An Efficient Algorithm for Learning Event-Recording Automata

Shang-Wei Lin¹, Étienne André¹, Jin Song Dong¹, Jun Sun², and Yang Liu²

¹ School of Computing, National University of Singapore
@example{linsw,andre,dongjs}@comp.nus.edu.sg
² Singapore University of Technology and Design
@example{sunjun,liuyang}@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 time constraints on the transitions of the DFA. We prove the correctness, termination, and minimality of the proposed $TL^*$ algorithm.

1 Introduction

In formal verification such as model checking [4, 11], system models and properties are assumed to be developed a priori during the verification process. In practice, however, modeling a system appropriately is not an easy task because if the model is too abstract, then 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, 12] proposed a learning algorithm, $L^*$, for inference of regular languages. Given an unknown language $U$ to be inferred, the $L^*$ algorithm 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 the unknown language $U$.

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 an automatic compositional synthesis framework, CAGS [9], based on the L* algorithm. Given a system modeled by a set of component models and a user given property, if the system does not satisfy the property, CAGS uses the L* algorithm to synthesize each component model such that the refined system satisfies the given property.

However, there were at most no extensions of the learning algorithm to inference timed regular languages until 2004, Grinchtein et al. [7] proposed a learning algorithm for event-recording automata [2] based on the L* algorithm. 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 to 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] and are sufficiently expressive to model many interesting timed systems. Because of its determinizability, a timed language accepted by an ERA can be classified into finite number of classes (each class can be represented by a location of an ERA). Thus we focus on learning timed languages accepted by ERA. Given a timed regular language UT accepted by ERA, TL* first learns a DFA M accepting U (the untimed version of UT) and then passively refines the time constraints on the transitions of M. 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 UT 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.

The rest of this paper is organized as follows: Section 2 gives some 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 $w_i = (a_1, t_1)(a_2, t_2)\ldots(a_n, t_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \ldots, n\}$ that are paired
with nonnegative real numbers $t_i \in \mathbb{R}^+$ such that the sequence $t = t_1 t_2 \ldots t_n$ of time-stamps is nondecreasing. Sometimes we denote the timed word $w_t$ by the pair $(\pi, T)$, where $\pi = a_1 a_2 \ldots a_n \in \Sigma^*$ is an untimed word over $\Sigma$. 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 $C_\Sigma$ to denote the set of event-recording clocks over $\Sigma$, i.e., $C_\Sigma = \{x_a \mid a \in \Sigma\}$.

A clocked word over $\Sigma$ is a finite sequence $w_c = (a_1, \gamma_1)(a_2, \gamma_2)\ldots(a_n, \gamma_n)$ of symbols $a_i \in \Sigma$ for $i \in \{1, 2, \ldots, n\}$ that are paired with clock valuations $\gamma_i$ 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 $1 < i \leq n$ and $a \neq a_{i-1}$. 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 $\gamma_i(x_a) = t_i$ 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 $j < k < i$. Sometimes we denote the clocked word $w_c$ by the pair $(\pi, \tau)$, where $\pi = a_1 a_2 \ldots a_n \in \Sigma^*$ is an untimed word over $\Sigma$ and $\tau = \gamma_1, \gamma_2, \ldots, \gamma_n$ is the sequence of clock valuations.

A clock constraint $\varphi$ is a conjunction of atomic constraints of the form $x_a \sim n$ for $x_a, x_b \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$, and $n \in \mathbb{N}$. A clock constraint $\varphi$ identifies a $|\Sigma|$-dimensional polyhedron $[\varphi] \subseteq (\mathbb{R}^+)^{|\Sigma|}$. A clock guard $g$ is a conjunction of atomic constraints of the form $x_a \sim n$ for $x_a \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$, and $n \in \mathbb{N}$. A clock guard $g$ identifies a $|\Sigma|$-dimensional hypercube $[g] \subseteq (\mathbb{R}^+)^{|\Sigma|}$. We use $G_\Sigma$ to denote the set of clock guards over $C_\Sigma$. 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 $i \in \{1, 2, \ldots, n\}$. Sometimes we denote the guarded word $w_g$ by the pair $(\pi, \overline{\tau})$, where $\pi = a_1 a_2 \ldots a_n \in \Sigma^*$ is an untimed word over $\Sigma$ and $\overline{\tau} = g_1 g_2 \ldots g_n$ is the sequence of clock guards.

**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 $\Sigma$, a finite set $L$ of locations, an initial location $l_0 \in L$, a set $L^f$ of accepting locations, and a transition function $\delta \subseteq L \times \Sigma \times G_\Sigma \rightarrow 2^L$. 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 $g_1, g_2 \in G_\Sigma$.

