



Learning Deterministic Multi-Clock Timed Automata

Yu Teng

School of Software Engineering,
Tongji University
Shanghai, China
tengyu@tongji.edu.cn

Miaomiao Zhang

School of Software Engineering,
Tongji University
Shanghai, China
miaomiao@tongji.edu.cn

Jie An

National Institute of Informatics
Tokyo, Japan
jieran@nii.ac.jp

ABSTRACT

We present an algorithm for active learning of deterministic timed automata with multiple clocks. The algorithm is within the querying framework of Angluin's L^* algorithm and follows the idea proposed in existing work on the active learning of deterministic one-clock timed automata. We introduce an equivalence relation over the reset-clocked language of a timed automaton and then transform the learning problem into learning the corresponding reset-clocked language of the target automaton. Since a reset-clocked language includes the clocks reset information which is not observable, we first present the approach of learning from a powerful teacher who can provide reset information by answering reset information queries from the learner. Then we extend the algorithm in a normal teacher situation in which the learner can only ask standard membership query and equivalence query while the learner guesses the reset information. We prove that the learning algorithm terminates and returns a correct deterministic timed automaton. Due to the need of guessing whether the clocks reset at the transitions, the algorithm is of exponential complexity in the size of the target automaton.

CCS CONCEPTS

• **Theory of computation** → **Regular languages**; **Active learning**; • **Computer systems organization** → **Real-time languages**.

KEYWORDS

automata learning, active learning, deterministic multi-clock timed automata, timed language, reset-clocked language

ACM Reference Format:

Yu Teng , Miaomiao Zhang , and Jie An . 2024. Learning Deterministic Multi-Clock Timed Automata. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24)*, May 14–16, 2024, Hong Kong SAR, China. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3641513.3650124>

1 INTRODUCTION

Automata learning (a.k.a. model learning) is a classic topic with an extensive history, as a research intersection of formal methods

(e.g. automata theory [18]) and machine learning (e.g. computation learning theory [20]). In general, according to the choices of the data set, it can be divided into two types. One is named *passive learning*, in which the learner has no control of the available data set. In [8, 14], Gold and Angluin prove that the passive learning problem of finding the smallest finite-state automaton consistent with a given set of accepted words and rejected words is NP-complete. The other is named *active learning*, in which a mechanism is designed for the learner to select the data set. In [9], Angluin proposes the famous L^* algorithm and proves that the complexity of the learning problem is polynomial under the active learning mechanism such that the learner is allowed to ask membership queries and equivalence queries for a teacher. In a membership query, the learner can ask whether a given word belongs to the target regular language. In an equivalence query, the learner provides a hypothesis automaton and asks whether it recognizes the target regular language. The learning process will continue until returning the exact model. Following Angluin's approach, many efficient algorithms and tools have been proposed for active learning of different kinds of automata. It has found a wide range of applications in model checking [13, 21, 23], compositional verification [4], legacy code analysis [24], etc. We refer to the surveys [26, 30] for a comprehensive introduction.

In recent decades, timed systems have received extensive attention. As we know, timing constraints play a key role in the correctness and safety of timed systems. Since finite-state automata are unable to describe the timing behaviors, this yields a fundamental difference and makes learning formal models of timed systems a challenging yet interesting problem.

In general, for timed systems, automata learning requires learning a timed model from either passive observations or active tests of the system, consisting of a collection of time-event sequences. The learned model should describe these timing behaviors correctly. Timed automata [2], extending finite-state automata with clock variables, is one of the most popular formal models of timed systems. However, there are several obstacles to active learning of timed automata. Since the transitions of timed automata contain both timing constraints that test the values of clocks and resets that update the clocks, from the view of constructing a timed automaton, we need to determine (i) the number of clocks, (ii) the reset information, and (iii) the timing constraints, none of which are directly observable from time-event sequences. The existing works consider timed automata with different restrictions. Recently, an active learning algorithm of deterministic timed automata is proposed in [36]. However, the number of locations in the learned automaton are highly increased compared to the original automaton, thus lacking interpretations. More introduction can be found in Section 7.

*The corresponding authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HSCC '24, May 14–16, 2024, Hong Kong SAR, China

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0522-9/24/05

<https://doi.org/10.1145/3641513.3650124>

In this paper, following the idea in [5] on active learning of deterministic one-clock timed automata, we extend the active learning method for deterministic timed automata (DTA) with multi-clocks under continuous-time semantics. Suppose that the number of clocks in the target DTA \mathcal{A} is known before learning, i.e. we know that \mathcal{A} has m clocks, our learning method can terminate and return an exact DTA \mathcal{H} equipped with m clocks, i.e. the recognized timed languages of \mathcal{A} and \mathcal{H} are equivalent.

The basic idea of our learning method is as follows. We define the reset-clocked languages of timed automata and present an equivalence relation on the reset-clocked languages. Based on the equivalence relation, we show that given a DTA, its reset-clocked language has a finite number of equivalent classes. Then, we prove that timed languages for two DTAs are equivalent if the reset-clocked languages are equivalent, which reduces the problem of learning the timed language of a DTA to that of learning a corresponding reset-clocked language.

Since a reset-clocked language includes the reset information which is not observable, we first consider a simple situation such that the learner learns from a *powerful teacher* who can answer three kinds of queries. In addition to answering membership queries and equivalence queries, the powerful teacher can provide the reset information on the transitions of target DTA by answering *reset information queries*. By defining the reset information query, we formalize the “powerful” teacher situation considered in [5]. Following L^* framework, we design the timed observation table to collect all membership queries and provide corresponding table operations. Compared to [5], we need to consider the clock regions due to the multiple clocks instead of clock valuations of a single clock. Moreover, we record the reset information of every clock in the table. When transforming an observation table into a hypothesis DTA, a two-step construction is proposed. We first transform the table into a DFA and then we present a partition function for recovering the timing constraints of each clock on each transition from a list of clock valuations, and thus get a DTA as the hypothesis in the following equivalence query.

After presenting learning DTA from a powerful teacher, we extend the learning algorithm to the situation of learning from a *normal teacher* who can only answer membership query and equivalence query. The main difference is that the learner now needs to guess the reset information instead of making reset information queries. By guessing the reset information in the observation table, every resulted possible table instance can be handled by the method in the powerful teacher situation. However, due to these guesses, the number of table instances is increased exponentially.

For both algorithms, we prove the termination and the correctness respectively. Moreover, we give the complexity analysis in the numbers of queries the learner needs to make.

In summary, our main contributions are as follows.

- An equivalence relation over reset-clocked languages.
- Two active learning algorithms for DTA with multi-clocks. One is learning from the powerful teacher who can answer reset information queries in addition, and the other is learning from the normal teacher who can only answer membership queries and equivalence queries.
- A preliminary implementation of the proposed method.

Structure. In what follows, Section 2 recalls the background on timed automata. The equivalence relation over reset-clocked languages is presented in Section 3. We then present the learning algorithms in the powerful teacher situation and in the normal teacher situation in Section 4 and Section 5, respectively. Section 6 reports the experimental results. Finally, Section 7 briefly summarizes the related work and Section 8 concludes the paper.

2 PRELIMINARIES

Let $\mathbb{R}_{\geq 0}$ and \mathbb{N} be the set of non-negative reals and natural numbers, respectively. $\mathbb{B} = \{\top, \perp\}$ represents the Boolean set, where \top is true and \perp is false. Let Σ be the fixed alphabet.

A *timed word* is a finite sequence $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, where t_i represents the delay time length before taking action σ_i for all $1 \leq i \leq n$. An equivalent definition is based on timestamps, i.e. $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$ where $\tau_k = \sum_{i=1}^k t_i$ for all $1 \leq k \leq n$. We also call such timed words as *delay-timed words* and use ϵ to denote the special empty word. A *timed language* \mathcal{L} is a set of timed words, i.e. $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$.

Timed Automata [2] is a kind of finite automata equipped with a set of real-valued clocks. Let C be the set of clock variables. A *clock constraint* ϕ is a conjunctive formula of atomic constraints of the form $c \sim n$, for $c \in C$ and $n \in \mathbb{N}$, where $\sim \in \{\leq, <, \geq, >, =\}$. We use $\Phi(C)$ to denote the set of clock constraints. A *clock valuation* $v : C \rightarrow \mathbb{R}_{\geq 0}$ is a function assigning a non-negative real value to each clock $c \in C$. We write $v \in \phi$ if the clock valuation v satisfies the clock constraint ϕ , i.e. ϕ evaluates to true using the values given by v . For $d \in \mathbb{R}_{\geq 0}$, let $v + d$ denote the clock valuation which maps every clock $c \in C$ to the value $v(c) + d$, and for a set $\mathcal{B} \subseteq C$, let $[\mathcal{B} \rightarrow 0]v$ denote the clock valuation which resets all clock variables in \mathcal{B} to 0 and agrees with v for other clocks in $C \setminus \mathcal{B}$.

Definition 2.1 (Timed automata [2]). A timed automaton is a tuple $\mathcal{A} = (\Sigma, L, l_0, F, C, \Delta)$, where

- Σ is the alphabet;
- L is a finite set of locations;
- l_0 is the initial location;
- $F \subseteq L$ is a set of accepting locations;
- C is the set of clocks;
- $\Delta \subseteq L \times \Sigma \times \Phi(C) \times 2^C \times L$ is a finite set of transitions.

A transition $\delta = (l, \sigma, \phi, \mathcal{B}, l')$ in Δ represents a jump from the source location l to the target location l' by performing an action $\sigma \in \Sigma$ if the constraint $\phi \in \Phi(C)$ is satisfied by current clock valuation. The set $\mathcal{B} \subseteq C$ gives the clocks to be reset with the transition. We call *guards* such transition constraints.

A *state* s of \mathcal{A} is a pair (l, v) , where $l \in L$ is a location and v is a clock valuation. A *run* ρ of \mathcal{A} over a timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$ is a sequence $\rho = (l_0, v_0) \xrightarrow{t_1, \sigma_1} (l_1, v_1) \xrightarrow{t_2, \sigma_2} \cdots \xrightarrow{t_n, \sigma_n} (l_n, v_n)$, satisfying the requirements: (1) l_0 is the initial location and $v_0(c) = 0$ for each clock $c \in C$; (2) for all $1 \leq i \leq n$, there is a transition $(l_{i-1}, \sigma_i, \phi_i, \mathcal{B}_i, l_i)$ such that $(v_{i-1} + t_i) \in \phi_i$ and $v_i = [\mathcal{B}_i \rightarrow 0](v_{i-1} + t_i)$. The run ρ is an *accepting run* if $l_n \in F$. For a run ρ over a timed word ω , we define the *trace* of ρ as the corresponding timed word $trace(\rho) = \omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ if $\rho \neq (l_0, v_0)$ or the empty word $trace(\rho) = \epsilon$ if

$\rho = (l_0, v_0)$. Given a timed automaton \mathcal{A} , we define its (recognized) timed language $\mathcal{L}(\mathcal{A}) = \{\text{trace}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$. We say two timed automata \mathcal{A}_1 and \mathcal{A}_2 are *equivalent* if and only if $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.

