SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2025, Vol. 68, Iss. 12, 222101:1–222101:17
https://doi.org/10.1007/s11432-024-4393-8

# Active learning of deterministic timed automata via timed classification tree

Yu TENG[1], Hanyue CHEN[1], Junri MI[1], Miaomiao ZHANG[1*], Jie AN[2] & Naijun ZHAN[3]

[1]*School of Computer Science and Technology, Tongji University, Shanghai 200092, China*
[2]*National Key Laboratory of Space Integrated Information System, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*
[3]*Key Laboratory of High Confidence Software Technologies (PKU), MOE, School of Computer Science, Peking University, Beijing 100871, China*

Received 2 August 2024/Revised 14 March 2025/Accepted 17 April 2025/Published online 15 September 2025

**Abstract** Active learning of timed automata is a crucial research topic. While a recent algorithm has been proposed for active learning of deterministic timed automata, it exhibits slow learning speed, particularly for automata with large sizes, posing challenges for application in complex systems. In this paper, we propose an enhanced learning algorithm, leveraging the idea of tree-based active learning. Our approach utilizes a data structure named timed classification tree, instead of an observation table, to store information acquired during the learning process. By avoiding checking the tree status and processing useless prefixes of counterexamples, it reduces the number of membership queries and equivalence queries, thereby also accelerating the practical learning speed. The experimental results demonstrate the effectiveness of our approach.

**Keywords** model learning, active learning, timed automata, timed classification tree, counterexample processing

**Citation** Teng Y, Chen H Y, Mi J R, et al. Active learning of deterministic timed automata via timed classification tree. Sci China Inf Sci, 2025, 68(12): 222101, https://doi.org/10.1007/s11432-024-4393-8

## 1 Introduction

Model learning [1] serves as a method for generating a formal model of a system from its input-output behaviors, typically classified into two types: passive learning and active learning. Passive learning aims to deduce a consistent model from a given set of samples [2], a topic extensively explored across various models [3–14]. However, it does not guarantee the acquisition of a fully correct formal model. In contrast, active learning furnishes a mechanism for a learner to choose sample sets. Angluin introduced the well-known $L^*$ algorithm within the active learning framework MAT, where the learner can make membership and equivalence queries to a teacher [15]. In a membership query, the learner asks whether a 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 exactly. The learning process iterates until it yields an exact model. Following this framework, active learning algorithms for many kinds of automata have been proposed, such as Mealy machines [16, 17], nondeterministic finite-state automata [18], register automata [19–21], I/O automata [22], and symbolic automata [23, 24].

Active learning for timed models has gained notable focus lately. For instance, Vaandrager et al. introduced an active learning algorithm for Mealy machines with a single timer [25]. For real-time automata, active learning algorithms have been proposed for both deterministic [26] and nondeterministic cases [27]. Furthermore, active learning of deterministic one-clock timed automata has been outlined [28]. Subsequent advancements include techniques such as PAC learning [29], mutation testing [30], and SMT solving [31], among others, aimed at enhancing effectiveness. It is noteworthy that these algorithms are limited in the ability to learn timed automata with multiple clocks. Recently, an active learning algorithm for learning deterministic timed automata (DTAs) with multiple clocks was introduced in [32]. However, the number of locations in the learned automata significantly increased compared to the target automata.

* Corresponding author (email: miaomiao@tongji.edu.cn)

To yield deterministic timed automata with smaller sizes, a novel learning algorithm was proposed in [33]. The authors first considered learning from a powerful teacher who can answer reset information queries additionally. Then, the learning is performed with a normal teacher, with the learner guessing the possible clock resets. Therefore, each resulting situation after guessing can be viewed as learning from a powerful teacher. However, its current limitation is that it can only learn very small-scale systems, which poses a challenge to its application to complex systems. In this paper, we address the following primary drawbacks associated with learning from a powerful teacher via the observation table [33].

**Additional overheads to issue a valid membership query.** Before making a membership query for a timed word in the table, it is essential to construct one of its valid successors first, which needs auxiliary time. Further details are presented in Section 2.

**Useless prefixes of counterexamples.** During counterexample processing, all prefixes of a counterexample are added to the observation table. However, most of these prefixes are useless for refining the hypothesis, resulting in redundant membership queries during the subsequent learning process.

**Time-consuming table operations and hypothesis construction.** The processes of checking whether the observation table is closed or consistent are time-consuming.

To tackle these challenges, we are inspired by tree-based learning algorithms for different kinds of automata. In [34], Isberner et al. proposed a learning algorithm for DFAs based on discrimination trees. After that, a method for learning Buchi automata using classification trees is proposed [35]. Vaandrager et al. also learned Mealy machines based on observation trees [36]. Recently, Dierl et al. proposed a tree-based learning algorithm for register automata [37]. However, their methods are not applicable to timed automata. In [38], to learn real-time automata, both the structure and operation of the real-time classification tree proposed are relatively simple. In [39], to learning DTAs with one clock, the one-clock classification tree is proposed. However, the one-clock classification tree can only deal with only one clock and fails to ensure the termination of the learning algorithm for a DTA with multiple clocks. Compared to these studies, we define a novel data structure termed the timed classification tree which can handle multiple clocks. This structure replaces the traditional observation table and efficiently stores query results. Leveraging this data structure, we present an improved active learning algorithm for DTAs.

Our main contributions are summarized as follows.

• Timed classification tree and the operations on it. Utilization of a timed classification tree instead of an observation table can reduce the number of membership queries, thereby reducing overall additional overheads. Furthermore, the structured nature of the tree, coupled with the method for hypothesis construction, enables the learner to construct a hypothesis without checking the tree status. These advancements effectively mitigate the first and third drawbacks mentioned above.

• Methods for counterexample analysis and processing. We propose a method to pinpoint an erroneous position within a counterexample, leading to the first conflict between the current hypothesis $\mathcal{H}$ and the target DTA $\mathcal{A}$. Utilizing the information of the erroneous position to refine the classification tree will indeed generate a modification of $\mathcal{H}$. This approach effectively reduces the membership queries for the useless prefixes of a counterexample that do not contribute to a modification of $\mathcal{H}$, thereby resolving the second drawback.

• We implemented our algorithm and compared it with the algorithm in [33]. The results show that our method can significantly reduce the number of membership queries and improve the efficiency.

## 2 Preliminaries

Let $\mathbb{R}_{\geqslant 0}$ be the set of non-negative real numbers, $\mathbb{N}$ be the set of natural numbers, and $\mathbb{B} = \{\top, \bot\}$ be the Boolean set, where $\top$ is true and $\bot$ is false. Let $\Sigma$, named alphabet, be a finite set of events or actions.