Note that in 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 deterministic by subset construction [2]. Fig. 1 (a) gives a deterministic ERA $A_1$ accepting the timed word $(a^*, \overline{T}) = (a, t_1)(a, t_2)(a, t_3)(a, t_4)\ldots = (a, 1)(a, 4)(a, 5)(a, 8)\ldots$, 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 $(a^*, \overline{\tau}) = (a, \gamma_1)(a, \gamma_2)(a, \gamma_3)(a, \gamma_4)\ldots$ to represent the timed word $(a^*, \overline{T})$ 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 $(a^*, \overline{\tau}) = (a, g_1)(a, g_2)(a, g_3)(a, g_4)\ldots$ to represent the timed word $(a^*, \overline{T})$ such that $g_{2i-1} = (x_a = 1)$ and $g_{2i} = (x_a = 3)$ for $i \in \mathbb{N}$. 
**Definition 2. (Complete Deterministic ERA).** A deterministic ERA $D = (\Sigma, L, l_0, \delta, L')$ is said to be complete if for all $l \in L$ and for all $a \in \Sigma$, $\delta(l, a, g_l)$ is defined for all $i \in \{1, 2, \ldots, n\}$ such that $[g_1] \cup [g_2] \cup \ldots \cup [g_n] = [true].$ 

If a deterministic ERA $D$ is not complete, we can always construct a complete deterministic ERA $Com(D) = (\Sigma, L \cup \{\tau\}, l_0, \delta', L')$ such that $L(D) = L(Com(D))$, where $\tau$ is an additional dead state and $\delta'$ is defined as follows: $\delta'(l, a, g) = \delta(l, a, g)$ if $\delta(l, a, g)$ is defined; otherwise, $\delta'(l, a, g) = \tau$. Fig. 1 (b) shows the complete deterministic ERA $Com(A_1)$ of $A_1$.

### 2.2 The L* Algorithm

The L* algorithm [3,12] is a formal method to learn a minimal DFA (with the minimal number of locations) that accepts an unknown language $U$ over an alphabet $\Sigma$. During the learning process, the L* algorithm interacts with a Minimal Adequate Teacher (Teacher for short) to make two types of queries: the membership and candidate queries. A membership query for a string $\sigma$ is a function $Q_m$ such that if $\sigma \in U$, then $Q_m(\sigma) = 1$; otherwise, $Q_m(\sigma) = 0$. A candidate query for a DFA $M$ is a function $Q_c$ such that if $L(M) = U$, then $Q_c(M) = 1$; otherwise, $Q_c(M) = 0$. During the learning process, the L* algorithm stores the membership query results 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 $T : (S \cup S \cdot \Sigma) \times E \rightarrow \{0, 1\}$ 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 $e \in E$. In the observation table, the L* algorithm categorizes strings based on Myhill-Nerode Congruence [8], as formulated in Definition 3.

**Definition 3. 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 $\sigma$ and $\sigma'$ are the representing strings of each other, denoted by $\sigma = [\sigma]_r$ and $\sigma' = [\sigma]_r$.

The L* algorithm 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 $\alpha \in \Sigma$. If the observation 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...
\[\Sigma_C = \Sigma, L_C = S, b^0_C = \{\lambda\}, \delta_C(s, \alpha) = [s \cdot \alpha], \text{ for } s \in S \text{ and } \alpha \in \Sigma, \text{ and } L_C^f = \{s \in S \mid T(s, \lambda) = 1\}.\]

Subsequently, the \(L^*\) algorithm will make a candidate query for \(C\).

If \(Q_C(M) = 0\), i.e., \(L(C) \neq U\), then Teacher will give a counterexample \(\sigma_{ce}\). The counterexample \(\sigma_{ce}\) is positive if \(\sigma_{ce} \in L(U) \setminus L(C)\); negative if \(\sigma_{ce} \in L(C) \setminus L(U)\). The \(L^*\) algorithm will analyze the counterexample \(\sigma_{ce}\) to find the witness suffix. For two strings that are classified by \(L^*\) into an equivalence class, a witness suffix is a string that when appended to the 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 \(\sigma_{ce}\) given by Teacher, we define an \(i\)-decomposition query of \(\sigma_{ce}\), denoted by \(Q_m^i(\sigma_{ce})\), as follows: \(Q_m^i(\sigma_{ce}) = Q_m([u_i]_r \cdot v_i)\) where \(\sigma_{ce} = u_i \cdot v_i\) is a decomposition of \(\sigma_{ce}\) such that \([u_i]_r = i\), and \([u_i]_r\) is the representing string of \(u_i\) in \(S\). The witness suffix of \(\sigma_{ce}\), denoted by \(WS(\sigma_{ce})\), is the suffix \(v_i\) of the decomposition of \(\sigma_{ce}\) such that \(Q_m_i(\sigma_{ce}) \neq Q_m^0(\sigma_{ce})\). Once the witness suffix \(WS(\sigma_{ce})\) is obtained, \(L^*\) uses \(WS(\sigma_{ce})\) to refine the candidate DFA \(C\) until \(L(C) = L(U)\). The pseudo-code of the \(L^*\) algorithm is given in Algorithm 1.

