mapsto \{0, 1\}$ defined by $f(e) = T(p \cdot e)$ for $e \in E$. Let $M = (\Sigma, L, l^0, \delta, L^f)$ be an ERA consistent with T. We then define $f'(s) = \delta(l^0, s)$ for every $s \in S$. For any two $s_1, s_2 \in S$, we have $row(s_1) \neq row(s_2)$ implying that there exists $e \in E$ such that $T(s_1 \cdot e) \neq T(s_2 \cdot e)$. Since M is consistent with T, exactly one of $\delta(l^0, s_1 \cdot e)$ and $\delta(l^0, s_2 \cdot e)$ is in L^f implying that $\delta(l^0, s_1)$ and $\delta(l^0, s_2)$ are distinct locations. Thus, $f'(s)$ takes on at least $|S|$ values implying that M has at lease $|S|$ locations.

Theorem 1. *TL* is correct and terminates in a finite number of iterations.*

Proof. The correctness is based on the fact that TL* returns an ERA only if it accepts the unknown timed language U_T. Let $\mathcal{A} = (\Sigma, L, l^0, \delta, L^f)$ be an ERA accepting U_T. In each iteration, TL* either adds a row into S in the observation table (S, E, T) or splits a clock guard of an event $\alpha \in \Sigma$ into at least two disjoint clock guards. Since the observation table should be consistent with $\mathcal{A}$ (otherwise, Teacher must have given wrong answers to membership queries), TL* adds at most $|L|$ rows into S. At last, each split clock guard will belong to $G_{\mathcal{A}}$. Thus, TL* terminates after $O(|L| + |\Sigma| \cdot |G_{\mathcal{A}}|)$ iterations.

Theorem 2. *The ERA learned by TL* has the minimal number of locations.*

Proof. Given a closed and consistent observation table (S, E, T), TL* constructs an ERA M exactly with $|S|$ locations. By Lemma 1, we can conclude that M has the minimal number of locations.

Comparison. Grinchtein et al.'s TL^*_{sg} uses region construction to actively guess all possible time constraints for an untimed word, so an original untimed membership query in L* gives rise to several membership queries of time words. The number of timed membership queries required by the TL^*_{sg} algorithm is $O(|\Sigma \times G_\Sigma| \cdot n^2 |\pi| \cdot |w| \binom{|\Sigma|+K}{|\Sigma|})$ where n is the number of locations of the learned ERA, π is the counterexample given by Teacher, w is the longest guarded word queried, and K is the largest constant appearing in the clock guards. We can observe that the number of timed membership queries required by TL^*_{sg} increases exponentially with the largest constant K and the size of the alphabet $|\Sigma|$. To learn the timed language accepted by $\mathcal{A}_1$, as shown in Fig. 1 (a) p. 467, TL^*_{sg} needs 34 timed membership queries, while our TL* only needs 16 timed membership queries. Note that our TL* algorithm is not affected by the largest constant K. If we change the guarded word $a[x_a = 3]$ in $\mathcal{A}_1$, as shown in Fig. 1 (a), into $a[x_a = 100]$, the number of membership queries required by our TL* algorithm is still 16, while that required by TL^*_{sg} increases exponentially.

4 Conclusion and Future Work

We proposed an efficient polynomial time algorithm, TL*, for learning ERAs. TL* can also be applied to other subclasses of timed automata, such as event-predicting automata [2], as they are determinizable. Our future work will implement TL* into the PAT model checker [12,14] such that PAT can automatically generate the assumptions for assume-guarantee reasoning for timed systems.

Acknowledgment. This work benefited from the discussions via e-mails with Olga Grinchtein, one of the authors of [7,8].

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. Theoretical Computer Science 211(1-2), 253–273 (1999)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
4. Clarke, E.M., Emerson, E.A.: Design and sythesis of synchronization skeletons using branching time temporal logic. In: Proceedings of the Logics of Programs Workshop, vol. 131, pp. 52–71 (1981)
5. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
6. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
7. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 379–395. Springer, Heidelberg (2004)
8. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theorectical Computer Science 411(47), 4029–4054 (2010)
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
10. Lin, S.W., Hsiung, P.A.: Counterexample-guided assume-guarantee synthesis through learning. IEEE Transactions on Computers 60(5), 734–750 (2011)
11. Lin, S.W., Hsiung, P.A., Huang, C.H., Chen, Y.R.: Model checking prioritized timed automata. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 370–384. Springer, Heidelberg (2005)
12. Liu, Y., Sun, J., Dong, J.S.: Analyzing hierarchical complex real-time systems. In: Proceedings of the 8th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), pp. 365–366. ACM, New York (2010)
13. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
14. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009)