Given a timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$ and a timed automaton \mathcal{A} , if there exists a run ρ , then by recording the reset information along the run, we get a *reset-delay-timed word* corresponding to ω , denoted by $\omega_r = \text{trace}_r(\rho) = (\sigma_1, t_1, \mathbf{b}_1)(\sigma_2, t_2, \mathbf{b}_2) \cdots (\sigma_n, t_n, \mathbf{b}_n)$, where $\mathbf{b}_i \in \mathbb{B}^{|C|}$ is a $|C|$ -tuple recording whether the corresponding transition resets the clocks or not when taking the timed action (σ_i, t_i) . We have $\mathbf{b}_{i,j} = \top$ if the j -th clock c_j is in \mathcal{B}_i , otherwise $\mathbf{b}_{i,j} = \perp$, for all $c_j \in C$ and $1 \leq j \leq |C|$. For example, if there are 4 clocks c_1, c_2, c_3, c_4 , i.e., $|C| = 4$, and there is a transition $(l_{i-1}, \sigma_i, \phi_i, \{c_2, c_3\}, l_i)$ taking the timed action (σ_i, t_i) , then we have $\mathbf{b}_i = (\perp, \top, \top, \perp)$. Similarly, the (recognized) *reset-timed language* $\mathcal{L}_r(\mathcal{A}) = \{\text{trace}_r(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

Additionally, based on the run ρ , we introduce the *clocked word* $\gamma = (\sigma_1, \mathbf{v}_1)(\sigma_2, \mathbf{v}_2) \cdots (\sigma_n, \mathbf{v}_n)$, where $\mathbf{v}_i \in \mathbb{R}_{\geq 0}^{|C|}$ records the clock value for each clock when taking action σ_i , i.e. $\mathbf{v}_{i,j} = \mathbf{v}_{i-1,j} + t_i$ for all $c_j \in C$ and $1 \leq j \leq |C|$. After recording the reset information along the run ρ , we get the *reset-clocked word* $\gamma_r = (\sigma_1, \mathbf{v}_1, \mathbf{b}_1)(\sigma_2, \mathbf{v}_2, \mathbf{b}_2) \cdots (\sigma_n, \mathbf{v}_n, \mathbf{b}_n)$.

For a reset-clocked word γ_r , we define two functions: $\text{vw}(\gamma_r)$ returns the clocked word $\gamma = (\sigma_1, \mathbf{v}_1)(\sigma_2, \mathbf{v}_2) \cdots (\sigma_n, \mathbf{v}_n)$, and $\text{resets}(\gamma_r) = \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$ extract the reset information of γ_r , which is a sequence from \mathbf{b}_1 to \mathbf{b}_n .

We denote by \leq the *lexicographic order* on the product of $|C|$ clocks. Thus, $\mathbf{v}_1 \leq \mathbf{v}_2$ means that there exists $1 \leq j \leq |C|$ such that $\mathbf{v}_{1,j} < \mathbf{v}_{2,j}$ and $\mathbf{v}_{1,i} = \mathbf{v}_{2,i}$ for all $1 \leq i < j$, or specially $\mathbf{v}_1 = \mathbf{v}_2$.

Given a system modelled by a timed automaton, a (delay) timed word can be regarded as a system behavior from the view of the global clock outside of the system. However, it is difficult to reflect the changing of the internal logical clock variables C if the system is treated as a black box. It is the main challenge for learning the transition constraints which all correspond to the clock valuations rather than the global clock. While clocked words contain clock valuations, they can be regarded as system behaviors from the view of the internal logical clock variables and are more suitable for learning transition constraints.

Moreover, if we know the reset information \mathbf{b}_i along one run ρ over the timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$, we can transform ω to a clocked word $\gamma = (\sigma_1, \mathbf{v}_1)(\sigma_2, \mathbf{v}_2) \cdots (\sigma_n, \mathbf{v}_n)$ by taking

$$\mathbf{v}_{i,j} = \begin{cases} t_i, & \text{if } i = 1 \text{ or } \mathbf{b}_{i-1,j} = \top \text{ for all } 2 \leq i \leq n; \\ \mathbf{v}_{i-1,j} + t_i, & \text{otherwise.} \end{cases} \quad (1)$$

where $1 \leq j \leq |C|$. Here we define two functions Γ_ρ and π_ρ such that Γ_ρ maps a (reset-)delay-timed word to the (reset-)clocked counterpart and π_ρ maps a clocked word γ to its reset-clocked counterpart γ_r according to a run ρ respectively. When the run ρ is determined, we omit the subscript ' ρ ' and use Γ and π directly. Given a timed automaton \mathcal{A} , the *recognized clocked language* of \mathcal{A} is given as $\mathcal{L}(\mathcal{A}) = \{\Gamma_\rho(\text{trace}(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$ and the *recognized reset-clocked language* of \mathcal{A} defined as $\mathcal{L}_r(\mathcal{A}) = \{\pi_\rho(\text{trace}_r(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

Given a timed automaton \mathcal{A} with clocks C , a clocked word $\gamma \in (\Sigma \times \mathbb{R}_{\geq 0}^{|C|})^*$ is a *valid* clocked word of \mathcal{A} if there exists at

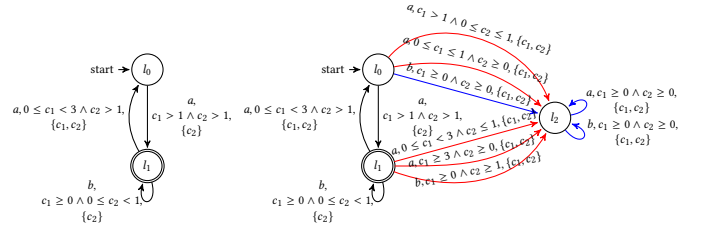


Figure 1: A DTA \mathcal{A} (left) and its corresponding CTA (right). The initial location l_0 is indicated by 'start' and the accepting location l_1 is doubly cycled.

least one run ρ corresponding to γ , i.e. $\gamma = \Gamma_\rho(\text{trace}(\rho))$. Similarly, a reset-clocked word $\gamma_r \in (\Sigma \times \mathbb{R}_{\geq 0}^{|C|} \times \mathbb{B}^{|C|})^*$ is a *valid* reset-clocked word of \mathcal{A} if there exists a run ρ corresponding to γ_r , i.e. $\gamma_r = \Gamma_\rho(\text{trace}_r(\rho)) = \pi_\rho(\Gamma_\rho(\text{trace}(\rho)))$. Otherwise, it is *invalid* w.r.t \mathcal{A} . Specially, we say a reset-clocked word is *doomed* if there is no timed automaton having a run corresponding to it. That means it cannot be transformed into a timed word according to Equation (1).

Definition 2.2 (Deterministic timed automata). A timed automaton \mathcal{A} is a *deterministic timed automaton* (DTA) if and only if there is at most one run ρ for any timed word ω .

It means that, for all $l \in L$ and $\sigma \in \Sigma$, for every pair of transitions of the form $(l, \sigma, \phi_1, -, -)$ and $(l, \sigma, \phi_2, -, -)$ in Δ , the clock constraints ϕ_1 and ϕ_2 are mutually exclusive.

A *complete deterministic timed automaton* (CTA) is that for any given timed word ω , there is exactly one run ρ . Given a DTA, it can be transformed into a CTA accepting the same timed language in three steps: (1) augment L with a "sink" location which is not an accepting location; (2) for every $l \in L$ and $\sigma \in \Sigma$, if there is no outgoing transition from l labelled with σ , introduce a transition resetting all clocks from l to "sink" with label σ and guards $c \geq 0$ for each clock $c \in C$; (3) otherwise, let $\text{Compl}_{l,\sigma}$ be the subset of $\mathbb{R}_{\geq 0}^{|C|}$ not covered by the guards of transitions from l with label σ . Write $\text{Compl}_{l,\sigma}$ as a union of guards I_1, \dots, I_k in a minimal way, then introduce a transition resetting all clocks from l to "sink" with label σ and guard I_j for each $1 \leq j \leq k$. Figure 1 depicts the transformation of DTA \mathcal{A} with $\Sigma = \{a, b\}$ and $C = \{c_1, c_2\}$ on the left into the CTA on the right by the above operation.

Since every DTA can be transformed into a CTA, in order to present our learning method clearly, we assume that we work with CTA in this paper.

According to the definition, a timed automaton has infinite states since the clock values are in $\mathbb{R}_{\geq 0}$. In the theory of timed automata, *clock region* is an abstraction used to form a finite partition of the state space. It will also be used in proving the termination of our learning algorithm in this paper. Here we review the definitions of regions and symbolic states of timed automata over regions introduced in [2].

Definition 2.3 (Region equivalence [2]). Given a timed automaton \mathcal{A} , let κ be a function, called a clock ceiling, mapping each clock $c \in C$ to the largest integer appearing in the guards over c . For a real number d , let $\text{frac}(d)$ denote the fractional part of d , and $\lfloor d \rfloor$ denote its integer part. Two clock valuations v, v' are region-equivalent, denoted by $v \sim v'$, iff all the following conditions hold:

- for all $c \in C$, either $\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor$ or both $v(c) > \kappa(c)$ and $v'(c) > \kappa(c)$.
- for all $c \in C$, if $v(c) \leq \kappa(c)$ then $\text{frac}(v(c)) = 0$ iff $\text{frac}(v'(c)) = 0$ and
- for all $c_i, c_j \in C$, if $v(c_i) \leq \kappa(c_i)$ and $v(c_j) \leq \kappa(c_j)$ then $\text{frac}(v(c_i)) \leq \text{frac}(v(c_j))$ iff $\text{frac}(v'(c_i)) \leq \text{frac}(v'(c_j))$.

A clock region for \mathcal{A} is an equivalence class induced by the equivalence relation \sim . For a clock valuation v , we denote by $\llbracket v \rrbracket$ the region containing it.

Given a timed automaton \mathcal{A} , there is only a finite number of regions. We denote the set of regions by \mathcal{R} . According to the definition of timed automata, given a transition constraint ϕ of \mathcal{A} if $v \sim v'$ then $v \in \phi$ if and only if $v' \in \phi$. The number of clock regions $|\mathcal{R}|$ is bounded by $|C|! \cdot 2^{|C|} \cdot \prod_{c \in C} (2\kappa(c) + 2)$ [2].

Definition 2.4 (Symbolic state [2]). A symbolic state of timed automata $\mathcal{A} = (L, l_0, F, C, \Sigma, \Delta)$ is a pair $(l, \llbracket v \rrbracket)$, where $l \in L$ and $\llbracket v \rrbracket$ is a clock region.

Therefore, the number of the symbolic states of \mathcal{A} is bounded by $|L| \cdot |C|! \cdot 2^{|C|} \cdot \prod_{c \in C} (2\kappa(c) + 2)$.

3 EQUIVALENCE RELATION ON RESET-CLOCKED WORDS

In this section, we aim to build an equivalence relation on reset-clocked words introduced in Section 2 and prove that it has a finite number of equivalence classes. We first introduce the *region word*, which is helpful in presenting the equivalence relation.