**Algorithm 1: \(L^*\) Algorithm**

\[\begin{align*}
\text{input} & : \Sigma; \text{ alphabet} \\
\text{output} & : \text{a DFA accepting the unknown language } U \\
1 & \text{Let } S = E = \{\lambda\}; \\
2 & \text{Update } T \text{ by } Q_m(\lambda) \text{ and } Q_m(\lambda \cdot \alpha) \text{, for all } \alpha \in \Sigma; \\
3 & \text{while } \text{true} \text{ do} \\
4 & \hspace{1em} \text{while there exists } (s \cdot \alpha) \text{ such that } (s \cdot \alpha) \neq s' \text{ for all } s' \in S \text{ do} \\
5 & \hspace{2em} S \leftarrow S \cup \{s \cdot \alpha\}; \\
6 & \hspace{2em} \text{Update } T \text{ by } Q_m((s \cdot \alpha) \cdot \beta) \text{, for all } \beta \in \Sigma; \\
7 & \hspace{1em} \text{Construct candidate DFA } M \text{ from } (S, E, T); \\
8 & \hspace{1em} \text{if } Q_C(M) = 1 \text{ then return } M; \\
9 & \hspace{1em} \text{else} \\
10 & \hspace{2em} \sigma_{ce} \leftarrow \text{the counterexample given by Teacher}; \\
11 & \hspace{2em} v \leftarrow WS(\sigma_{ce}); \\
12 & \hspace{2em} E \leftarrow E \cup \{v\}; \\
13 & \hspace{2em} \text{Update } T \text{ by } Q_m(s \cdot v) \text{ and } Q_m(s \cdot \alpha \cdot v) \text{, for all } s \in S \text{ and } \alpha \in \Sigma; \\
\end{align*}\]

We use an example to illustrate how the \(L^*\) algorithm works to learn a minimal DFA accepting an unknown language. Suppose the unknown language \(U = (a|b|c) \cdot a^*\) over \(\Sigma = \{a, b, c\}\) needs to be learned. Initially, \(S\) and \(E\) are initialized to \(\{\lambda\}\) and then the membership queries of \(\lambda, a, b,\) and \(c\) are performed. At this point, the observation table with \(S = \{\lambda\}, E = \{\lambda\}\) is shown in Fig. 2 (a). The observation table now is not closed because there is not any \(s \in S\) such that \(a \equiv s\). So, \(a\) is added into \(S\), and then the membership queries of \(aa, ab,\) and \(ac\) are performed respectively. At this point, the observation table with \(S = \{\lambda, a\},\)
$E = \{ \lambda \}$ is closed as shown in Fig. 2 (b). The corresponding DFA $M_1$ is shown in Fig. 2 (c). The candidate query of $M_1$ is performed.

However, Teacher gives a negative counterexample $abc$ that is accepted by $M_1$ but not in $U$. The $L^*$ algorithm analyzes the negative counterexample $abc$ to get the witness suffix as follows: $Q^0_{m}(abc) = 0$, $Q^1_{m}(abc) = Q_m([a] \cdot bc) = Q_m(abc) = 0$, $Q^2_{m}(abc) = Q_m([ab] \cdot c) = Q_m(\lambda \cdot c) = Q_m(c) = 1 \neq Q^0_{m}(abc)$. After analyzing the counterexample $abc$, the witness suffix is $c$. So, $c$ is added into $E$, and the membership queries of $c$, $ac$, $bc$, $cc$, $aac$, $abc$, and $acc$ are performed. The observation table now with $S = \{ \lambda, a \}$, $E = \{ \lambda, c \}$ is shown in Fig. 3 (a). However, the observation table is not closed because there is no $s \in S$ such that $ab \equiv s$. So, $ab$ is added into $S$, and then the membership queries of $aba$, $abb$, $abc$, and $abcc$ are performed. At this point, the observation table with $S = \{ \lambda, a, ab \}$, $E = \{ \lambda, c \}$ is closed as shown in Fig. 3 (b). The corresponding DFA $M_2$ is shown in Fig. 3 (c) and $L(M_2) = U$.

Assume $\Sigma$ is the alphabet of the unknown regular language $U$ and the number of states 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. Angluin [3] proved that as long as the unknown language $U$ is regular, the $L^*$ algorithm will learn a complete minimal DFA $M$ such that $L(M) = U$ in at most $n - 1$ iterations.
3 An Efficient Algorithm for Learning Event-Recording Automata

The intuition behind the L* algorithm is to classify untimed words into 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, therefore a timed language accepted by an ERA can also be classified into finite number of classes. The proposed TL* algorithm also tries to find the finite number of classes (locations). Section 3.1 introduces the TL* algorithm, and its analysis is given in Section 3.2.