### 2.1 Timed words, timed languages, and timed automata

A timed word is a finite sequence $\omega = (\sigma_1, t_1)(\sigma_2, t_2)\cdots(\sigma_n, t_n) \in (\Sigma \times \mathbb{R}_{\geqslant 0})^*$, where $t_i$ represents the delay time length before taking action $\sigma_i$ for all $1 \leqslant i \leqslant n$. A timed language $\mathcal{L}$ is a set of timed words.

Timed automata (TAs) [40] extend finite-state automata with a finite set of clock variables. Let $\mathcal{C}$ be the set of clock variables.

A clock constraint $\phi$ is a conjunctive formula of atomic constraints of the form $c \sim n$, for $c \in \mathcal{C}$ and $n \in \mathbb{N}$, where $\sim \in \{\leqslant, <, \geqslant, >, =\}$. Let $\Phi(\mathcal{C})$ be the set of clock constraints.

A clock valuation $\nu : \mathcal{C} \to \mathbb{R}_{\geqslant 0}$ is a function assigning a non-negative real value to each clock. Given a clock valuation $\nu$ for $\mathcal{C}$ and a clock constraint $\phi$ over $\mathcal{C}$, if $\phi$ evaluates to true using the values given by $\nu$, then $\nu \in \phi$, i.e., $\nu$ satisfies $\phi$. For $d \in \mathbb{R}_{\geqslant 0}$, $\nu + d$ denotes the clock valuation which maps each clock $c \in \mathcal{C}$ to the value $\nu(c) + d$. For a set $\mathcal{B} \subseteq \mathcal{C}$, $[\mathcal{B} \to 0]\nu$ is the clock valuation which resets each $c \in \mathcal{B}$ to 0, and agrees with $\nu$ for each $c \in \mathcal{C}\backslash\mathcal{B}$.

**Definition 1** (Timed automata [40])**.** A timed automaton (TA) is a tuple $\mathcal{A} = (\Sigma, L, l_0, F, \mathcal{C}, \Delta)$, where $\Sigma$ 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, $\mathcal{C}$ is the set of clocks, and $\Delta \subseteq L \times \Sigma \times \Phi(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is a finite set of transitions.

A transition $\delta = (l, \sigma, \phi, \mathcal{B}, l') \in \Delta$ represents a jump in $\mathcal{A}$ from the source location $l$ to the target location $l'$ by performing the action $\sigma$ when the current clock valuation satisfies the constraint $\phi$. The set $\mathcal{B} \subseteq \mathcal{C}$ gives the clocks to be reset in the transition.

A state $s$ of $\mathcal{A}$ is a pair $(l, \nu)$, where $l \in L$ is a location and $\nu$ is a clock valuation. Given a timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$, a run $\rho$ of $\mathcal{A}$ over $\omega$ is a sequence $\rho = (l_0, \nu_0) \xrightarrow{t_1,\sigma_1} (l_1, \nu_1) \xrightarrow{t_2,\sigma_2} \cdots \xrightarrow{t_n,\sigma_n} (l_n, \nu_n)$, satisfying the requirements: (1) $l_0$ is the initial location and $\nu_0(c) = 0$ for each clock $c \in \mathcal{C}$; (2) for all $1 \leqslant i \leqslant n$, there is a transition $(l_{i-1}, \sigma_i, \phi_i, \mathcal{B}_i, l_i)$ such that $(\nu_{i-1} + t_i) \in \phi_i$ and $\nu_i = [\mathcal{B}_i \to 0](\nu_{i-1} + t_i)$. For a run $\rho$ over a timed word $\omega$, if $\rho \neq (l_0, \nu_0)$, the *trace* of $\rho$ is defined as the corresponding timed word $\omega$, i.e., $\omega = \mathit{trace}(\rho)$. If $\rho = (l_0, \nu_0)$, $\mathit{trace}(\rho) = \epsilon$. By recording the reset information along the run $\rho$ over the timed word $\omega$, we get a reset-timed word corresponding to $\omega$, denoted by $\omega_r = \mathit{trace}_r(\rho) = (\sigma_1, t_1, \boldsymbol{b}_1)(\sigma_2, t_2, \boldsymbol{b}_2) \cdots (\sigma_n, t_n, \boldsymbol{b}_n)$, where $\boldsymbol{b}_i \in \mathbb{B}^{|\mathcal{C}|}$ is a $|\mathcal{C}|$-tuple. For all $c_j \in \mathcal{C}$ and $1 \leqslant j \leqslant |\mathcal{C}|$, $\boldsymbol{b}_{i,j}$ records when taking $(\sigma_i, t_i)$, whether the $j$-th clock $c_j$ is reset.

For a run $\rho = (l_0, \nu_0) \xrightarrow{t_1,\sigma_1} (l_1, \nu_1) \xrightarrow{t_2,\sigma_2} \cdots \xrightarrow{t_n,\sigma_n} (l_n, \nu_n)$, if $l_n \in F$, $\rho$ is an accepting run of $\mathcal{A}$. For a timed automaton $\mathcal{A}$, its (recognized) timed language $\mathcal{L}(\mathcal{A}) = \{\mathit{trace}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$. Two timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if and only if $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. Similarly, the (recognized) reset-timed language $\mathcal{L}_r(\mathcal{A}) = \{\mathit{trace}_r(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

**Clocked word and reset-clocked word.** A timed word can be considered a system behavior observed from the perspective of the global clock. To represent the system behavior observed from the perspective of the logical clock variables $\mathcal{C}$ within the system, a clocked word is defined based on a run $\rho = (l_0, \nu_0) \xrightarrow{t_1,\sigma_1} (l_1, \nu_1) \xrightarrow{t_2,\sigma_2} \cdots \xrightarrow{t_n,\sigma_n} (l_n, \nu_n)$. In particular, a clocked word is denoted by $\gamma = (\sigma_1, \boldsymbol{v}_1)\ (\sigma_2, \boldsymbol{v}_2) \cdots (\sigma_n, \boldsymbol{v}_n)$, where $\boldsymbol{v}_i \in \mathbb{R}_{\geqslant 0}^{|\mathcal{C}|}$ records the clock value for each clock when taking action $\sigma_i$, i.e. $\boldsymbol{v}_{i,j} = \nu_{i-1}(c_j) + t_i$ for all $c_j \in \mathcal{C}$ and $1 \leqslant j \leqslant |\mathcal{C}|$. Let $\boldsymbol{\Sigma} = \Sigma \times \mathbb{R}_{\geqslant 0}^{|\mathcal{C}|}$ be a infinite set of clocked actions. After extending the reset information along the run $\rho$ on $\gamma$, we get the reset-clocked word $\gamma_r = (\sigma_1, \boldsymbol{v}_1, \boldsymbol{b}_1)(\sigma_2, \boldsymbol{v}_2, \boldsymbol{b}_2) \cdots (\sigma_n, \boldsymbol{v}_n,\ \boldsymbol{b}_n)$. Let $vw(\gamma_r) = \gamma = (\sigma_1, \boldsymbol{v}_1)\ (\sigma_2, \boldsymbol{v}_2) \cdots (\sigma_n,\ \boldsymbol{v}_n)$ and $\mathit{resets}(\gamma_r) = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$ to extract the reset information of $\gamma_r$. Similarly, $\boldsymbol{\Sigma}_r = \Sigma \times \mathbb{R}_{\geqslant 0}^{|\mathcal{C}|} \times \mathbb{B}^{|\mathcal{C}|}$ is a infinite set of reset-clocked actions. Moreover, for a given reset-timed word $\omega_r = (\sigma_1, t_1, \boldsymbol{b}_1)(\sigma_2, t_2, \boldsymbol{b}_2) \cdots (\sigma_n, t_n, \boldsymbol{b}_n)$, we can compute the only corresponding clocked word $\gamma = (\sigma_1, \boldsymbol{v}_1)(\sigma_2, \boldsymbol{v}_2) \cdots\ (\sigma_n, \boldsymbol{v}_n)$, where for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant |\mathcal{C}|$, if $i = 1$ or $\boldsymbol{b}_{i-1,j} = \top$, $\boldsymbol{v}_{i,j} = t_i$, otherwise, $\boldsymbol{v}_{i,j} = \boldsymbol{v}_{i-1,j} + t_i$.