Definition 3.1 (Region word). Given a timed automaton \mathcal{A} , a region word ξ is a finite sequence of pairs $(\sigma, \llbracket v \rrbracket)$, where $\sigma \in \Sigma$ is an action of \mathcal{A} and $\llbracket v \rrbracket$ represents a region, i.e. $\llbracket v \rrbracket \in \mathcal{R}$.

For each clocked word γ of \mathcal{A} , there must be a unique region word ξ corresponding to γ . We denote it by $\xi = \llbracket \gamma \rrbracket$. For example, if $\gamma = (a, \{0, 0\})(a, \{1.5, 1.5\})$, then $\xi = \llbracket \gamma \rrbracket = (a, c_1 = 0 \wedge c_2 = 0)(a, 1 < c_1 < 2 \wedge 1 < c_2 < 2 \wedge c_1 = c_2)$.

LEMMA 3.2. *Given a CTA \mathcal{A} , for all valid clocked words γ and γ' such that $\llbracket \gamma \rrbracket = \llbracket \gamma' \rrbracket$, they have the same transition sequence of \mathcal{A} and reach the same symbolic state. Thus, the following two conclusions hold:*

- $\gamma \in \mathcal{L}(\mathcal{A})$ iff $\gamma' \in \mathcal{L}(\mathcal{A})$;
- $\text{resets}(\gamma_r) = \text{resets}(\gamma'_r)$, where γ_r and γ'_r are the reset-clocked words corresponding to γ and γ' , respectively.

The first conclusion shows that given a region word ξ , all valid clocked words γ with $\xi = \llbracket \gamma \rrbracket$ have the same accepting result over \mathcal{A} . The second conclusion tells that such clocked words witness the same reset information over the transitions of \mathcal{A} .

Definition 3.3 (Valid successor). Given a CTA \mathcal{A} , a reset-clocked word γ_r and a region word ξ , we say a reset-clocked word γ'_r is a valid successor of γ_r corresponding to ξ if it satisfies two conditions:

- $\llbracket \text{ow}(\gamma'_r) \rrbracket = \xi$;
- $\gamma_r \cdot \gamma'_r$ is a valid reset-clocked word of \mathcal{A} , i.e. there exists a run ρ in \mathcal{A} such that $\gamma_r \cdot \gamma'_r = \Gamma(\text{trace}_r(\rho))$.

We denote the set of *valid successors* of γ_r corresponding to ξ by $\text{vs}_{\mathcal{A}}(\gamma_r, \xi)$.

Example 3.4. Consider the CTA \mathcal{A} in Fig. 1. Given a valid reset-clocked word $\gamma_r = (a, \{1.1, 1.1\}, \{\perp, \top\})$ and a region word $\xi = (b, 1 < c_1 < 2 \wedge c_2 = 0)$, $\gamma'_r = (b, \{1.1, 0\}, \{\perp, \top\})$ is a valid successor of γ_r corresponding to ξ since $\llbracket \text{ow}(\gamma'_r) \rrbracket = \llbracket (b, \{1.1, 0\}) \rrbracket = \xi$ and $\gamma_r \cdot \gamma'_r$ corresponds to the run $\rho = (l_0, \{0, 0\}) \xrightarrow{a, 1.1} (l_1, \{1.1, 0\}) \xrightarrow{b, 0} (l_1, \{1.1, 0\})$. We denote it by $\gamma'_r \in \text{vs}_{\mathcal{A}}(\gamma_r, \xi)$.

LEMMA 3.5. *Given a valid reset-clocked word γ_r of a CTA \mathcal{A} and a region word ξ , for all $\gamma'_r, \gamma''_r \in \text{vs}_{\mathcal{A}}(\gamma_r, \xi)$, $\gamma_r \gamma'_r$ and $\gamma_r \gamma''_r$ witness the same transition sequence of \mathcal{A} and reach the same symbolic state in the end.*

LEMMA 3.6. *Given two valid reset-clocked words γ_{r1} and γ_{r2} of \mathcal{A} and a region word ξ , if γ_{r1} and γ_{r2} reach the same symbolic state of \mathcal{A} , then for all $\gamma'_{r1} \in \text{vs}_{\mathcal{A}}(\gamma_{r1}, \xi)$ and $\gamma'_{r2} \in \text{vs}_{\mathcal{A}}(\gamma_{r2}, \xi)$, the following two conditions hold:*

- $\text{resets}(\gamma'_{r1}) = \text{resets}(\gamma'_{r2})$;
- $\gamma_{r1} \gamma'_{r1}$ and $\gamma_{r2} \gamma'_{r2}$ reach some same symbolic state.

Given a DTA \mathcal{A} , let Σ be the finite alphabet and \mathcal{R} the finite set of regions of \mathcal{A} . We let $\Sigma = \Sigma \times \mathbb{R}_{\geq 0}^{|C|}$ be the infinite set of clocked actions, $\Sigma_G = \Sigma \times \mathcal{R}$ be the finite set of region actions, and $\Sigma_r = \Sigma \times \mathbb{R}_{\geq 0}^{|C|} \times \mathbb{B}^{|C|}$ be the infinite set of reset-clocked actions. An equivalence relation over the reset-clocked language $\mathcal{L}_r(\mathcal{A})$ is defined as follows.

Definition 3.7 (Equivalence relation). Let $\mathcal{L}_r(\mathcal{A})$ be the reset-clocked language of a DTA \mathcal{A} , two reset-clocked words $\gamma_{r1}, \gamma_{r2} \in \Sigma_r^*$ are indistinguishable by $\mathcal{L}_r(\mathcal{A})$, denoted by $\gamma_{r1} \sim_{\mathcal{L}_r(\mathcal{A})} \gamma_{r2}$, if for all $\gamma'_{r1} \in \text{vs}_{\mathcal{A}}(\gamma_{r1}, \xi)$ and $\gamma'_{r2} \in \text{vs}_{\mathcal{A}}(\gamma_{r2}, \xi)$, the following conditions holds for all $\xi \in \Sigma_G^*$:

- $\gamma_{r1} \gamma'_{r1} \in \mathcal{L}_r(\mathcal{A})$ iff $\gamma_{r2} \gamma'_{r2} \in \mathcal{L}_r(\mathcal{A})$;
- $\text{resets}(\gamma'_{r1}) = \text{resets}(\gamma'_{r2})$.

LEMMA 3.8. *If two valid reset-clocked words γ_{r1}, γ_{r2} of \mathcal{A} reach the same symbolic state of \mathcal{A} , then $\gamma_{r1} \sim_{\mathcal{L}_r(\mathcal{A})} \gamma_{r2}$.*

LEMMA 3.9. *All invalid reset-clocked words belong to the same equivalence class of $\sim_{\mathcal{L}_r(\mathcal{A})}$.*

THEOREM 3.10. *Let $\mathcal{L}_r(\mathcal{A})$ be the reset-clocked language of a DTA \mathcal{A} , and then $\sim_{\mathcal{L}_r(\mathcal{A})}$ has a finite number of equivalence classes.*

4 LEARNING DTA FROM A POWERFUL TEACHER

In this section, we present the algorithm for learning a DTA \mathcal{A} from a powerful teacher. The basic idea is that we transform the learning problem for the timed language of an underlying DTA \mathcal{A} into the learning problem for the reset-clocked language of \mathcal{A} . We first introduce the basic idea and then describe the settings of the powerful teacher. After presenting the details of the learning algorithm, we show its correctness, termination and complexity.

THEOREM 4.1. *Given two DTAs \mathcal{A}_1 and \mathcal{A}_2 , if $\mathcal{L}_r(\mathcal{A}_1) = \mathcal{L}_r(\mathcal{A}_2)$, then $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.*

According to Theorem 4.1, to construct a DTA that recognizes a target timed language $\mathcal{L} = \mathcal{L}(\mathcal{A})$, it suffices to learn a hypothesis DTA \mathcal{H} such that $\mathcal{L}_r(\mathcal{A}) = \mathcal{L}_r(\mathcal{H})$. Instead of checking directly if $\mathcal{L}_r(\mathcal{A}) = \mathcal{L}_r(\mathcal{H})$, the contraposition of the theorem guarantees

that we can still perform equivalence queries over their timed languages: if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H})$, then \mathcal{H} recognizes the target language already; otherwise, a counterexample making $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{H})$ also yields an evidence for $\mathcal{L}_r(\mathcal{A}) \neq \mathcal{L}_r(\mathcal{H})$.

There are two roles in the learning algorithm: a learner and a powerful teacher. The teacher holds the underlying target DTA \mathcal{A} and three oracles to answer three kinds of queries: *reset information query*, *membership query* and *equivalence query* from the learner. For a reset information query, the learner asks the reset information along the run of a valid clocked word γ . The teacher returns the reset information of the clocks, and the learner gets a reset-clocked word. For a membership query, the teacher receives a reset-clocked word γ_r from the learner and answers whether $\gamma_r \in \mathcal{L}_r(\mathcal{A})$. For an equivalence query, the learner submits a hypothesis DTA \mathcal{H} and the teacher answers whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H})$. If the answer is no, the powerful teacher returns a reset-delay-timed word ω_r as a counterexample in addition. We denote the three oracles for reset information query, membership query and equivalence query by RQ, MQ and EQ, respectively.

Suppose that the DTA \mathcal{A} is a complete DTA and the learner knows the number of the clocks $|C|$ in advance. We show more details as follows.

4.1 Timed observation table, reset information query, and membership query

Similar to the classic L^* algorithm, we design an observation table to collect the query results.

Definition 4.2 (Timed observation table for DTA). A timed observation table for DTA is a tuple $T = (\Sigma, \Sigma_G, \Sigma_r, S, R, E, f, g)$, where $S, R \subset \Sigma_r^*$ are finite sets of reset-clocked words, named prefix set and boundary, respectively, and $E \subset \Sigma_G^*$ is a finite set of region words, named suffix set. Specifically,

- S and R are disjoint, i.e., $S \cup R = S \uplus R$, and $S \cup R$ is prefix-closed.
- The empty word is, by default, both a prefix and a suffix, i.e., $\epsilon \in E$ and $\epsilon \in S$;
- $f : S \cup R \times E \rightarrow \{+, -\}$ is the classification function such that for a reset-clocked word $\gamma_r \in S \cup R$ and a region word $e \in E$, $f(\gamma_r, e) = -$ if $vs_{\mathcal{A}}(\gamma_r, e) = \emptyset$ or $MQ(\gamma_r e_r) = -$ i.e. $\gamma_r e_r \notin \mathcal{L}_r(\mathcal{A})$ where $e_r \in vs_{\mathcal{A}}(\gamma_r, e)$, and $f(\gamma_r, e) = +$ if $MQ(\gamma_r e_r) = +$, i.e. $\gamma_r e_r \in \mathcal{L}_r(\mathcal{A})$.
- $g : S \cup R \times E \rightarrow \{\perp, \top\}^{|e| \times |C|}$ is a labelling function such that for a reset-clocked word $\gamma_r \in S \cup R$ and a region word $e \in E \setminus \{\epsilon\}$, $g(\gamma_r, e) = resets(e_r)$ where $e_r \in vs_{\mathcal{A}}(\gamma_r, e)$ if $vs_{\mathcal{A}}(\gamma_r, e) \neq \emptyset$, otherwise $g(\gamma_r, e) = \{\top\}^{|e| \times |C|}$.