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 $Q_{m\text{r}}^T$ such that $Q_{m\text{r}}^T(w_g) = 1$ if $w_g \in U_T$; otherwise $Q_{m\text{r}}^T(w_g) = 0$. A timed candidate query for an ERA $M$ is a function $Q_{c\text{r}}^T$ such that $Q_{c\text{r}}^T(M) = 1$ if $L(M) = U_T$; otherwise, $Q_{c\text{r}}^T(M) = 0$.

The idea behind the TL* algorithm is to first learn a DFA $M$ accepting Untime($U_T$), the untimed language with respect to $U_T$, and then to refine the time constraints on the transitions of $M$. Therefore, TL* consists of two phases, namely the untimed learning phase and the timed refinement phase. Algorithm 2 gives the pseudo-code of TL*. The details of TL* are described as follows.

**Untimed Learning.** In this phase, the L* algorithm is used to learn a DFA $M$ accepting Untime($U_T$), the untimed language with respect to $U_T$, and then to refine the time constraints on the transitions of $M$. Therefore, TL* consists of two phases, namely the untimed learning phase and the timed refinement phase. Algorithm 2 gives the pseudo-code of TL*. The details of TL* are described as follows.

**Timed Refinement.** In this phase, the TL* algorithm tries to refine the time constraints on the transitions of the DFA $M$ learned in the untimed learning phase. It performs the following steps:

1. Modify the untimed alphabet into timed alphabet, i.e., replace $\alpha$ by $(\alpha, true)$ for each $\alpha \in \Sigma$. Modify all untimed prefixes (rows) and suffixes (columns) in the observation table $(S, E, T)$ into timed versions, i.e., replace $s$ by $(s, true)$, $s \cdot \alpha$ by $(s, true)(\alpha, true)$, and $e$ by $(e, true)$ for each $s \in S$, $\alpha \in \Sigma$, and $e \in E$, respectively. (Line 3)

2. Perform the candidate query for the ERA $M$. If the answer is “yes”, then the ERA $M$ accepts the language $U_T$ to be learned, and $M$ is returned. (Line 5)

3. If the answer to the candidate query for $M$ is “no” with a counterexample $(a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ given by Teacher, TL* will split prefixes (rows) and suffixes (columns) in the observation table as follows. If a prefix $p \in S \cup (S \cdot \Sigma)$ or a suffix $e \in E$ in the observation table has a substring of the form $(a_i, g)$ for some $i \in \{1, 2, \ldots, n\}$ and $[g] \subset [g]$, then $[g]$ is partitioned using $g_i$ such that $[g] = [g_i] \cup G$ where $G = \{g_1, g_2, \ldots, g_n\}$ is obtained by $[g] - [g_i]$ using DBM subtraction [10]. The prefix $p$ is split into $\{p_0, p_1, p_2, \ldots, p_m\}$.
Algorithm 2: TL* Algorithm

input: : \( \Sigma \): alphabet, \( C_E \): 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 \( U_{time}(U_T) \);
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
   if \( Q^2_t(M) = 1 \) then return \( M \);
   else
      foreach \( (a_i, g_i) \), \( i \in \{1, 2, \ldots, n\} \) do
         if \((a_i, g_i)\) is a substring of \( p \) or \( e \) for some \( p \in S \cup (S \cdot \Sigma) \) and \( e \in E \) such that \([g_i] \in [g] \) then
            Let \( G = \{g_i \}, g_i, \ldots, g_m \} \) obtained by \([g] - [g] \);
            \( \Sigma = \Sigma \cdot \{g_i \}, (a_i, g_i), \ldots, (a_i, g_m), \} \);
            Split \( p \) into \( \{p_0, p_1, p_2, \ldots, p_m \} \) where \( (a_i, g_i) \) is a substring of \( p_0 \) and \( (a_i, g_j) \) is a substring of \( p_j \) for all \( j \in \{1, 2, \ldots, m\} \);
            Split \( e \) into \( \{e_0, e_1, e_2, \ldots, e_m \} \) where \( (a_i, g_i) \) is a substring of \( e_0 \) and \( (a_i, g_j) \) is a substring of \( e_j \) for all \( j \in \{1, 2, \ldots, m\} \);
            Update \( T \) by \( Q_m^r(\hat{p}_j \cdot \hat{e}_j) \) for all \( j \in \{0, 1, 2, \ldots, m\} \);
         while there exists \( (s \cdot \alpha) \) such that \( s \cdot \alpha \neq s' \) for all \( s' \in S \) do
            \( S = S \cup \{s \cdot \alpha\} \);
            Update \( T \) by \( Q_m^r((s \cdot \alpha) \cdot \beta) \) for all \( \beta \in \Sigma \);
            \( v \leftarrow WS((a_1, g_1), (a_2, g_2), \ldots, (a_n, g_n)) \);
            if \(|v| > 0 \) then
               \( E = E \cup \{v\} \);
            Update \( T \) by \( Q_m^r(s \cdot v) \) and \( Q_m^r(s \cdot \alpha \cdot v) \) for all \( s \in S \) and \( \alpha \in \Sigma \);
         Construct candidate \( M \) from \((S, E, T)\);
      end
   end
end