Let $\Gamma_\rho$ map a (reset-)timed word to the (reset-)clocked counterpart and $\pi_\rho$ map a clocked word $\gamma$ to its reset-clocked counterpart $\gamma_r$ according to a run $\rho$ respectively. When $\rho$ is determined, we use $\Gamma$ and $\pi$ directly. For a timed automaton $\mathcal{A}$, the recognized clocked language $\mathscr{L}(\mathcal{A}) = \{\Gamma_\rho(\mathit{trace}(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$ and the recognized reset-clocked language $\mathscr{L}_r(\mathcal{A}) = \{\Gamma_\rho(\mathit{trace}_r(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$. In a timed automaton $\mathcal{A}$, a clocked word $\gamma$ is valid for $\mathcal{A}$ if there exists a run $\rho$ satisfying $\Gamma_\rho(\mathit{trace}(\rho)) = \gamma$. Similarly, a reset-clocked word $\gamma_r$ is valid for $\mathcal{A}$ if there exists a run $\rho$ satisfying $\Gamma_\rho(\mathit{trace}_r(\rho)) = \gamma_r$. A reset-clocked word $\gamma_r$ is doomed if it is invalid for any timed automaton.

**Definition 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 $\rho$ for any timed word $\omega$.

In a DTA $\mathcal{A} = (\Sigma, L, l_0, F,\ \mathcal{C}, \Delta)$, 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 $\Delta$, the clock constraints $\phi_1$ and $\phi_2$ are mutually exclusive.

A timed automaton $\mathcal{A}$ is a complete deterministic timed automaton (CTA) if and only if there is exactly one run $\rho$ for any given timed word $\omega$. Every DTA can be transformed into a CTA. The transition method can be divided into three steps: (1) introduce a sink location which is not accepted; (2) for each $l \in L$ and $\sigma \in \Sigma$, if there is no transition from $l$ performing $\sigma$, add a transition resetting all clocks from $l$ to "sink" with label $\sigma$ and guards $c \geqslant 0$ for each $c \in \mathcal{C}$; (3) otherwise, let $\mathit{Compl}_{l,\sigma}$ be the subset of $\mathbb{R}_{\geqslant 0}^{|\mathcal{C}|}$ that is not covered by the guards of transitions from $l$ with label $\sigma$. Represent $\mathit{Compl}_{l,\sigma}$ as a union of

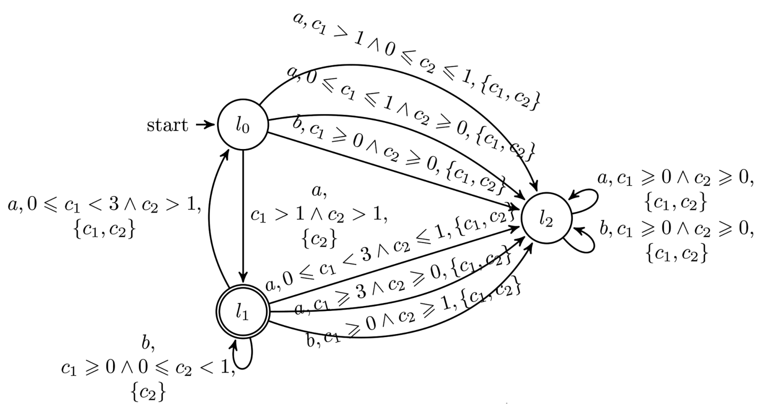

$a, c_1 > 1 \wedge 0 \leqslant c_2 \leqslant 1, \{c_1, c_2\}$
$a, 0 \leqslant c_1 \leqslant 1 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
start
$l_0$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$a, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$l_2$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$a, 0 \leqslant c_1 < 3 \wedge c_2 > 1, \{c_1, c_2\}$
$a, c_1 > 1 \wedge c_2 > 1, \{c_2\}$
$a, 0 \leqslant c_1 < 3 \wedge c_2 \leqslant 1, \{c_1, c_2\}$
$a, c_1 \geqslant 3 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$l_1$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 1, \{c_1, c_2\}$
$b, c_1 \geqslant 0 \wedge 0 \leqslant c_2 < 1, \{c_2\}$

**Figure 1** A CTA $\mathcal{A}$.

guards $I_1, \dots, I_k$ in the simplest way, then for each $1 \leqslant j \leqslant k$, add a transition resetting all clocks from $l$ to "sink" with label $\sigma$ and guard $I_j$. Figure 1 shows a CTA $\mathcal{A}$ with an initial location $l_0$, an accepting location $l_1$, a "sink" location $l_2$, and two clocks $c_1, c_2$. Since the number of clock valuations is infinite, the number of states is infinite. To form a finite partition of the state space, region is introduced.