Consider the classification function f . It records the results of the membership queries i.e., whether $\gamma_r e_r$ is in $\mathcal{L}_r(\mathcal{A})$. Depending on Lemma 3.5 and Lemma 3.8, e_r can represent all other valid successors e'_r according to the region word e such that $\gamma_r e_r \sim_{\mathcal{L}_r(\mathcal{A})} \gamma_r e'_r$. The function g records the reset information in the valid successor e_r and assigns all resets to \top if there is no valid successor e_r . However, according to Definition 3.3, the valid successor e_r depends on a known DTA \mathcal{A} . We claim that the learner can compute the valid successors using reset information queries instead of knowing DTA \mathcal{A} .

Algorithm 1: FindValidSuccessor

input : A reset-clocked word
 $\gamma_r = (\sigma_1, v_1, b_1)(\sigma_2, v_2, b_2) \cdots (\sigma_n, v_n, b_n) \in S \cup R$;
 a region word $e = (\sigma'_1, \llbracket v_1 \rrbracket)(\sigma'_2, \llbracket v_2 \rrbracket) \cdots (\sigma'_k, \llbracket v_k \rrbracket) \in E$.

output : A valid successor $e_r \in vs_{\mathcal{A}}(\gamma_r, e)$.

```

1  $e_r \leftarrow \epsilon$ ;
2 if  $\gamma_r$  is doomed then
3   return  $\perp$ 
4 else
5   if  $\gamma_r = \epsilon$  then
6      $v_n \leftarrow 0; b_n \leftarrow \top$ ; // Set the initial clock valuation.
7   for  $i \leftarrow 1$  to  $k$  do
8     for  $j \leftarrow 1$  to  $|C|$  do
9       if  $b_{n+i-1,j} = \top$  then
10         $v_{n+i,j} \leftarrow 0$ ;
11       else
12         $v_{n+i,j} \leftarrow v_{n+i-1,j}$ ;
13      $flag, d \leftarrow Solve(\llbracket v_{n+i} + d \rrbracket = \llbracket v_i \rrbracket)$ 
14     if  $flag = \perp$  then
15       return  $\perp$ ;
16     else
17        $v_{n+i} \leftarrow v_{n+i} + d$ ;
18      $b_{n+i} \leftarrow RQ(vw(\gamma_r) \cdot (\sigma'_i, v_{n+i}))$ ;
19      $e_r \leftarrow e_r \cdot (\sigma'_i, v_{n+i}, b_{n+i})$ ;
20      $\gamma_r \leftarrow \gamma_r \cdot (\sigma'_i, v_{n+i}, b_{n+i})$ ;
21 return  $e_r$ ;
```

Algorithm 1 shows the computation for a valid successor e_r . Given a reset-clocked word $\gamma_r \in S \cup R$, we can use Equation 1 to determine whether γ_r is doomed. If it is doomed, then we stop the computation as we know that there is no valid successor (Line 3). Otherwise, we compute e_r by considering the region actions in the region word $e \in E$ one by one. For the first region action $(\sigma'_1, \llbracket v_1 \rrbracket)$, depending on the last reset information in γ_r , we can compute the clock values v_{n+1} . Then we find a delay time $d \in \mathbb{R}_{\geq 0}$ such that $v_{n+1} + d \in \llbracket v_1 \rrbracket$ by the function $Solve()$ and use $flag$ to represent whether there is such d or not. If there is no such d , it means that there is no clock valuation in $\llbracket v_1 \rrbracket$ that can form a valid successor. Otherwise, we update v_{n+1} and get a clocked word (σ'_1, v_{n+1}) . Then we make a reset information query for $vw(\gamma_r) \cdot (\sigma'_1, v_{n+1})$ to get the last reset information b_{n+1} (Line 18). After that, we update e_r and γ_r . The computation for other region actions in e is similar.

Example 4.3. Fig. 2 shows an example of a timed observation table. Suppose that the CTA \mathcal{A} in Fig. 1 is the underlying target automaton. Consider $\gamma_r = (a, \{0, 0\}, \{\top, \top\})$ and $e = (a, 1 < c_1 < 2 \wedge 1 < c_2 < 2 \wedge c_1 = c_2)$, $n = 1$ for the length of γ_r and there is only one region action in e . According to Algorithm 1, we get $e_r = (a, \{1.05, 1.05\}, \{\top, \top\}) \in vs_{\mathcal{A}}(\gamma_r, e)$.

Similar to the L^* algorithm, for the observation table, we also define a function row mapping each $\gamma_r \in S \cup R$ to a vector indexed by every $e \in E$. Each element of the vector is defined as $\{f(\gamma_r, e), g(\gamma_r, e)\}$. For example, in Fig. 2, consider $\gamma_r = (a, \{1.05, 1.05\}, \{\perp, \top\})$, we have $row(\gamma_r) = \{\{+\}, \{-, \top, \top\}\}$.

Before constructing a hypothesis \mathcal{H} based on the timed observation table T , the learner has to ensure that T satisfies the following conditions:

- *Reduced*: $\forall s, s' \in S: s \neq s' \implies row(s) \neq row(s')$;
- *Closed*: $\forall r \in R, \exists s \in S: row(s) = row(r)$;

| | ϵ | $(a, 1 < c_1 < 2 \wedge 1 < c_2 < 2 \wedge c_1 = c_2)$ |
|---|------------|--|
| ϵ | – | $+, \perp, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})$ | + | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{1.05, 0\}, \{\perp, \top\})$ | + | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{2.05, 1\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{1.05, 1.05\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |

Figure 2: An example of the observation table.

| T1 | ϵ |
|---|------------|
| ϵ | – |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})$ | + |
| $(a, \{0, 0\}, \{\top, \top\})$ | – |
| $(b, \{0, 0\}, \{\top, \top\})$ | – |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{1.05, 0\}, \{\perp, \top\})$ | + |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{2.05, 1\}, \{\top, \top\})$ | – |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{1.05, 1.05\}, \{\top, \top\})$ | – |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – |
| $(a, \{0, 0\}, \{\top, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – |

| T2 | ϵ | $(a, 1 < c_1 < 2 \wedge 1 < c_2 < 2 \wedge c_1 = c_2)$ |
|---|------------|--|
| ϵ | – | $+, \perp, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})$ | + | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{1.05, 0\}, \{\perp, \top\})$ | + | $-, \top, \top$ |
| $(a, \{1.05, 1.05\}, \{\perp, \top\})(b, \{2.05, 1\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{1.05, 1.05\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(a, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |
| $(a, \{0, 0\}, \{\top, \top\})(b, \{0, 0\}, \{\top, \top\})$ | – | $-, \top, \top$ |

Figure 3: Example of making the observation table consistent.

- **Consistent:** $\forall \gamma_r, \gamma'_r \in S \cup R$, $\text{row}(\gamma_r) = \text{row}(\gamma'_r)$ implies that: for all $\sigma_r, \sigma'_r \in \Sigma_r$ such that $\llbracket vw(\sigma_r) \rrbracket = \llbracket vw(\sigma'_r) \rrbracket$, if $\gamma_r \sigma_r, \gamma'_r \sigma'_r \in S \cup R$, the following conditions are satisfied:
 - $\text{row}(\gamma_r \sigma_r) = \text{row}(\gamma'_r \sigma'_r)$;
 - $\text{resets}(\sigma_r) = \text{resets}(\sigma'_r)$.
- **Evidence-closed:** $\forall s \in S$ and $\forall e \in E$, if $\text{vs}_{\mathcal{A}}(s, e)$ is not empty, there is $e_r \in \text{vs}_{\mathcal{A}}(s, e)$ such that $se_r \in S \cup R$;

A timed observation table T is prepared if it satisfies the above four conditions. To get the table prepared, the learner can perform the following operations.

Making T closed. If there is $r \in R$ such that for all $s \in S$, $\text{row}(r) \neq \text{row}(s)$, the learner will move r from R to S . Moreover, for each $\sigma \in \Sigma$, a reset-clocked word $\pi(vw(r) \cdot \sigma)$ needs to be added to R , where $\sigma = (\sigma, \{0\}^{|C|})$. The mapping π defined in Section 2 can be implemented by a reset information query in the powerful teacher situation.

Making T consistent. Considering the first case, there exist two reset-clocked words $\gamma_r, \gamma'_r \in S \cup R$ at least, for $\sigma_r, \sigma'_r \in \Sigma_r$ such that $\llbracket vw(\sigma_r) \rrbracket = \llbracket vw(\sigma'_r) \rrbracket$, both $\gamma_r \sigma_r$ and $\gamma'_r \sigma'_r \in S \cup R$. However, $\text{row}(\gamma_r) = \text{row}(\gamma'_r)$ but $\text{row}(\gamma_r \sigma_r) \neq \text{row}(\gamma'_r \sigma'_r)$. In this case, there must be $e \in E$ such that $f(\gamma_r \sigma_r, e) \neq f(\gamma'_r \sigma'_r, e)$ or $g(\gamma_r \sigma_r, e) \neq g(\gamma'_r \sigma'_r, e)$. Thus, the learner adds $\llbracket vw(\sigma_r) \rrbracket \cdot e$ to E , so that γ_r and γ'_r are separated. Consider the second case that $\text{resets}(\sigma_r) \neq \text{resets}(\sigma'_r)$. In this case, $\llbracket vw(\sigma_r) \rrbracket$ is enough to distinguish γ_r from γ'_r . So, the learner adds $\llbracket vw(\sigma_r) \rrbracket$ to E . After extending E , the learner fills the table by making reset information queries and membership queries.

Making T evidence-closed. In this case, $\exists s \in S$ and $\exists e \in E$, $\text{vs}_{\mathcal{A}}(s, e)$ is not empty, and there is no $e_r \in \text{vs}_{\mathcal{A}}(s, e)$ such that $se_r \in S \cup R$. The learner finds $e_r \in \text{vs}_{\mathcal{A}}(s, e)$ and then adds all prefixes of se_r to R , except for those already in $S \cup R$. Similarly, the learner needs to fill the table.

The condition that an observation table is reduced is inherently preserved by the aforementioned operations, together with the counterexample processing described later in Section 4.3. Furthermore, a table may need several rounds of these operations before being prepared since certain conditions may be violated by different interleaved operations.

Example 4.4. Fig. 3 shows a process for making the table consistent. Assuming that the CTA in Fig. 1 recognizes the target timed language and an observation table $T1$ is generated in the learning process. We show the inconsistency of $T1$ as follows. Let $\gamma_r = \epsilon$ and $\gamma'_r = (a, \{0, 0\}, \{\top, \top\})$, we have $\text{row}(\gamma_r) = \text{row}(\gamma'_r)$. Choose $\sigma_r = (a, \{1.05, 1.05\}, \{\perp, \top\})$ and $\sigma'_r = (a, \{1.05, 1.05\}, \{\top, \top\})$ such that $\llbracket vw(\sigma_r) \rrbracket = \llbracket vw(\sigma'_r) \rrbracket$. We have $\gamma_r \sigma_r = (a, \{1.05, 1.05\}, \{\perp, \top\})$ and $\gamma'_r \sigma'_r = (a, \{0, 0\}, \{\top, \top\})$ ($(a, \{1.05, 1.05\}, \{\top, \top\})$ in $S \cup R$). However, we find $\text{row}(\gamma_r \sigma_r) \neq \text{row}(\gamma'_r \sigma'_r)$ and $\text{resets}(\sigma_r) \neq \text{resets}(\sigma'_r)$. Thus, $T1$ violates the two conditions of consistency. In order to make the table consistent, we add $\llbracket (a, \{1.05, 1.05\}) \rrbracket \cdot \epsilon$, which equals $(a, 1 < c_1 < 2 \wedge 1 < c_2 < 2 \wedge c_1 = c_2)$, to E to distinguish ϵ and $(a, \{0, 0\}, \{\top, \top\})$, and then fill the table and get $T2$.