where \((a_i, g_i)\) is a substring of \( p_0 \) and \((a_i, g_j)\) is a substring of \( p_j \) for all \( j \in \{1, 2, \ldots, m\} \). Similarly, the suffix \( e \) is also split into \( \{e_0, e_1, e_2, \ldots, e_m \} \) where \((a_i, g_i)\) is a substring of \( e_0 \) and \((a_i, g_j)\) is a substring of \( e_j \) for all \( j \in \{1, 2, \ldots, m\} \).

4. Check whether the observation table \((S, E, T)\) is closed. If there is a prefix \( s \cdot \alpha \) with no \( s' \in \Sigma \) such that \( s \cdot \alpha \equiv s' \) for some \( s \in S \) and \( \alpha \in \Sigma \), the observation table is not closed, and \( s \cdot \alpha \) is added into the set of prefixes \( S \). Then the observation table is updated by performing the timed membership queries \( Q_m^r(\hat{p}_j \cdot \hat{e}_j) \) for all \( j \in \{0, 1, 2, \ldots, m\} \). (Lines 7-14)

5. Analyze the counterexample to find the witness suffix \( v \). For a counterexample \( \pi \) given by Teacher and the observation table \((S, E, T)\), we also define an \( i \)-decomposition query of the guarded word \( \pi \), denoted by \( Q_m^r(i)(\pi) \), as
follows: \( Q^i_{m,r}(\pi) = Q^m_{\pi^r}(u_i \cdot v_i) \) where \( \pi = u_i \cdot v_i \) is a decomposition of \( \pi \) such that \(|u_i| = i \) and \(|u_i|_r \) is the representing string of \( u_i \) in \( S \). The witness suffix of \( \pi \), denoted by \( WS(\pi) \), is the suffix \( v_i \) of the decomposition of \( \pi \) such that \( Q^i_{m,r}(\pi) \neq Q^0_{m,r}(\pi) \). If there is a witness suffix \( v_i \), i.e., \(|v_i| > 0 \), then \( v_i \) is added into the set of suffixes \( E \), and the observation table is updated by performing the timed membership queries \( Q^r_{m,r}(s \cdot v_i) \) and \( Q^r_{m,r}(s \cdot \alpha \cdot v_i) \) for each \( s \in S \) and \( \alpha \in \Sigma \). (Lines 18-21)

6. Construct the ERA \( M = (\Sigma_M, L_M, \rho^0_M, \delta_M, L^f_M) \) from the observation table \((S, E, T)\) such that \( \Sigma_M = \Sigma, L_M = S, \rho^0_M = \{ \lambda \}, \delta_M(s, a) = [s \cdot a]_r \) for \( s \in S \) and \( a \in \Sigma \), and \( L^f_M = \{ s \in S : T(s, \lambda) = 1 \} \). Goto Step 3. (Line 22)

We use an example to illustrate the TL* algorithm. Suppose the timed language \( U_T \) to be learned is accepted by the ERA \( A_1 \) as shown in Fig. 1 (a). In the untimed learning phase, the L* algorithm is used to learn the DFA \( M_1 \), as shown in Fig. 4 (c), accepting the untimed language \( a^* \), and the observation table \((S, E, T)\) obtained by \( L^* \) is shown in Fig. 4 (a). At this time, \( \Sigma = \{ a \}, S = \{ \lambda \}, \) and \( E = \{ \lambda \} \).

\[
\begin{array}{c|c|c}
\lambda & a & 1 \\
---&---&---
\end{array}
\begin{array}{c|c|c}
\lambda & a \cdot true & 1 \\
---&---&---
\end{array}
\begin{array}{c}
\lambda \\
---
\end{array}
\]
\( a 
\)
\( 1 
\)