**Definition 3** (Region equivalence [40]). Given a timed automaton $\mathcal{A}$, for each $c \in \mathcal{C}$, let $\kappa(c)$ represent the largest integer appearing in the constraints over $c$ in $\mathcal{A}$. For any real time $a$, let $\mathit{frac}(a)$ denote the fractional part of $a$ and $\lfloor a \rfloor$ denote the integral part of $a$. Based on these concepts, two clock valuations $\nu$ and $\nu'$ are region-equivalent, denoted by $\nu \sim \nu'$, iff all the following conditions hold:

- for all $c \in \mathcal{C}$, either $\lfloor \nu(c) \rfloor = \lfloor \nu'(c) \rfloor$ or both $\nu(c) > \kappa(c)$ and $\nu'(c) > \kappa(c)$;
- for all $c \in \mathcal{C}$, if $\nu(c) \leqslant \kappa(c)$ then $\mathit{frac}(\nu(c)) = 0$ iff $\mathit{frac}(\nu'(c)) = 0$; and
- for all $c_i, c_j \in \mathcal{C}$, if $\nu(c_i) \leqslant \kappa(c_i)$ and $\nu(c_j) \leqslant \kappa(c_j)$ then $\mathit{frac}(\nu(c_i)) \leqslant \mathit{frac}(\nu(c_j))$ iff $\mathit{frac}(\nu'(c_i)) \leqslant \mathit{frac}(\nu'(c_j))$.

In a timed automaton $\mathcal{A}$, a region is an equivalence class that relies on the equivalence relation $\sim$. Let $\mathcal{R}$ be the set of regions of $\mathcal{A}$. The number of regions $|\mathcal{R}|$ is bounded by $|\mathcal{C}|! \cdot 2^{|\mathcal{C}|} \cdot \prod_{c \in \mathcal{C}} (2\kappa(c) + 2)$ [40]. For a clock valuation $\nu$, $[\![\nu]\!]$ is used to represent the region containing it.

A symbolic state [40] of $\mathcal{A} = (L, l_0, F, \mathcal{C}, \Sigma, \Delta)$ is a pair $(l, [\![\nu]\!])$, where $l \in L$ and $[\![\nu]\!] \in \mathcal{R}$.

## 2.2 Active learning of deterministic timed automata

In this section, we briefly introduce the active learning method for DTA proposed in [33]. Its basic idea is as follows. First, it can be proven that if reset-clocked languages of two DTAs are equivalent (i.e., $\mathscr{L}_r(\mathcal{A}_1) = \mathscr{L}_r(\mathcal{A}_2)$), their timed languages are equivalent (i.e., $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$). Hence, the problem of learning the timed language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is transformed into that of learning $\mathscr{L}_r(\mathcal{A})$. Then, an equivalence relation on reset-clocked languages is established. According to it, $\mathscr{L}_r(\mathcal{A})$ can be divided into a finite number of equivalence classes. Subsequently, the active learning algorithm of $\mathscr{L}_r(\mathcal{A})$ is proposed.

To learn $\mathscr{L}_r(\mathcal{A})$, the equivalence relation on $\mathscr{L}_r(\mathcal{A})$ needs to be defined. To define this equivalence relation, we recall two notions region word and valid successor utilized in [33].

**Definition 4** (Region word). Given a timed automaton $\mathcal{A}$, a region word $\xi$ is a finite sequence of pairs $(\sigma, [\![\nu]\!])$, where $\sigma \in \Sigma$ is an action of $\mathcal{A}$ and $[\![\nu]\!] \in \mathcal{R}$ represents a region of $\mathcal{A}$.

Let $\boldsymbol{\Sigma_G} = \Sigma \times \mathcal{R}$ be the finite set of region actions. For each clocked word $\gamma$ of $\mathcal{A}$, there is a unique region word $\xi$ corresponding to it, denoted by $\xi = [\![\gamma]\!]$.

**Lemma 1.** Given a CTA $\mathcal{A}$, for all valid clocked words $\gamma$ and $\gamma'$ such that $[\![\gamma]\!] = [\![\gamma']\!]$, they have the same transition sequence of $\mathcal{A}$ and reach the same symbolic state. Thus, the following two conclusions hold: (1) $\gamma \in \mathscr{L}(\mathcal{A})$ iff $\gamma' \in \mathscr{L}(\mathcal{A})$; (2) $\mathit{resets}(\gamma_r) = \mathit{resets}(\gamma'_r)$, where $\gamma_r$ and $\gamma'_r$ are the reset-clocked words corresponding to $\gamma$ and $\gamma'$, respectively.

**Definition 5** (Valid successor). Given a DTA $\mathcal{A}$, a reset-clocked word $\gamma_r$ and a region word $\xi$, a reset-clocked word $\gamma'_r$ is a valid successor of $\gamma_r$ corresponding to $\xi$ if $[\![vw(\gamma'_r)]\!] = \xi$ and $\gamma_r \cdot \gamma'_r$ is a valid reset-clocked word of $\mathcal{A}$.

The set of valid successors of $\gamma_r$ corresponding to $\xi$ is denoted by $vs_{\mathcal{A}}(\gamma_r, \xi)$. The method of finding valid successors has been presented in [33].

**Lemma 2.** Given a valid reset-clocked word $\gamma_r$ of a CTA $\mathcal{A}$ and a region word $\xi$, for all $\gamma_r', \gamma_r'' \in vs_{\mathcal{A}}(\gamma_r, \xi)$, $\gamma_r\gamma_r'$ and $\gamma_r\gamma_r''$ witness the same transition sequence of $\mathcal{A}$ and reach the same symbolic state in the end.

**Lemma 3.** Given two valid reset-clocked words $\gamma_{r1}$ and $\gamma_{r2}$ of $\mathcal{A}$ and a region word $\xi$, if $\gamma_{r1}$ and $\gamma_{r2}$ reach the same symbolic state of $\mathcal{A}$, then for all $\gamma_{r1}' \in vs_{\mathcal{A}}(\gamma_{r1}, \xi)$ and $\gamma_{r2}' \in vs_{\mathcal{A}}(\gamma_{r2}, \xi)$, the following two conditions hold: (1) $resets(\gamma_{r1}') = resets(\gamma_{r2}')$; (2) $\gamma_{r1}\gamma_{r1}'$ and $\gamma_{r2}\gamma_{r2}'$ reach some same symbolic state.

**Definition 6** (Equivalence relation)**.** Suppose $\mathscr{L}_r(\mathcal{A})$ is the reset-clocked language of a DTA $\mathcal{A}$. Two reset-clocked words $\gamma_{r1}, \gamma_{r2}$ are indistinguishable by $\mathscr{L}_r(\mathcal{A})$, denoted by $\gamma_{r1} \sim_{\mathscr{L}_r(\mathcal{A})} \gamma_{r2}$, if for all $\gamma_{r1}' \in vs_{\mathcal{A}}(\gamma_{r1}, \xi)$ and $\gamma_{r2}' \in vs_{\mathcal{A}}(\gamma_{r2}, \xi)$, the following conditions hold for all $\xi \in \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$: (1) $\gamma_{r1}\gamma_{r1}' \in \mathscr{L}_r(\mathcal{A})$ iff $\gamma_{r2}\gamma_{r2}' \in \mathscr{L}_r(\mathcal{A})$; (2) $resets(\gamma_{r1}') = resets(\gamma_{r2}')$.