4.2 Hypothesis construction

When the observation table T is prepared, a hypothesis \mathcal{H} can be constructed based on T in two steps: 1. Build a DFA M from T ; 2. Convert M into DTA \mathcal{H} .

Given a prepared table $T = (\Sigma, \Sigma_G, \Sigma_r, S, R, E, f, g)$, the learner first builds a DFA $M = (L_M, l_M^0, F_M, \Sigma_M, \Delta_M)$ as follows.

- the finite set of locations $L_M = \{l_{\text{row}(s)} \mid s \in S\}$;
- the initial location $l_M^0 = l_{\text{row}(\epsilon)}$ for $\epsilon \in S$;
- the finite set of accepting locations $F_M = \{l_{\text{row}(s)} \mid f(s \cdot \epsilon) = + \text{ for } s \in S \text{ and } \epsilon \in E\}$;
- the finite abstract alphabet $\Sigma_M = \{\sigma_r \in \Sigma_r \mid \gamma_r \sigma_r \in S \cup R \text{ for } \gamma_r \in \Sigma_r^*\}$;
- the finite set of transitions $\Delta_M = \{(l_{\text{row}(\gamma_r)}, \sigma_r, l_{\text{row}(\gamma_r \sigma_r)}) \mid \gamma_r, \gamma_r \sigma_r \in S \cup R \text{ for } \sigma_r \in \Sigma_r\}$.

The basic idea in the above construction is that we view $\sigma_r \in \Sigma_r$ as an abstract action if σ_r occurs in the table and consider each row in the table as the location which is reached by the row. We label the locations by function row . The constructed DFA M is compatible with the timed observation table T in a sense captured by the following lemma.

LEMMA 4.5. *Given a prepared observation table $T = (\Sigma, \Sigma_G, \Sigma_r, S, R, E, f, g)$, for all $\gamma_r \cdot e \in (S \cup R) \cdot E$, we have*

- if $f(\gamma_r, e) = +$, there is $\gamma'_r \in \text{vs}_{\mathcal{A}}(\gamma_r, e)$ such that $\text{resets}(\gamma'_r) = g(\gamma_r, e)$ and $\gamma_r \gamma'_r$ is accepted in the constructed DFA M ;
- if $f(\gamma_r, e) = -$ and $\text{vs}_{\mathcal{A}}(\gamma_r, e) \neq \emptyset$, there is $\gamma'_r \in \text{vs}_{\mathcal{A}}(\gamma_r, e)$ such that $\text{resets}(\gamma'_r) = g(\gamma_r, e)$ and $\gamma_r \gamma'_r$ is not accepted in the constructed DFA M .
- if $f(\gamma_r, e) = -$ and $\text{vs}_{\mathcal{A}}(\gamma_r, e) = \emptyset$, DFA M has no accepting word in the form of $\gamma_r \gamma'_r$ such that $\llbracket vw(\gamma'_r) \rrbracket = e$.

After the DFA M is constructed, the learner constructs a hypothesis $\mathcal{H} = (L, l_0, F, C, \Sigma, \Delta)$ from M . In the hypothesis \mathcal{H} , C is the set of clocks known before learning and Σ is the given alphabet as in T . In addition, $L = L_M$, $l_0 = l_M^0$, and $F = F_M$. The set of transitions Δ in \mathcal{H} can be constructed as follows: For any $l \in L_M$ and $\sigma \in \Sigma$, let $\Psi_{l,\sigma} = \{v_i \mid (l, (\sigma, v_i, b_i), l') \in \Delta_M\}$, then applying the partition function $P^c(\cdot)$ to $\Psi_{l,\sigma}$ returns n disjoint clock constraints, written as I_1, \dots, I_n , satisfying $v_i \in I_i$ for any $1 \leq i \leq n$, where $n = |\Psi_{l,\sigma}|$. Then for each transition $(l, (\sigma, v_i, b_i), l') \in \Delta_M$, we add a transition $(l, \sigma, I_i, \{c_j \mid b_{i,j} = \top\}, l')$ in Δ . After that, the construction of hypothesis \mathcal{H} is finished.

Definition 4.6 (Partition function). Given a set of clock valuations $\Psi_{l,\sigma}$, the partition function $P(\cdot)$ operates on it in the five steps:

- For all clock valuations in $\Psi_{l,\sigma}$, we list them as v_1, v_2, \dots, v_n , with $\{0\}^{|C|} = v_1 \leq v_2 \leq \dots \leq v_n$. Then for $1 \leq i \leq n$, we map v_i to a constraint A_i , which is defined as

$$A_i = \begin{cases} \llbracket v_i \rrbracket, & \text{if } \exists 1 \leq j \leq |C| \text{ satisfying that } v_{i,j} > \kappa(c_j); \\ \emptyset, & \text{otherwise.} \end{cases}$$

We denote the union of A_i for all $1 \leq i \leq n$ by U_0 .

- After U_0 is constructed, for $1 \leq i \leq n$, we map clock valuation v_i to a new constraint $U_i = \bigwedge_{1 \leq j \leq |C|} ic_{i,j}$ where

$$ic_{i,j} = \begin{cases} c_j \geq v_{i,j}, & \text{if } v_{i,j} \in \mathbb{N}; \\ c_j > \lfloor v_{i,j} \rfloor, & \text{otherwise.} \end{cases}$$

- Based on U_0 constructed in the first step and U_1, \dots, U_n constructed in the second step, map the n clock valuations v_1, v_2, \dots, v_n to the n constraints W_1, W_2, \dots, W_n , respectively, where W_i is defined as

$$\begin{cases} U_n - (U_n \cap U_0), & \text{if } i = n; \\ U_i - (U_i \cap (W_{i+1} \cup W_{i+2} \cup \dots \cup W_n \cup U_0)), & \text{otherwise.} \end{cases}$$

- Based on W_1, \dots, W_n and A_1, \dots, A_n , map the n clock valuations v_1, v_2, \dots, v_n to the n constraints I_1, I_2, \dots, I_n , respectively, which is defined as

$$I_i = W_i + A_i.$$

- Finally, find whether there are v_i, v_j such that $A_i = A_j = \emptyset$ and $U_i = U_j$, but $\llbracket v_i \rrbracket \neq \llbracket v_j \rrbracket$. If so, we support that $v_i \leq v_j$, then after the previous three steps, $W_i = \emptyset$ and $W_j = U_j - (U_j \cap (W_{j+1} \cup W_{j+2} \cup \dots \cup W_n \cup U_0))$ since $U_i = U_j$. Next, in the fourth step, $I_i = \emptyset$ and $I_j = W_j$ since $A_i = A_j = \emptyset$. To map v_i to the correct partition, let $I_i = \llbracket v_i \rrbracket$ and thus the updated $I_j = W_j - I_i$. Note that after this step, the union of the updated I_i and I_j is not changed.

The basic idea of the partition function is to generate disjoint clock constraints I_1, \dots, I_n containing the clock valuations. Specially, we need to generate that for each v_i , there is one and only one final constraint I_i containing it. In the first step, to avoid dividing a region $\llbracket v_i \rrbracket$ into more than one final constraint, we select $\llbracket v_i \rrbracket$ which may lead to this situation, add it to U_0 and guarantee that U_0 does not participate in the division operations in the second step and third step. In the second step, we compute a coarse constraints U_i containing the corresponding clock valuations v_i . But they are not disjoint. In the third step, we guarantee the disjoint property

and get the constraints W_i . In the fourth step, since U_0 does not participate in the division operations in the second step and third step, we select A_i from U_0 and add it to W_i . In the second step and third step, we do not consider the relation between clocks in one clock valuation. But they may form different regions (constraints). Hence, for two clock valuations such that $\llbracket v_i \rrbracket \neq \llbracket v_j \rrbracket$ but $U_i = U_j$ and $A_i = A_j = \emptyset$, if $v_i \leq v_j$, then $I_i = \emptyset$, which means that v_i has no corresponding final constraint containing it. Therefore, in the final step, we divide the constraint I_j into two constraints I_i and I_j .

LEMMA 4.7. *The clock constraints I_1, \dots, I_n generated by partition function $P(\cdot)$ are disjoint, i.e. $I_i \cap I_j = \emptyset$ for all $i \neq j$ and $1 \leq i, j \leq n$.*

THEOREM 4.8. *$P(\cdot)$ returns a partition on $\mathbb{R}_{\geq 0}^{|C|}$.*

LEMMA 4.9. *Given a DFA $M = (L_M, l_M^0, F_M, \Sigma_M, \Delta_M)$, which is generated from a prepared timed observation table T , the hypothesis $\mathcal{H} = (L, l_0, F, C, \Sigma, \Delta)$ is transformed from M . For all $\gamma_r \cdot e \in (S \cup R) \cdot E$, we have*

- if $f(\gamma_r, e) = +$, there is $\gamma'_r \in vs_{\mathcal{A}}(\gamma_r, e)$ such that $resets(\gamma'_r) = g(\gamma_r, e)$ and $\gamma_r \gamma'_r$ is accepted in the constructed hypothesis \mathcal{H} ;
- if $f(\gamma_r, e) = -$ and $vs_{\mathcal{A}}(\gamma_r, e) \neq \emptyset$, there is $\gamma'_r \in vs_{\mathcal{A}}(\gamma_r, e)$ such that $resets(\gamma'_r) = g(\gamma_r, e)$ and $\gamma_r \gamma'_r$ is not accepted in the constructed hypothesis \mathcal{H} .
- if $f(\gamma_r, e) = -$ and $vs_{\mathcal{A}}(\gamma_r, e) = \emptyset$, \mathcal{H} has no accepting word in the form of $\gamma_r \gamma'_r$ such that $\llbracket vw(\gamma'_r) \rrbracket = e$.