Fig. 4. Untimed Learning Phase

In the timed refinement phase, TL* first modifies the alphabet and the observation table into timed version. At this time, \( \Sigma = \{(a, true)\}, S = \{ (\lambda, true) \}, \) and \( E = \{ (\lambda, true) \} \). The current timed observation table \( T_2 \) is shown in Fig. 4 (b). Then, TL* performs the timed candidate query for the first candidate ERA \( M_1 \). However, the answer to the candidate query is “no”, and Teacher gives a negative counterexample \((a, x_a < 1) \in L(M_1) \setminus L(U_T) \). 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 \((a, x_a \geq 1) \), 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. 5 (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 \( T_4 \) is shown in Fig. 5 (b), and the corresponding ERA \( M_2 \) is constructed as shown in Fig. 5 (c). 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 \((a, x_a = 1) \in L(U_T) \setminus L(M_2) \). Because there are two prefixes \((a, x_a \geq 1) \) and \((a, x_a < 1) \) in the observation table \((S, E, T)\) such
\[ \lambda \quad \lambda \]

<table>
<thead>
<tr>
<th></th>
<th>( \lambda )</th>
<th>( \lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a, x_a &lt; 1 )</td>
<td>0</td>
<td>1 (s_0)</td>
</tr>
<tr>
<td>( a, x_a = 1 )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( a, x_a &gt; 1 )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( (a, x_a &lt; 1) )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( (a, x_a &gt; 1) )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( (a, x_a &lt; 1)(a, x_a &lt; 1) )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( (a, x_a &gt; 1)(a, x_a &gt; 1) )</td>
<td>0</td>
<td>0 (s_1)</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>( a[x_a = 1] )</td>
<td>( a[x_a \neq 1] )</td>
</tr>
</tbody>
</table>

Fig. 5. Timed Refinement 1

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 new prefixes are performed. The current closed observation table \( T_3 \) is shown in Fig. 6 (a), and the corresponding ERA \( M_3 \) is shown in Fig. 6 (b). At this time, \( \Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a > 1)\} \), \( S = \{\{\lambda, true\}, (a, x_a < 1)\} \), and \( E = \{\{\lambda, true\}\} \).

\[
\begin{array}{ccc}
\lambda & \lambda & \lambda \\
(\lambda, 1) & (\lambda, 0) & (\lambda, 0) \\
(a, x_a < 1) & (a, x_a = 1) & (a, x_a > 1) \\
\end{array}
\]

Fig. 6. Timed Refinement 2

In the third iteration of the timed refinement phase, TL$^*$ performs the timed candidate query for the ERA \( M_4 \). However, the answer 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. \( Q^0_{m,T}(\pi) = Q^0_{m,T}((a, x_a = 1)(a, x_a = 1)) = 0 \). \( Q^1_{m,T}(\pi) = Q^1_{m,T}((a, x_a = 1)(a, x_a = 1)) = Q^1_{m,T}((a, x_a = 1)) = 1 \neq Q^0_{m,T}(\pi) \). Thus, we have a witness suffix \( v = (a, x_a = 1) \), and \( v \) is added into the set of suffixes \( E \). Then the timed membership queries for \( s \cdot (a, x_a = 1) \) for all \( s \in S \) are performed, and the current observation table \( T_6 \) is shown in Fig. 7 (a). However, the observation table \( T_6 \) is not closed because there is a prefix \((a, x_a = 1)\) with no \( s \in S \) such that \( s \equiv (a, x_a = 1) \), so \((a, x_a = 1)\) is added into the set of prefixes \( S \) and the timed membership queries for \((a, x_a = 1) \cdot \alpha \) for all \( \alpha \in \Sigma \) are performed. The closed observation table \( T_7 \) is shown in Fig. 7 (b), and the corresponding ERA \( M_4 \) is shown in Fig. 7 (c). 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 \( E = \{\{\lambda, true\}, (a, x_a = 1)\} \).

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 \( \pi = (a, x_a = 1)(a, x_a = 3) \in \mathcal{L}(U_T) \setminus \mathcal{L}(M_4) \).
Because there are 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 such that \([x_a = 3]\) \(\subseteq [x_a > 1]\), the prefix \((a, x_a > 1)\) is split into three prefixes \((a, x_a < 1)(a, x_a < 3), (a, x_a = 3),\) and \((a, x_a > 3);\) the prefix \((a, x_a < 1)(a, x_a > 1)\) is split into three prefixes \((a, x_a < 1)(a, x_a < 3), (a, x_a < 1)(a, x_a = 3),\) and \((a, x_a < 1)(a, x_a > 3);\) the prefix \((a, x_a = 1)(a, x_a > 1)\) is also split into three prefixes \((a, x_a = 1)(a, x_a < 3), (a, x_a = 1)(a, x_a = 3),\) and \((a, x_a = 1)(a, x_a > 3)\). 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 \(Q^1_{m,T}(\pi) = Q^1_{m,T}(\pi) = Q^2_{m,T}(\pi),\) therefore there is no witness suffix for \(\pi\).