According to Lemma 3 and the definition of the equivalence relation, we can get the following lemma.

**Lemma 4.** If two valid reset-clocked words $\gamma_{r1}, \gamma_{r2}$ of $\mathcal{A}$ reach the same symbolic state of $\mathcal{A}$, then $\gamma_{r1} \sim_{\mathscr{L}_r(\mathcal{A})} \gamma_{r2}$.

According to Lemma 4, it can be proven that for a DTA $\mathcal{A}$, $\sim_{\mathscr{L}_r(\mathcal{A})}$ has a finite number of equivalence classes, which is the key to ensure the termination of the learning algorithm for $\mathscr{L}_r(\mathcal{A})$.

In the algorithm of learning a DTA from a powerful teacher, the learner can make three kinds of queries to the teacher: reset information query ($\mathsf{RQ}_{\mathcal{A}}$), membership query ($\mathsf{MQ}_{\mathcal{A}}$), and equivalence query ($\mathsf{EQ}$). In a reset information query, the learner asks for the reset information along the run of a valid clocked word $\gamma$. In a membership query, the teacher receives a reset-clocked word $\gamma_r$ from the learner and answers whether $\gamma_r \in \mathscr{L}_r(\mathcal{A})$. In an equivalence query, the learner submits a hypothesis DTA $\mathcal{H}$ and the teacher answers whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H})$.

The basic learning process follows the $L^*$ framework. The learner utilizes an observation table $\boldsymbol{T}$ to store query results. At the beginning, the learner makes membership queries for reset-clocked words and performs operations on the table until it is closed, consistent, and evidence-closed. After that, a hypothesis DTA $\mathcal{H}$ can be constructed according to the table and whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H})$ is determined by an equivalence query. If the answer is 'no', the learner obtains a reset-timed word $\omega_r$ as a counterexample and adds all prefixes of $\Gamma(\omega_r)$ to $\boldsymbol{T}$. The whole procedure repeats until the teacher gives a positive answer to an equivalence query. The reset information queries will be performed when it needs to compute valid successors for membership queries. In the normal teacher situation, they will be replaced by guessing.

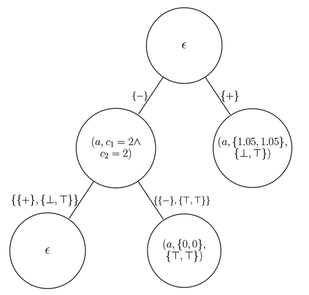
The observation table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}_{\boldsymbol{G}}, \boldsymbol{\Sigma}_r, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, g)$, where $\boldsymbol{S}, \boldsymbol{R} \subset \boldsymbol{\Sigma}_{\boldsymbol{r}}^*$ is a finite set of reset-clocked words, $\boldsymbol{E} \subset \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$ is a finite set of region words, $f$ is a classification function mapping $\boldsymbol{S} \cup \boldsymbol{R} \times \boldsymbol{E}$ to the set $\{+, -\}$, and $g$ is a labelling function mapping $\boldsymbol{S} \cup \boldsymbol{R} \times \boldsymbol{E}$ to $\{\bot, \top\}^{|\xi \in \boldsymbol{E}| \times |\mathcal{C}|}$.

For each reset-clocked word $\gamma_r \in \boldsymbol{S} \cup \boldsymbol{R}$ and each region word $\xi \in \boldsymbol{E}$, the learner needs to find a valid successor $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$ before making a membership query. We call it the additional overhead to issue a valid membership query. If $vs_{\mathcal{A}}(\gamma_r, \xi) \neq \emptyset$ and $\mathsf{MQ}_{\mathcal{A}}(\gamma_r \cdot e_r) = +$, then $f(\gamma_r, \xi) = +$, otherwise, $f(\gamma_r, \xi) = -$. $g$ is utilized to record $resets(e_r)$. A function $row$ maps each $\gamma_r \in \boldsymbol{S} \cup \boldsymbol{R}$ to a vector indexed by every $\xi \in \boldsymbol{E}$. Each element of the vector is defined as $\{f(\gamma_r, \xi), g(\gamma_r, \xi)\}$.

Before constructing a hypothesis based on the table $\boldsymbol{T}$, the learner needs to check whether $\boldsymbol{T}$ is closed, consistent, and evidence-closed. These conditions are introduced in detail as follows.

- Closed. $\forall r \in \boldsymbol{R}, \exists s \in \boldsymbol{S}$: $row(s) = row(r)$.
- Consistent. $\forall \gamma_r, \gamma_r' \in \boldsymbol{S} \cup \boldsymbol{R}$, $row(\gamma_r) = row(\gamma_r')$ implies that for all $\boldsymbol{\sigma}_r, \boldsymbol{\sigma}_r' \in \boldsymbol{\Sigma}_r$ such that $[\![vw(\boldsymbol{\sigma}_r)]\!] = [\![vw(\boldsymbol{\sigma}_r')]\!]$, if $\gamma_r\boldsymbol{\sigma}_r, \gamma_r'\boldsymbol{\sigma}_r' \in \boldsymbol{S} \cup \boldsymbol{R}$, the following conditions are satisfied: (1) $row(\gamma_r\boldsymbol{\sigma}_r) = row(\gamma_r'\boldsymbol{\sigma}_r')$; (2) $resets(\boldsymbol{\sigma}_r) = resets(\boldsymbol{\sigma}_r')$.
- Evidence-closed. $\forall s \in \boldsymbol{S}$ and $\forall e \in \boldsymbol{E}$, if $vs_{\mathcal{A}}(s, e)$ is not empty, $\exists e_r \in vs_{\mathcal{A}}(s, e)$ such that $se_r \in \boldsymbol{S} \cup \boldsymbol{R}$.