THEOREM 4.10. *The hypothesis \mathcal{H} is a CTA.*

THEOREM 4.11. *For all $\gamma_r \cdot e \in (S \cup R) \cdot E$, for every valid clocked word γ' such that $\llbracket \gamma' \rrbracket = \llbracket vw(\gamma_r) \rrbracket \cdot e$, \mathcal{H} accepts $\pi(\gamma') = \gamma'_r$ if and only if $f(\gamma_r, e) = +$.*

4.3 Equivalence query and counterexample processing

The equivalence problem determining whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{A})$ can be decomposed into the two-side inclusion problem, i.e., whether $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{H})$. Since the deterministic timed automata are closed under complementation [2], we can implement the equivalence oracle as follows: $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{A})$ iff $\mathcal{L}(\mathcal{H}) \cap \overline{\mathcal{L}(\mathcal{A})} = \emptyset$, and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{H})$ iff $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{H})} = \emptyset$. If $\mathcal{L}(\mathcal{H}) \not\subseteq \mathcal{L}(\mathcal{A})$, there is a delay-timed word ω where the corresponding run ρ in \mathcal{H} reaches an accepting location and the corresponding run ρ' in \mathcal{A} reaches an unaccepting location. In this case, the teacher provides the corresponding reset-delay-timed word of ω_r in the target automaton \mathcal{A} as a negative counterexample $ctx_- = (\omega_r, -)$. Similarly, if $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{H})$, the teacher provides a positive counterexample $ctx_+ = (\omega_r, +)$. When receiving a counterexample $ctx = (\omega_r, +/-)$, the learner converts the counterexample to a reset-clocked word $\gamma_r = (\sigma_1, v_1, b_1)(\sigma_2, v_2, b_2) \dots (\sigma_n, v_n, b_n)$ and add all prefixes of γ_r to R except for those already in $S \cup R$. According to the definition of the reset-clocked word, ω_r and γ_r trigger the same sequence of transitions in \mathcal{A} . According to the contraposition of Theorem 4.1, if ω is an evidence of $\mathcal{L}(\mathcal{H}) \neq \mathcal{L}(\mathcal{A})$, γ_r is an evidence of $\mathcal{L}_r(\mathcal{H}) \neq \mathcal{L}_r(\mathcal{A})$.

4.4 Learning algorithm

The learning process that integrates all the previously stated ingredients is presented in Algorithm 2. First, the learner initializes a timed

Algorithm 2: Learning deterministic timed automata from a powerful teacher

input : the timed observation table $T = (\Sigma, \Sigma_G, \Sigma_r, S, R, E, f, g)$; the number of clocks $|C|$.

output : the hypothesis \mathcal{H} recognizing the target language \mathcal{L} .

```

1  $S \leftarrow \{\epsilon\}$ ;  $R \leftarrow \{\pi(\gamma) \mid \gamma = (\sigma, \{0\}^{|C|}), \forall \sigma \in \Sigma\}$ ;  $E \leftarrow \{\epsilon\}$ ;
   // initialization
2 fill  $T$  by membership queries;
3  $equivalent \leftarrow \perp$ ;
4 while  $equivalent = \perp$  do
5    $prepared \leftarrow is\_prepared(T)$ ; // whether the table is prepared
6   while  $prepared = \perp$  do
7     if  $T$  is not closed then
8        $make\_closed(T)$ 
9     if  $T$  is not consistent then
10       $make\_consistent(T)$ 
11     if  $T$  is not evidence-closed then
12       $make\_evidence\_closed(T)$ 
13    $prepared \leftarrow is\_prepared(T)$ ;
14    $M \leftarrow build\_DFA(T)$ ; // transforming  $T$  to a DFA  $M$ 
15    $\mathcal{H} \leftarrow build\_hypothesis(M)$ ; // constructing a hypothesis  $\mathcal{H}$  from  $M$ 
16    $equivalent, ctx \leftarrow equivalence\_query(\mathcal{H})$ ;
17   if  $equivalent = \perp$  then
18      $ctx\_processing(T, ctx)$ ; // counterexample processing
19 return  $\mathcal{H}$ ;
```

observation table T , where $S = \{\epsilon\}$, $E = \{\epsilon\}$. For each $\sigma \in \Sigma$, the learner obtains the corresponding reset-clocked word $\pi(\gamma)$ and the acceptance status in \mathcal{A} of the clocked word $\gamma = (\sigma, \{0\}^{|C|})$ through membership queries and adds the results to R . Before constructing a hypothesis, the learner performs several rounds of operations described in Section 4.1 until T is prepared. Then, a hypothesis \mathcal{H} is constructed leveraging an intermediate DFA M and submitted to the teacher for an equivalence query. If the answer is positive, \mathcal{H} recognizes the target language. Otherwise, the learner receives a counterexample ctx to update T . The whole procedure repeats until the teacher gives a positive answer to an equivalence query.

THEOREM 4.12. *Algorithm 2 terminates and returns a CTA \mathcal{H} which recognizes the target timed language \mathcal{L} .*

4.5 Complexity

Suppose that \mathcal{L} is a target timed language accepted by a DTA \mathcal{A} , let $n = |L|$ represent the number of locations of \mathcal{A} , $m = |\Sigma|$ represent the size of the alphabet, κ be a function mapping each clock $c \in C$ to the largest integer appearing in the guards over c and $\Lambda = |C|! \cdot 2^{|C|} \cdot \prod_{c \in C} (2\kappa(c) + 2)$ represent the bound of the number of clock regions. We measure the number of queries.

Equivalence query. According to the proof of Theorem 4.12, \mathcal{H} has at most $n \cdot \Lambda$ locations. The number of transitions from a location for an action is at most Λ . Therefore, the number of transitions in \mathcal{H} is at most $mn\Lambda^2$. As every counterexample adds at least one fresh transition to \mathcal{H} if considering each final timing constraint of the partition corresponds to a transition, the number of equivalence queries is bounded by $mn\Lambda^2$.

Membership query. In order to count the number of membership queries, $(|S| + |R|) \cdot |E|$ needs to be counted. Since in the observation table, E is used to distinguish the elements in S , we have $|E| \leq n \cdot \Lambda$. Next, $|R|$ is analyzed as follows. Let h denote the maximum length of

all counterexamples obtained by the learner. The algorithm adds elements to R in three cases: processing of counterexamples, making T closed, and making T evidence-closed. In the first case, because the number of counterexamples equals to the number of equivalence queries, which is bounded by $mn\Lambda^2$, the number of prefixes added to R is at most $hmn\Lambda^2$ in this case. In the second case, since for each prefix in S and each action, there is a corresponding prefix in R , the number of prefixes added to R in this step is at most $mn \cdot \Lambda$. In the last case, the number of prefixes added to R depends on $|S| \cdot |E|$, which is at most $(n \cdot \Lambda)^2$. Therefore, the number of membership queries is bounded by $(n \cdot \Lambda + hmn\Lambda^2 + mn \cdot \Lambda + (n \cdot \Lambda)^2) \cdot n \cdot \Lambda$.

Reset information query. In each membership query, the number of reset information queries depends on the length of the suffix. Let p be the length of the longest suffix in E . During the learning process, each new suffix added to E is composed of a region word with length one and a suffix that has already been in E . Therefore, p is at most $|E| - 1$. Hence, the number of reset information queries to fill the observation table is bounded by $(|E| - 1) \cdot (|S| + |R|) \cdot |E|$. Moreover, to make the observation table closed, for each $s \in S$, m reset information queries are needed. As a result, the number in total is bounded by $(|E| - 1) \cdot (|S| + |R|) \cdot |E| + m \cdot |S|$.

5 LEARNING DTA FROM A NORMAL TEACHER

In this section, we extend the learning algorithm for learning a DTA \mathcal{A} from a normal teacher. In this situation, the teacher can only answer the standard membership query and equivalence query. In a standard membership query, the input is delay-timed words, and the teacher returns whether the timed word is in the target timed language or not. In a standard equivalence query, if $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{H})$, the teacher provides a delay-timed word as a counterexample.

The algorithm in the normal teacher situation is based on the learning procedure in Section 4. In this algorithm, the structure of the observation table is unchanged. $S \cup R$ still stores reset-clocked words, and E still stores region words. In order to obtain reset-clocked words from delay-timed words of the membership queries, the learner *guesses* clock reset information in the table.

Since guessing the reset information, for each reset case, it will result in a table instance. A list *ToExplore* is maintained to collect all these table instances that need to be explored later. We use breadth-first search (BFS) to explore the tables in *ToExplore*, which is crucial for ensuring the termination. Since the algorithm cannot detect incorrect guesses of reset information, depth-first search (DFS) may not terminate if it follow a branch that never finds the correct hypothesis. Moreover, our BFS is not based on a strictly FIFO queue. In each round, we take out the table instance with the minimal number of guessed reset. Since the table instance contains reset information, the operations on the table instance are the same as those in the powerful teacher situation. Details are as follows.

The initial tables in *ToExplore* are as follows. For each action $\sigma \in \Sigma$, there is one row in R corresponding to the clocked word $\gamma = (\sigma, \{0\}^{|C|})$. To transform it to a reset-clocked word, the learner needs to guess the reset information of each clock in γ , get a reset-clocked word $\gamma_r = (\sigma, \{0\}^{|C|}, \mathbf{b})$ and add γ_r into R . For each action σ , there are $2^{|C|}$ possible combinations of guesses. So, in this step, $2^{|C|} \cdot |\Sigma|$ table instances are added to *ToExplore*.

In each iteration of the algorithm, one table instance is taken out of *ToExplore*. The learner checks whether the table is closed, consistent, and evidence-closed. If the table is not closed, i.e. there exists $r \in R$ such that $\text{row}(r) \neq \text{row}(s)$ for all $s \in S$. In this case, the learner moves r from R to S . Then for each $\sigma \in \Sigma$, a word $\gamma = vw(r) \cdot (\sigma, \{0\}^{|C|})$ is created, and the learner needs to guess the reset information of $(\sigma, \{0\}^{|C|})$. Similar to the initialization of the table, there are $2^{|C|}$ possible combinations of guesses for action σ . Therefore, $2^{|C||\Sigma|}$ unfilled table instances will be generated. Next, these $2^{|C||\Sigma|}$ unfilled table instances are filled by membership queries. Because the learner cannot initiate the reset information query, for each added entry in R , it is necessary to guess reset information for all transitions in $e \in E$. For an element $e \in E$, the reset information of $|e| \cdot |C|$ clocks needs to be guessed. Therefore, for each new row in R , a total of $(\sum_{e_i \in E \setminus \{e\}} |e_i|) \cdot |C|$ clock reset information needs to be guessed. Therefore, this procedure inserts at most $2^{(\sum_{e_i \in E \setminus \{e\}} |e_i|) \cdot |C|^2 \cdot |\Sigma|}$ table instances to *ToExplore*.

If the table is not consistent, i.e. there exist two reset-clocked words $\gamma_r, \gamma_r' \in S \cup R$ at least, for $\sigma_r, \sigma_r' \in \Sigma_r$ such that $\llbracket vw(\sigma_r) \rrbracket = \llbracket vw(\sigma_r') \rrbracket$, both $\gamma_r \sigma_r$ and $\gamma_r' \sigma_r' \in S \cup R$. However, $\text{row}(\gamma_r) = \text{row}(\gamma_r')$ but $\text{row}(\gamma_r \sigma_r) \neq \text{row}(\gamma_r' \sigma_r')$ or $\text{resets}(\sigma_r) \neq \text{resets}(\sigma_r')$. Let $e \in E$ be one place where they are different. Then $\llbracket vw(\sigma_r) \rrbracket \cdot e$ needs to be added to E . For each entry in $S \cup R$, all transitions in $\llbracket vw(\sigma_r) \rrbracket \cdot e$ need to be guessed, and then the table can be filled. Hence, $2^{\llbracket vw(\sigma_r) \rrbracket \cdot e \cdot |C| \cdot (|S| + |R|)}$ filled table instances will be generated and inserted into *ToExplore*. The operation for making tables evidence-closed is analogous.