The closed observation table \(T_5\) is shown in Fig. 8 (a), and it corresponding ERA \(M_5\) is constructed as shown in Fig. 8 (b). At this time, \(\Sigma = \{(a, x_a < 1), (a, x_a = 1), (a, x_a < 3), (a, x_a = 3), (a, x_a > 3)\}, E = \{(\lambda, true), (a, x_a < 1), (a, x_a = 1)\},\) and \(E = \{(\lambda, true), (a, x_a = 1)\}\).

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

### 3.2 Analysis of the TL* Algorithm

The time complexity of TL* is analyzed as follows. Given a timed language \(U_T\) accepted by a deterministic ERA \(A = (\Sigma, L, l_0, \delta, L')\), the TL* algorithm learns \(Com(A)\) to accept \(U_T\). In the learning process of TL*, each untimed word \((\alpha, true)\) for \(\alpha \in \Sigma\) might be split into \(|G_A|\) timed words, where \(G_A\) is the set of clock zones partitioned by the clock guards appearing in \(A\). For example, the clock guards appearing in \(A_1\), as shown in Fig. 1 (a), are \(x_a = 1\) and \(x_a = 3\), so \(G_A = \{x_a < 1, x_a = 1, 1 < x_a < 3, x_a = 3, x_a > 3\}\). Thus, each membership query of untimed word \((a, true)\) gives rise to \(|G_A|\) timed membership queries. Totally, TL* needs to perform \(O(|\Sigma| \cdot |G_A| \cdot |L|^2 + |L| \log |\pi|)\) membership
queries to learn $\text{Com}(A)$, where $\pi$ is the counterexamples given by Teacher. After a candidate query of a candidate ERA $M$, TL$^*$ either adds a location into $M$ or splits a clock guard on a transition of $M$ into at least two clock guards. Thus, TL$^*$ needs to perform $O(|L| + |\Sigma| \cdot |G_A|)$ to learn $\text{Com}(A)$ accepting $U_T$.

Theorem 1 proves the correctness of the TL$^*$ algorithm, and the termination of the TL$^*$ algorithm is proved in Theorem 2.

**Theorem 1.** The TL$^*$ algorithm is correct.

**Proof.** Let $U_T$ be the timed language and $M$ be the ERA learned by TL$^*$. We want to prove that $\mathcal{L}(M) = U_T$ holds, i.e., $\mathcal{L}(M) \subseteq U_T$ and $U_T \subseteq \mathcal{L}(M)$. Let’s prove $\mathcal{L}(M) \subseteq U_T$ by contradiction. Assume $\mathcal{L}(M) \setminus U_T \neq \emptyset$, which implies $Q_{c,T}(M) = 0$. However, this contradicts to the fact that $M$ is returned by TL$^*$ because $Q_{c,T}(M) = 1$. Let’s prove $U_T \subseteq \mathcal{L}(M)$ also by contradiction. Assume $U_T \setminus \mathcal{L}(M) \neq \emptyset$, which implies $Q_{c,T}(M) = 0$. However, this also contradicts to the fact that $M$ is returned by TL$^*$ because $Q_{c,T}(M) = 1$. \hfill $\Box$

**Theorem 2.** The TL$^*$ algorithm terminates.

**Proof.** Let $U_T$ be the unknown timed language. Assume that $U_T$ is accepted by a hypothetic deterministic ERA $A = (\Sigma, L, \delta, L_f)$. In the learning process, TL$^*$ will constructively modify the observation table such that the final observation table is consistent with $\text{Com}(A)$. After each timed candidate query of a candidate ERA $M$, TL$^*$ either adds a location into $M$ or splits a guarded word $(a,g)$ on a transition of $M$ into several guarded words $(a,g_1), (a,g_2), \ldots, (a,g_n)$ such that $[g] = [g_1] \cup [g_2] \cup \ldots \cup [g_n]$. Let $G_A$ be the set of clock zones partitioned by the clock guards appearing in $A$. At last, each split clock guard $g_i$ will belongs to $G_A$ for all $i \in \{1, 2, \ldots, n\}$. Since both $|L|$ and $|G_A|$ are finite, the TL$^*$ algorithm will terminate after $O(|L| + |\Sigma| \cdot |G_A|)$ iterations. \hfill $\Box$

Theorem 3 proves the minimality of the TL$^*$ algorithm, i.e., given an unknown timed language $U_T$, the number of locations of the ERA $M$ learned by the TL$^*$ algorithm is minimum among all ERAs $M'$ such that $\mathcal{L}(M) = U_T$. 

---

**Fig. 8.** Timed Refinement 4
\textbf{Theorem 3.} Assume the observation table $(S, E, T)$ is closed and consistent and $M = (\Sigma, L, l^0, \delta, L^f)$ is the ERA constructed from the observation table $(S, E, T)$. If $M' = (\Sigma, L', l^{0'}, \delta', L'^{f})$ is any other ERA consistent with $T$, then $M'$ has at least $|L|$ locations.