To make the table satisfy the above properties, the learner can perform the following operations. (1) If the table is not consistent, there exist $\gamma_r, \gamma_r' \in \boldsymbol{S} \cup \boldsymbol{R}$ satisfying $row(\gamma_r) = row(\gamma_r')$, for $\boldsymbol{\sigma}_r, \boldsymbol{\sigma}_r' \in \boldsymbol{\Sigma}_r$ such that $[\![vw(\boldsymbol{\sigma}_r)]\!] = [\![vw(\boldsymbol{\sigma}_r')]\!]$, $\gamma_r\boldsymbol{\sigma}_r$ and $\gamma_r'\boldsymbol{\sigma}_r' \in \boldsymbol{S} \cup \boldsymbol{R}$. However, $row(\gamma_r\boldsymbol{\sigma}_r) \neq row(\gamma_r'\boldsymbol{\sigma}_{\boldsymbol{r}}')$ or $resets(\boldsymbol{\sigma}_r) \neq resets(\boldsymbol{\sigma}_r')$. If $row(\gamma_r\boldsymbol{\sigma}_r) \neq row(\gamma_r'\boldsymbol{\sigma}_{\boldsymbol{r}}')$, the learner finds $e \in \boldsymbol{E}$ such that $f(\gamma_r\boldsymbol{\sigma}_r, e) \neq f(\gamma_r'\boldsymbol{\sigma}_{\boldsymbol{r}}', e)$ or $g(\gamma_r\boldsymbol{\sigma}_r, e) \neq g(\gamma_r'\boldsymbol{\sigma}_{\boldsymbol{r}}', e)$ and adds $\xi = [\![vw(\boldsymbol{\sigma}_r)]\!] \cdot e$ to $\boldsymbol{E}$. If $resets(\boldsymbol{\sigma}_r) \neq resets(\boldsymbol{\sigma}_r')$, the learner adds $\xi = [\![vw(\boldsymbol{\sigma}_r)]\!]$ to $\boldsymbol{E}$. In any two of the above cases, for each $\gamma_r \in \boldsymbol{S} \cup \boldsymbol{R}$, the learner finds $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$ and then makes membership queries to fill the table. (2) If $\boldsymbol{T}$ is not closed, there is $r \in \boldsymbol{R}$ such that for all $s \in \boldsymbol{S}$, $row(r) \neq row(s)$. The learner will move $r$ from $\boldsymbol{R}$ to $\boldsymbol{S}$. Moreover, for each $\sigma \in \Sigma$, a reset-clocked word $\pi(vw(r) \cdot \boldsymbol{\sigma})$ needs to be added to $\boldsymbol{R}$, where $\boldsymbol{\sigma} = (\sigma, \{0\}^{|\mathcal{C}|})$. Hence, the learner needs to add $|\Sigma|$ reset-clocked words to $\boldsymbol{R}$. After that, for each added reset-clocked word $\gamma_r$ and each $\xi \in \boldsymbol{E}$, the learner also needs to find a valid successor $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$ and then fill the table by membership queries. (3) If


$\epsilon$
$\{-\}$
$\{+\}$
$(a, c_1 = 2 \wedge c_2 = 2)$
$(a, \{1.05, 1.05\}, \{\perp, \top\})$
$\{\{+\}, \{\perp, \top\}\}$
$\{\{-\}, \{\top, \top\}\}$
$\epsilon$
$(a, \{0, 0\}, \{\top, \top\})$

**The set of leaf node jumps $T_Q$:**

$(\epsilon, (a, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$(\epsilon, (b, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$(\epsilon, (a, \{1.05, 1.05\}, \{\perp, \top\}), (a, \{1.05, 1.05\}, \{\perp, \top\}))$
$((a, \{1.05, 1.05\}, \{\perp, \top\}), (a, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$((a, \{1.05, 1.05\}, \{\perp, \top\}), (b, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$((a, \{1.05, 1.05\}, \{\perp, \top\}), (b, \{1.05, 0\}, \{\perp, \top\}), (a, \{1.05, 1.05\}, \{\perp, \top\}))$
$((a, \{1.05, 1.05\}, \{\perp, \top\}), (b, \{2.05, 1\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$((a, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$
$((a, \{0, 0\}, \{\top, \top\}), (b, \{0, 0\}, \{\top, \top\}), (a, \{0, 0\}, \{\top, \top\}))$

**Figure 2** A $\mathcal{TR}$ generated in the process of learning DTA $\mathcal{A}$ in Figure 1.

$\boldsymbol{T}$ is not evidence-closed, $\exists s \in \boldsymbol{S}$ and $\exists e \in \boldsymbol{E}$, $vs_{\mathcal{A}}(s, e)$ is not empty, and there is no $e_r \in vs_{\mathcal{A}}(s, e)$ such that $se_r \in \boldsymbol{S} \cup \boldsymbol{R}$. The learner finds $e_r \in vs_{\mathcal{A}}(s, e)$ and then adds all prefixes of $se_r$ to $\boldsymbol{R}$. Next, for each added reset-clocked word $\gamma_r$ and each $\xi \in \boldsymbol{E}$, the leaner also finds $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$ and then fills the table by membership queries. Therefore, all operations lead to membership queries and additional overheads.

## 3 Timed classification tree

Before presenting the definition of timed classification trees, we introduce two functions as follows. They are helpful for labelling membership query results and resetting information on the edges of a tree.

- $f_{tr}$ is a function mapping $\boldsymbol{\Sigma}_r^* \times \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$ to the set $\{+, -\}$. For a reset-clocked word $\gamma_r \in \boldsymbol{\Sigma}_r^*$ and a region word $\xi \in \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$, in the case of $\xi \neq \epsilon$, find a valid successor $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$. If $vs_{\mathcal{A}}(\gamma_r, \xi) \neq \emptyset$ and $\mathsf{MQ}_{\mathcal{A}}(\gamma_r \cdot e_r) = +$, $f_{tr}(\gamma_r, \xi) = +$, otherwise, $f_{tr}(\gamma_r, \xi) = -$. If $\xi = \epsilon$, $f_{tr}(\gamma_r, \xi) = \mathsf{MQ}_{\mathcal{A}}(\gamma_r)$.
- $g_{tr}$ is function mapping $\boldsymbol{\Sigma}_r^* \times \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$ to $\{\perp, \top\}^{|\xi \in \boldsymbol{\Sigma}_{\boldsymbol{G}}^*| \times |\mathcal{C}|}$. For a reset-clocked word $\gamma_r \in \boldsymbol{\Sigma}_r^*$ and a region word $\xi \in \boldsymbol{\Sigma}_{\boldsymbol{G}}^*$, in the case of $\xi \neq \epsilon$, find a valid successor $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$. If $vs_{\mathcal{A}}(\gamma_r, \xi) \neq \emptyset$, $g_{tr}(\gamma_r, \xi) = \mathit{resets}(e_r)$. Otherwise, let $\xi[:i]$ be the prefix before the $i$-th position (including the $i$-th position) of $\xi$, where $1 \leqslant i < |\xi|$. If there exists $i$ such that $vs_{\mathcal{A}}(\gamma_r, \xi[:i]) \neq \emptyset$ and $vs_{\mathcal{A}}(\gamma_r, \xi[:i+1]) = \emptyset$, find an $e'_r \in vs_{\mathcal{A}}(\gamma_r, \xi[:i])$ and $g_{tr}(\gamma_r, \xi) = \{\mathit{resets}(e'_r), \{\top\}^{(|\xi|-i) \times |\mathcal{C}|}\}$. Otherwise, $g_{tr}(\gamma_r, \xi) = \{\top\}^{|\xi| \times |\mathcal{C}|}$. In the case of $\xi = \epsilon$, $g_{tr}(\gamma_r, \xi) = \emptyset$.