If the current table is prepared, the learner builds a hypothesis \mathcal{H} and makes an equivalence query to the teacher. If the answer is negative, the teacher gives a delay-timed word ω as a counterexample. The learner first finds the longest reset-clocked word γ_r in $S \cup R$, which agrees with a prefix of ω after converting γ_r to a delay-timed word. Then the reset information of γ_r is copied to ω , and the reset information of the remainder of ω needs to be guessed. Hence, at most $2^{|\omega| \cdot |C|}$ unfilled table instances are generated. To fill these table instances, the reset information of E also needs to be guessed. Therefore, there are at most $2^{|\omega| \cdot (\sum_{e_i \in E \setminus \{e\}} |e_i|) \cdot |C|^2}$ filled table instances need to be added in *ToExplore*.

THEOREM 5.1. *The algorithm learning DTA from a normal teacher terminates and returns a correct CTA \mathcal{H} .*

Complexity analysis. If $T = (\Sigma, \Sigma_G, \Sigma_r, S, R, E, f, g)$ is the final table corresponding to the correct hypothesis, the number of guessed resets in $S \cup R$ is $(|S| + |R|) \cdot |C|$, and the number of guessed resets for entries in each row of the table is $(\sum_{e_i \in E \setminus \{e\}} |e_i|) \cdot |C|$. Since we use BFS to explore the tables in *ToExplore* w.r.t. the size of the tables, a smaller table is always handled before any larger ones. Hence, before finding the final table T corresponding to the correct hypothesis, the number of the handled tables is bounded by $O(2^{(|S| + |R|) \cdot |C|^2 \cdot (\sum_{e_i \in E \setminus \{e\}} |e_i|)})$.

6 EXPERIMENTS

We have implemented the a prototype named DTAL in JAVA for our learning algorithms of deterministic timed automata with multiple clocks. Specially, we denote the implementations for the powerful and the normal teachers by DTAL_POWERFUL and by DTAL_NORMAL,

respectively. We compared our methods with LEARNNTA proposed in [36] for learning DTAs, and with LEARNRTA proposed in [6] for learning real-time automata (RTAs). These two algorithms are implemented in C++ and Python, respectively. Note that we omitted checking of evidence-closed condition in our experiments to avoid adding too many extra rows in R . The evaluations have been carried out on an Intel Core-i7 processor with 16GB RAM running 64-bit Ubuntu 16.04.

6.1 Evaluation on DTAL_POWERFUL

Experiment on RTAs. We first compared the performance of the learning algorithm DTAL_POWERFUL with the algorithm LEARNRTA proposed in [6] by learning real-time automata (RTAs). Since RTA can be viewed as a TA with only one clock which resets at every transition, we can directly use DTAL_POWERFUL to learn real-time automata instead of DTAL_NORMAL by trivially answering reset information queries. We used the 240 randomly generated RTAs from [6], which are divided into 12 groups depending on different numbers of locations, size of alphabet, and maximum number of time intervals that can be generated by a single division. These three numbers make up case ID. Table 1 summarizes the experimental results. It can be seen that the number of equivalence queries needed in DTAL_POWERFUL is always very close to that in LEARNRTA. However, the number of membership queries needed in DTAL_POWERFUL is slightly smaller than that in LEARNRTA. This is because that we omitted checking of evidence-closed condition. Besides, the runtime needed in DTAL_POWERFUL is shorter than that in LEARNRTA. This may be due to the fact that these two algorithms are written in JAVA and Python, respectively.

Experiment on DTAs. We first compared the performance of our learning algorithm DTAL_POWERFUL with the algorithm LEARNNTA proposed in [36] on learning the DTA “unbalanced:2” in [36] as an example. The target automaton contains 5 locations, 1 action and 5 transitions. Our algorithm DTAL_POWERFUL yields a timed automaton with only 8 locations (including the “sink” location) and 2 clocks, whereas LEARNNTA yields a timed automaton with 25 locations and 7 clocks. It gives an evidence that our algorithm can return a much smaller DTA with fewer number of clocks. Since when learning a DTA by using the algorithm DTAL_NORMAL, the final table will never be larger than the final table in the powerful teacher situation. Therefore, the experimental result gives us highly confidence that DTAL_NORMAL will return a much smaller automaton than LEARNNTA once it terminates.

6.2 Evaluation on DTAL_NORMAL

Table 2 summarizes the experimental results. It shows that, as long as the maximum constant $\kappa(C)$ increases, the number of locations in the timed automata learned by LEARNNTA increases dramatically. This may be due to the fact that in the timed observation table of LEARNNTA, prefixes consist of a series of simple elementary languages. Hence, as the value of $\kappa(C)$ increases, the number of prefixes in the timed observation table increases significantly, thus affecting the number of locations of the hypothesis. Besides, in LEARNNTA, the number of the membership queries blows up exponentially to the value of $\kappa(C)$. This is because that when learning

Table 1: Experimental results on learning RTAs. #Reset, #Membership, #Equivalence represent the number of reset information queries, membership queries and equivalence queries needed during the learning process, respectively. n is the number of locations of the learned automaton.

| Case ID | Algorithm | #Reset | #Membership | #Equivalence | n | time(s) |
|---------|---------------|--------|-------------|--------------|-----|---------|
| 4-4-4 | DTAL_POWERFUL | 228.0 | 267.6 | 28.7 | 5 | 2.9 |
| | LEARNTA | — | 298.4 | 28.3 | 5 | 4.5 |
| 6-4-4 | DTAL_POWERFUL | 529.3 | 578.8 | 46.8 | 7 | 16.4 |
| | LEARNTA | — | 696.0 | 45.6 | 7 | 37.1 |
| 7-2-4 | DTAL_POWERFUL | 388.2 | 383.3 | 32.6 | 8 | 3.5 |
| | LEARNTA | — | 554.0 | 31.2 | 8 | 9.0 |
| 7-3-4 | DTAL_POWERFUL | 458.2 | 456.9 | 37.6 | 8 | 6.5 |
| | LEARNTA | — | 630.4 | 36.7 | 8 | 17.0 |
| 7-4-2 | DTAL_POWERFUL | 469.9 | 436.9 | 23.6 | 8 | 2.9 |
| | LEARNTA | — | 630.0 | 22.8 | 8 | 11.0 |
| 7-4-3 | DTAL_POWERFUL | 470.9 | 496.6 | 32.0 | 8 | 5.8 |
| | LEARNTA | — | 680.0 | 31.4 | 8 | 16.1 |
| 7-4-4 | DTAL_POWERFUL | 529.4 | 571.5 | 46.6 | 8 | 14.0 |
| | LEARNTA | — | 793.6 | 45.2 | 8 | 34.7 |
| 7-4-5 | DTAL_POWERFUL | 613.6 | 639.3 | 50.0 | 8 | 19.7 |
| | LEARNTA | — | 831.3 | 48.5 | 8 | 46.0 |
| 7-4-6 | DTAL_POWERFUL | 907.8 | 986.4 | 85.2 | 8 | 110.3 |
| | LEARNTA | — | 1149.9 | 83.4 | 8 | 142.5 |
| 7-4-7 | DTAL_POWERFUL | 1029.8 | 1143.1 | 95.0 | 8 | 176.1 |
| | LEARNTA | — | 1304.3 | 93.1 | 8 | 219.9 |
| 8-4-4 | DTAL_POWERFUL | 804.0 | 841.4 | 56.3 | 9 | 33.9 |
| | LEARNTA | — | 1156.5 | 54.3 | 9 | 55.7 |
| 10-4-4 | DTAL_POWERFUL | 1268.2 | 1291.1 | 70.2 | 11 | 94.1 |
| | LEARNTA | — | 1843.3 | 67.9 | 11 | 167.7 |

Table 2: Experimental results on learning DTAs. Each row represents one DTA where $|L|$ is the number of locations, $|\Sigma|$ is the alphabet size, $|C|$ is the number of clock variables, and $\kappa(C)$ is the maximum constant in the guards in the DTA. n is the number of locations of the learned automaton. — represents a timeout (1.5 hours).

| $ L $ | $ \Sigma $ | $ C $ | $\kappa(C)$ | Algorithm | #Membership | #Equivalence | n | time(s) |
|-------|------------|-------|-------------|-------------|-------------|--------------|-----|---------|
| 2 | 1 | 2 | 3 | DTAL_NORMAL | 763145 | 272734 | 3 | 2776 |
| | | | | LEARNTA | 1189 | 6 | 3 | 0.035 |
| 2 | 1 | 2 | 30 | DTAL_NORMAL | 891513 | 320108 | 3 | 4806 |
| | | | | LEARNTA | 793983 | 33 | 29 | 631 |
| 3 | 1 | 2 | 3 | DTAL_NORMAL | 1019356 | 101936 | 3 | 1356 |
| | | | | LEARNTA | 1251 | 7 | 4 | 0.009 |
| 3 | 1 | 2 | 30 | DTAL_NORMAL | 913890 | 122534 | 3 | 2457 |
| | | | | LEARNTA | 1874410 | 49 | 39 | 475 |
| 4 | 1 | 2 | 50 | DTAL_NORMAL | — | — | — | — |
| | | | | LEARNTA | — | — | — | — |

DTA from a normal teacher, symbolic membership queries is forbidden, a symbolic membership query needs to be reduced to finitely many membership queries. However, the size of this decomposition exponentially blows up with respect to $\kappa(C)$. Similarly, in LEARN TA, the number of equivalence queries also grows as $\kappa(C)$ increases.

The experimental results show that DTAL in the normal teacher situation requires more membership queries and equivalence queries. This is because when guessing the reset information of the target DTA, a large number of observation tables are generated, and more membership queries and equivalence queries are needed to maintain these observation tables. But it can return a much smaller automaton, which benefits from the equivalence relation presented in Section 3. The results also show that both algorithms, i.e., DTAL and LEARN TA, need improvements before being applied in practice.

7 RELATED WORK

Active learning. In [15, 16], Grinchtein et al. propose learning algorithms for deterministic event-recording automata (ERA) [3]. ERA is a kind of timed automata with several restrictions on clocks such that for every action a , a clock x_a is associated to record the time length from the last occurrence of a to the current. Recently, in [17], Henry et al. consider the learning problem of reset-free ERA in which some transitions may reset no clocks. In [5], An et al. provide an active learning method for deterministic one-clock timed automata (DOTA). After that, several improvements have been proposed, e.g., using PAC learning [25], mutation testing [27], SMT solving [37], etc. Real-time automata can be viewed as a kind of DOTA such that the single clock resets at every transition. The efficient learning algorithms have been designed in both the deterministic case [6] and the nondeterministic case [7]. Instead of using clocks, the “timer” is another kind of model to record the time length. It can be assigned a value, and if it counts down to zero, a visible timeout signal will be triggered. In [10], Caldwell et al. proposed an algorithm for learning the Mealy machine with timers from programmable logic controllers. In [31], Vaandrager et al. presented an efficient learning algorithm for such models with one timer. Recently, an active learning algorithm of DTA with Myhill-Nerode style characterization is proposed in [36]. However, with a large maximum constant in the guards in the target DTA, the number of queries required during learning and the number of locations in the learned DTA grows significantly in this algorithm. Our algorithm can return a much smaller DTA.