\textit{Proof.} Before we prove this theorem, let us first formally define the membership query results of a row in the observation table. If $p \in S \cup (S \cdot \Sigma)$ is a prefix (row) of the observation table, we use $\text{row}(p)$ to denote the finite function $f : E \mapsto \{0,1\}$ defined by $f(e) = T(p \cdot e)$ for $e \in E$. Because $M'$ is consistent with $T$, therefore for each $p \in S \cup (S \cdot \Sigma)$ and for each $e \in E$, $\delta'(l^{0'}, p \cdot e) \in L'^f$ iff $T(p \cdot e) = 1$, which means that $\delta'(l^{0'}, p, e) \in L'^f$ iff $T(p, e) = 1$, so $\text{row}(\delta'(l^{0'}, p))$ is equal to $\text{row}(p)$. Since $p$ ranges over $L$ and $\text{row}(\delta'(l^{0'}, p))$ ranges over $L'$, so $|L'| \geq |L|$, i.e., $M'$ must have at least $|L|$ locations. \hfill \Box

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
$\lambda$ & $\lambda(a, x_a = 3)$ & $\lambda(a, x_a = 1)$ \\
\hline
(a, x_a = 0) & (a, x_a = 0, x_a = 0) & 0 0 (s_0) \\
(a, x_a = 1) & (a, x_a = 1, x_a = 0) & 1 1 (s_1) \\
(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a \geq 3) & 0 0 & 0 0 \\
(a, 0 \leq x_a \leq 1) & 0 0 & 0 0 \\
(a, 1 \leq x_a \leq 2) & 0 0 & 0 0 \\
(a, 2 \leq x_a \leq 3) & 0 0 & 0 0 \\
(a, 0 \leq x_a \leq 2) & 0 0 & 0 0 \\
(a, 1 \leq x_a \leq 3) & 0 0 & 0 0 \\
(a, 0 \leq x_a \leq 3) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 0)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 0) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 1) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 3) & 0 0 & 0 0 \\
(a, x_a = 1)(a, x_a = 2) & 0 0 & 0 0 \\
\hline
\end{tabular}
\end{center}

\textbf{Fig. 9.} Grinchtein’s TL$_{sg}^*$ Algorithm

\textbf{Comparison.} Since Grinchtein et al. [6, 7] did not implement the TL$_{sg}^*$ algorithm, to be fair, let us compare our TL$^*$ algorithm with TL$_{sg}^*$ from a theoretical point of view. 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 \( O(|\Sigma \times G| \cdot n^2|\pi| \cdot |w|(|\Sigma|+|K|)) \) where \( n \) is the number of locations of the learned ERA, \( \pi \) 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 Grinchtein’s TL\( ^*_{sg} \) algorithm increases exponentially to the largest constant \( K \) and the number of alphabet \(|\Sigma|\). To learn the timed language accepted by \( A_1 \), as shown in Fig. 1 (a), Grinchtein’s TL\( ^*_{sg} \) algorithm needs 34 timed membership queries, while our TL\( ^* \) only needs 16 timed membership queries. Fig. 9 (a) shows the observation table obtained by Grinchtein’s TL\( ^*_{sg} \) algorithm, and the final learned ERA is shown in Fig. 9 (b). Note that our TL\( ^* \) algorithm is not affected by the largest constant \( K \) appearing in the clock guards. If we change the guarded word \( a[x_a=3] \) in \( A_1 \), 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 Grinchtein’s TL\( ^*_{sg} \) algorithm increases exponentially. Further, Grinchtein’s TL\( ^*_{sg} \) algorithm cannot guarantee the number of locations of the learned ERA is minimal, while our TL\( ^* \) algorithm can. Fig. 10 (a) gives another unknown timed language \( U'_T \) accepted by \( A_2 \). Our TL\( ^* \) algorithm learns \( \text{Com}(A_2) \) to accept \( U'_T \), whose number of locations is minimal as shown in Fig. 10 (b), while the ERA \( M' \) learned by Grinchtein’s TL\( ^*_{sg} \) algorithm is shown in Fig. 10 (c), whose number of locations is not minimal.

![Diagram](image-url)

**Fig. 10.** Difference between TL\( ^* \) and TL\( ^*_{sg} \)
4 Conclusion and Future Work

We proposed an efficient polynomial time algorithm, TL*, for learning event-recording automata. The TL* algorithm can also be applied to learn other determinizable subclasses of timed automata, such as event-predicting automata (EPA) [2], as long as the timed Teacher answers the timed membership and timed candidate queries correctly. Our future work will implement the TL* algorithm into the PAT model checker [13] 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, the author of [6, 7].

References