A timed classification tree contains leaf nodes and internal nodes. A leaf node stores a reset-clocked word and an internal node stores a region word. In the following, without ambiguity, we reuse $\xi$ to denote an internal node storing a region word $\xi$ and reuse $\gamma_r$ to be a leaf node storing a reset-clocked word $\gamma_r$.

**Definition 7** (Timed classification tree)**.** A timed classification tree $\mathcal{TR} = (\Sigma, V, Q, E, f_{tr}^E, g_{tr}^E, T_Q)$ is a multi-way tree, where $\Sigma$ is the alphabet; $V$ is a finite set of internal nodes; $Q$ is a finite set of leaf nodes; $E \subseteq V \times \{V \cup Q\}$ is a finite set of edges;

- $f_{tr}^E$ is a classification function mapping $E$ to the set $\{+, -\}$. For each edge $e = (u_1, u_2) \in E$, if $u_2 \in V$, find an arbitrary leaf node $q$ under the sub-tree of $u_2$, and if $u_2 \in Q$, let $q = u_2$. Let $\gamma_r$ be the reset-clocked word stored in $q$ and $\xi$ be the region word stored in $u_1$. We have $f_{tr}^E(e) = f_{tr}(\gamma_r, \xi)$.
- $g_{tr}^E$ is a labelling function mapping $E$ to $\{\perp, \top\}^{|\xi \in V| \times |\mathcal{C}|}$. For each edge $e = (u_1, u_2) \in E$, same to $f_{tr}^E$, find a leaf node $q$ for $u_2$. For the reset-clocked word $\gamma_r$ stored in $q$ and the region word $\xi$ stored in $u_1$, $g_{tr}^E(e) = g_{tr}(\gamma_r, \xi)$.
- $T_Q \subseteq Q \times \boldsymbol{\Sigma_r} \times Q$ is a finite set describing the relations between leaf nodes. A leaf node jump $(q, (\sigma, \boldsymbol{v}, \boldsymbol{b}), q')$ in $T_Q$ represents a jump from the leaf node $q$ to the leaf node $q'$ by performing the reset-clocked action $(\sigma, \boldsymbol{v}, \boldsymbol{b})$.

Intuitively, a leaf node indicates a location in the hypothesis, the internal nodes are used to distinguish the leaf nodes, and the edges record the membership query results and reset information. Using the tree operations described subsequently, we always keep that for each two leaf nodes $\gamma_{r1}, \gamma_{r2}$, there exists a shared internal node $\xi$ on the paths from the root to each of the two nodes, which serves to distinguish them, i.e., $f_{tr}(\gamma_{r1}, \xi) \neq f_{tr}(\gamma_{r2}, \xi)$ or $g_{tr}(\gamma_{1r}, \xi) \neq g_{tr}(\gamma_{r2}, \xi)$. For each leaf node, there exists at least one jump from it in the set $T_Q$. The leaf node jumps explicitly hold the information which will be used to construct a hypothesis. Specifically, when constructing a hypothesis from a tree (described later), each leaf node corresponds to one location in the hypothesis, and the leaf node jumps form the transitions.

**Example 1.** Suppose DTA $\mathcal{A}$ in Figure 1 is the target DTA. Figure 2 is a tree $\mathcal{TR}$ generated in the learning process. It contains two internal nodes and three leaf nodes. Taking the edge from the internal node $\xi = (a, c_1 = 2 \wedge c_2 = 2)$ to the leaf node $\gamma_r = (a, \{0,0\}, \{\top,\top\})$ as an example, $f^E_{tr}((\xi,\gamma_r))$ and $g^E_{tr}((\xi,\gamma_r))$ are computed as follows. Since $\gamma_r$ is a leaf node, according to the definitions of $f^E_{tr}((\xi,\gamma_r))$ and $g^E_{tr}((\xi,\gamma_r))$, $f^E_{tr}((\xi,\gamma_r)) = f_{tr}(\gamma_r,\xi)$ and $g^E_{tr}((\xi,\gamma_r)) = g_{tr}(\gamma_r,\xi)$. To calculate $f_{tr}(\gamma_r,\xi)$ and $g_{tr}(\gamma_r,\xi)$, the learner first finds $e_r = (a, \{2,2\}, \{\top,\top\}) \in \mathit{vs}_{\mathcal{A}}(\gamma_r,\xi)$ and then gets $g_{tr}(\gamma_r,\xi) = \mathit{resets}(e_r) = \{\top,\top\} = g^E_{tr}((\xi,\gamma_r))$. Since $\mathsf{MQ}_{\mathcal{A}}(\gamma_r \cdot e_r) = \mathsf{MQ}_{\mathcal{A}}((a,\{0,0\},\{\top,\top\}) \cdot (a,\{2,2\},\{\top,\top\})) = -$, $f_{tr}(\gamma_r,\xi) = - = f^E_{tr}((\xi,\gamma_r))$. For the leaf node $\epsilon$, there are three leaf node jumps from it.

We introduce three operations on a tree, namely sift, leaf node jump construction, and split.

**Sift.** Given a reset-clocked word $\gamma_r$, the sift operation $\mathit{sift}(\gamma_r)$ returns a leaf node in $\mathcal{TR}$ corresponding to $\gamma_r$, which is performed as follows.

(1) Let the root node be the current node $x$. Note that $x$ contains a region word.

(2) Compute $f_{tr}(\gamma_r,x)$ and $g_{tr}(\gamma_r,x)$. Next, find an edge $(x,y)$ from $x$ such that $f^E_{tr}((x,y)) = f_{tr}(\gamma_r,x)$ and $g^E_{tr}((x,y)) = g_{tr}(\gamma_r,x)$, and then set $y$ to be the current node $x$.

(3) Repeat Step 2 until $x$ is a leaf node or there is no edge $(x,y)$ from $x$ such that $f^E_{tr}((x,y)) = f_{tr}(\gamma_r,x)$ and $g^E_{tr}((x,y)) = g_{tr}(\gamma_r,x)$. If $x$ is a leaf node, the sift operation ends and returns $x$ as the leaf node corresponding to $\gamma_r$ in $\mathcal{TR}$. Otherwise, construct a new leaf node containing $\gamma_r$ as a child of $x$, add a new edge $(x,\gamma_r)$ labelled by $f^E_{tr}((x,\gamma_r)) = f_{tr}(\gamma_r,x)$ and $g^E_{tr}((x,\gamma_r)) = g_{tr}(\gamma_r,x)$ to $E$, and then return the new leaf node $\gamma_r$.