Passive learning. In [32, 35], an algorithm is proposed to learn deterministic RTA [12] from labelled time-stamped event sequences. A passive learning algorithm for timed automata with one clock is further proposed in [33, 34]. In [28], Tappler et al. present a passive learning algorithm for deterministic timed automata based on genetic programming (GP). In further, by introducing a testing process on the system under test (SUT), Aichernig et al. add an active manner in the GP-based learning procedure [1]. Recently, in [29], Tappler et al. present an SMT-based passive learning algorithm for deterministic timed automata. There are also some works on passive learning of timed automata in practical cases [11, 22] and dynamical systems [19]. As mentioned before, passive learning algorithms cannot guarantee the correctness of the learned model.

8 CONCLUSION AND FUTURE WORK

We propose an active learning approach for deterministic multi-clock timed automata. We transform the problem into learning the reset-clocked language of the target automaton. We first introduce an equivalence relation for reset-clocked languages. Based on it, we present a learning algorithm for deterministic timed automata in the powerful teacher situation by designing the membership query and the reset information query over region words. In further, we drop the reset information query and present the active learning algorithm in the normal teacher situation where the learner can only make standard membership queries and equivalence queries.

In future work, we intend to improve the algorithm by using the technique reported in [37] so that it can be used in learning timed models with large sizes of practical systems.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback and suggestions for improving the quality of this paper. Y. Teng and M. Zhang are supported by Shanghai 2023 “Science and Technology Innovation Action Plan”: Special Project for Key Technical Breakthrough of Blockchain (No. 23511100800), and the National Natural Science Foundation of China (No. 61972284, 62032019). J. An is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

REFERENCES

- [1] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. 2020. From Passive to Active: Learning Timed Automata Efficiently. In *Proceedings of the 12th International Symposium on NASA Formal Methods, NFM 2020 (LNCS, Vol. 12229)*. Springer, 1–19. https://doi.org/10.1007/978-3-030-55754-6_1
- [2] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235. [https://doi.org/10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8)
- [3] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. 1999. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theor. Comput. Sci.* 211, 1–2 (1999), 253–273. [https://doi.org/10.1016/S0304-3975\(97\)00173-4](https://doi.org/10.1016/S0304-3975(97)00173-4)
- [4] Rajeev Alur, P. Madhusudan, and Wonhong Nam. 2005. Symbolic Compositional Verification by Learning Assumptions. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005 (LNCS, Vol. 3576)*. Springer, 548–562. https://doi.org/10.1007/11513988_52
- [5] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2020. Learning One-Clock Timed Automata. In *26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2020 (LNCS, Vol. 12078)*. Springer, 444–462. https://doi.org/10.1007/978-3-030-45190-5_25
- [6] Jie An, Lingtai Wang, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2021. Learning real-time automata. *Sci. China Inf. Sci.* 64, 9 (2021). <https://doi.org/10.1007/s11432-019-2767-4>
- [7] Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2021. Learning Nondeterministic Real-Time Automata. *ACM Trans. Embed. Comput. Syst.* 20, 5s (2021), 99:1–99:26. <https://doi.org/10.1145/3477030>
- [8] Dana Angluin. 1978. On the Complexity of Minimum Inference of Regular Sets. *Inf. Control* 39, 3 (1978), 337–350. [https://doi.org/10.1016/S0019-9958\(78\)90683-6](https://doi.org/10.1016/S0019-9958(78)90683-6)
- [9] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106. [https://doi.org/10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6)
- [10] Ben Caldwell, Rachel Cardell-Oliver, and Tim French. 2016. Learning Time Delay Mealy Machines From Programmable Logic Controllers. *IEEE Trans. Autom. Sci. Eng.* 13, 2 (2016), 1155–1164. <https://doi.org/10.1109/TASE.2015.2496242>
- [11] L  naig Cornauguer, Christine Largou  t, Laurence Roz  , and Alexandre Termier. 2022. TAG: Learning Timed Automata from Logs. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*. AAAI Press, 3949–3958. <https://ojs.aaai.org/index.php/AAAI/article/view/20311>
- [12] Catalin Dima. 2001. Real-Time Automata. *J. Autom. Lang. Comb.* 6, 1 (2001), 3–23. <https://doi.org/10.25596/jalc-2001-003>
- [13] Paul Fiterau-Brosteau, Ramon Janssen, and Frits W. Vaandrager. 2016. Combining Model Learning and Model Checking to Analyze TCP Implementations. In *Proceedings of the 28th International Conference on Computer Aided Verification, CAV 2016 (LNCS, Vol. 9780)*. Springer, 454–471. https://doi.org/10.1007/978-3-319-41540-6_25
- [14] E. Mark Gold. 1978. Complexity of Automaton Identification from Given Data. *Inf. Control* 37, 3 (1978), 302–320. [https://doi.org/10.1016/S0019-9958\(78\)90562-4](https://doi.org/10.1016/S0019-9958(78)90562-4)
- [15] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. 2010. Learning of event-recording automata. *Theor. Comput. Sci.* 411, 47 (2010), 4029–4054. <https://doi.org/10.1016/j.tcs.2010.07.008>
- [16] Olga Grinchtein, Bengt Jonsson, and Paul Pettersson. 2006. Inference of Event-Recording Automata Using Timed Decision Trees. In *Proceedings of the 17th International Conference on Concurrency Theory, CONCUR 2006 (LNCS, Vol. 4137)*. Springer, 435–449. https://doi.org/10.1007/11817949_29
- [17] L  o Henry, Thierry J  ron, and Nicolas Markey. 2020. Active Learning of Timed Automata with Unobservable Resets. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2020 (LNCS, Vol. 12288)*. Springer, 144–160. https://doi.org/10.1007/978-3-030-57628-8_9
- [18] John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.
- [19] Xiangyu Jin, Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2021. Inferring Switched Nonlinear Dynamical Systems. *Formal Aspects Comput.* 33, 3 (2021), 385–406. <https://doi.org/10.1007/S00165-021-00542-7>
- [20] Michael J. Kearns and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. MIT Press.
- [21] Martin Leucker. 2006. Learning Meets Verification. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects, FMCO 2006 (LNCS, Vol. 4709)*. Springer, 127–151. https://doi.org/10.1007/978-3-540-74792-5_6
- [22] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. 2017. Timed k-Tail: Automatic Inference of Timed Automata. In *Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017*. IEEE Computer Society, 401–411. <https://doi.org/10.1109/ICST.2017.43>
- [23] Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. 2002. Black Box Checking. *Journal of Automata, Languages and Combinatorics* 7, 2 (2002), 225–246. <https://doi.org/10.25596/jalc-2002-225>
- [24] Mathijs Schuts, Jozef Hooman, and Frits W. Vaandrager. 2016. Refactoring of Legacy Software Using Model Learning and Equivalence Checking: An Industrial Experience Report. In *Proceedings of the 12th International Conference on Integrated Formal Methods, IFM 2016 (LNCS, Vol. 9681)*. Springer, 311–325. https://doi.org/10.1007/978-3-319-33693-0_20
- [25] Wei Shen, Jie An, Bohua Zhan, Miaomiao Zhang, Bai Xue, and Naijun Zhan. 2020. PAC Learning of Deterministic One-Clock Timed Automata. In *Proceedings of the 22nd International Conference on Formal Engineering Methods, ICFEM 2020 (LNCS, Vol. 12531)*. Springer, 129–146. https://doi.org/10.1007/978-3-030-63406-3_8
- [26] Bernhard Steffen, Falk Howar, and Maik Merten. 2011. Introduction to Active Automata Learning from a Practical Perspective. In *Proceedings of the 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems - Formal Methods for Eternal Networked Software Systems, SFM 2011 (LNCS, Vol. 6659)*. Springer, 256–296. https://doi.org/10.1007/978-3-642-21455-4_8
- [27] Xiaochen Tang, Wei Shen, Miaomiao Zhang, Jie An, Bohua Zhan, and Naijun Zhan. 2022. Learning Deterministic One-Clock Timed Automata via Mutation Testing. In *Proceedings of the 20th International Symposium on Automated Technology for Verification and Analysis, ATVA 2022 (LNCS, Vol. 13505)*. Springer, 233–248. https://doi.org/10.1007/978-3-031-19992-9_15
- [28] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. 2019. Time to Learn - Learning Timed Automata from Tests. In *Proceedings of the 17th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2019 (LNCS, Vol. 11750)*. Springer, 216–235. https://doi.org/10.1007/978-3-030-29662-9_13
- [29] Martin Tappler, Bernhard K. Aichernig, and Florian Lorber. 2022. Timed Automata Learning via SMT Solving. In *Proceedings of the 14th International Symposium on NASA Formal Methods , NFM 2022 (LNCS, Vol. 13260)*. Springer, 489–507. https://doi.org/10.1007/978-3-031-06773-0_26
- [30] Frits W. Vaandrager. 2017. Model learning. *Commun. ACM* 60, 2 (2017), 86–95. <https://doi.org/10.1145/2967606>
- [31] Frits W. Vaandrager, Roderick Bloem, and Masoud Ebrahimi. 2021. Learning Mealy Machines with One Timer. In *Proceedings of the 15th International Conference Language and Automata Theory and Applications, LATA 2021 (LNCS, Vol. 12638)*. Springer, 157–170. https://doi.org/10.1007/978-3-030-68195-1_13
- [32] SE Verwer, MM De Weerd, and Cees Witteveen. 2007. An algorithm for learning real-time automata. In *Benelearn 2007: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands, Amsterdam, The Netherlands, 14-15 May 2007*.
- [33] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. 2009. One-Clock Deterministic Timed Automata Are Efficiently Identifiable in the Limit. In *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5457)*. Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Mart  n-Vide (Eds.). Springer, 740–751. https://doi.org/10.1007/978-3-642-00982-2_63
- [34] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. 2011. The efficiency of identifying timed automata and the power of clocks. *Inf. Comput.* 209, 3 (2011), 606–625. <https://doi.org/10.1016/j.ic.2010.11.023>
- [35] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. 2012. Efficiently identifying deterministic real-time automata from labeled data. *Mach. Learn.* 86, 3 (2012), 295–333. <https://doi.org/10.1007/s10994-011-5265-4>
- [36] Masaki Waga. 2023. Active Learning of Deterministic Timed Automata with Myhill-Nerode Style Characterization. In *CAV 2023 (LNCS, Vol. 13964)*. Springer, 3–26. https://doi.org/10.1007/978-3-031-37706-8_1
- [37] Runqing Xu, Jie An, and Bohua Zhan. 2022. Active Learning of One-Clock Timed Automata Using Constraint Solving. In *Proceedings of the 20th International Symposium on Automated Technology for Verification and Analysis, ATVA 2022 (LNCS, Vol. 13505)*. Springer, 249–265. https://doi.org/10.1007/978-3-031-19992-9_16