Due to the nature of tree structure and the information stored on edges of timed classification tree $\mathcal{TR}$, to find a corresponding leaf node to a reset-clock word $\gamma_r$, the learner only needs to go through one path from the root, skipping the checks of other branches via $f_{tr}$ and $g_{tr}$. Therefore, it reduces the membership queries and thus also the additional overheads.

**Theorem 1.** For two reset-clocked words $\gamma_{r1}$ and $\gamma_{r2}$ reaching the same symbolic state of $\mathcal{A}$, $\mathit{sift}(\gamma_{r1})$ and $\mathit{sift}(\gamma_{r2})$ return the same leaf node.

*Proof.* Depending on the definition of the symbolic states, the clock valuations after running $\gamma_{r1}$ and $\gamma_{r2}$ on $\mathcal{A}$ must belong to the same region. Because by letting time pass from any two points in the same region, the next visited region is the same [41], there are only two cases about the valid successors of $\gamma_{r1}$ and $\gamma_{r2}$ according to any internal node $\xi$. The first case is $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi) \neq \emptyset$ and the second case is $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi) = \mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi) = \emptyset$.

Consider the first case. Suppose that $\gamma'_{r1} \in \mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi)$ and $\gamma'_{r2} \in \mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi)$. Since $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi) \neq \emptyset$, both $\gamma_{r1}$ and $\gamma_{r2}$ are valid. Therefore, according to Lemma 3.6 in [33], $\mathit{resets}(\gamma'_{r1}) = \mathit{resets}(\gamma'_{r2})$ and $\gamma_{r1}\gamma'_{r1}$, $\gamma_{r2}\gamma'_{r2}$ also reach another same symbolic state in $\mathcal{A}$. Hence, $g_{tr}(\gamma_{r1},\xi) = g_{tr}(\gamma_{r2},\xi)$ and $f_{tr}(\gamma_{r1},\xi) = f_{tr}(\gamma_{r2},\xi)$.

Consider the second case. In this case, $f_{tr}(\gamma_{r1},\xi) = f_{tr}(\gamma_{r2},\xi) = -$. To compute $g_{tr}(\gamma_{r1},\xi)$ and $g_{tr}(\gamma_{r2},\xi)$, the learner finds $i$ such that $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i]) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i+1]) = \emptyset$, where $1 \leqslant i < |\xi|$. Since by letting time pass from any two points in the same region, the next visited region is the same [41], if there is $i$ such that $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i]) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i+1]) = \emptyset$, then $\mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi[:i]) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi[:i+1]) = \emptyset$. Suppose that $\gamma''_{r1} \in \mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i])$ and $\gamma''_{r2} \in \mathit{vs}_{\mathcal{A}}(\gamma_{r2},\xi[:i])$. According to Lemma 3.6 in [33], $\mathit{resets}(\gamma''_{r1}) = \mathit{resets}(\gamma''_{r2})$. Hence, $g_{tr}(\gamma_{r1},\xi) = g_{tr}(\gamma_{r2},\xi) = \{\mathit{resets}(\gamma''_{r1}), \{\top\}^{(|\xi|-i)\times|\mathcal{C}|}\}$. If there is no $i$ such that $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i]) \neq \emptyset$ and $\mathit{vs}_{\mathcal{A}}(\gamma_{r1},\xi[:i+1]) = \emptyset$, $g_{tr}(\gamma_{r1},\xi) = g_{tr}(\gamma_{r2},\xi) = \{\top\}^{|\xi|\times|\mathcal{C}|}$.

Therefore, for any internal node $\xi$, $f_{tr}(\gamma_{r1},\xi) = f_{tr}(\gamma_{r2},\xi)$ and $g_{tr}(\gamma_{r1},\xi) = g_{tr}(\gamma_{r2},\xi)$. So $\mathit{sift}(\gamma_{r1})$ and $\mathit{sift}(\gamma_{r2})$ return the same leaf node.

**Leaf node jump construction.** To construct a leaf node jump from a leaf node $\gamma_r$ by performing a clocked action $\boldsymbol{\sigma} = (\sigma, \boldsymbol{v}) \in \boldsymbol{\Sigma}$, the learner executes the following two steps.

(1) If the clocked word $vw(\gamma_r)\cdot(\sigma,\boldsymbol{v})$ is valid for $\mathcal{A}$, obtain a reset-clocked word $\gamma_r \cdot (\sigma,\boldsymbol{v},\boldsymbol{b})$ by making $\mathsf{RQ}_{\mathcal{A}}(vw(\gamma_r)\cdot(\sigma,\boldsymbol{v}))$. Otherwise, construct a reset-clocked word $\gamma_r \cdot (\sigma,\boldsymbol{v},\boldsymbol{b})$, where $\boldsymbol{b} = \{\top\}^{|\mathcal{C}|}$.

(2) Compute $\mathit{sift}(\gamma_r \cdot (\sigma,\boldsymbol{v},\boldsymbol{b}))$ and add a jump $(\gamma_r, (\sigma,\boldsymbol{v},\boldsymbol{b}), \mathit{sift}(\gamma_r\cdot(\sigma,\boldsymbol{v},\boldsymbol{b})))$ to $T_Q$.

Note that, after each new leaf node $q$ is added to $\mathcal{TR}$, for each action $\sigma \in \Sigma$, a leaf node jump from $q$ by performing $(\sigma, 0^{|\mathcal{C}|})$ will be constructed.

**Theorem 2.** In $T_Q$, for any two jumps $(q, \boldsymbol{\sigma}_r, \mathit{sift}(q\cdot\boldsymbol{\sigma}_r))$ and $(q, \boldsymbol{\sigma}'_r, \mathit{sift}(q\cdot\boldsymbol{\sigma}'_r))$ from a leaf node $q$, if $[\![vw(\boldsymbol{\sigma}_r)]\!] = [\![vw(\boldsymbol{\sigma}'_r)]\!]$, we have $\mathit{resets}(\boldsymbol{\sigma}_r) = \mathit{resets}(\boldsymbol{\sigma}'_r)$ and $\mathit{sift}(q\cdot\boldsymbol{\sigma}_r) = \mathit{sift}(q\cdot\boldsymbol{\sigma}'_r)$.

*Proof.* If $\mathit{vs}_{\mathcal{A}}(q, [\![vw(\boldsymbol{\sigma}_r)]\!]) \neq \emptyset$, according to the construction method for a leaf node jump in the whole paper, both $\boldsymbol{\sigma}_r$ and $\boldsymbol{\sigma}'_r \in \mathit{vs}_{\mathcal{A}}(q, [\![vw(\boldsymbol{\sigma}_r)]\!])$. Since $\mathit{vs}_{\mathcal{A}}(q, [\![vw(\boldsymbol{\sigma}_r)]\!]) \neq \emptyset$, $q$ is valid. According to Lemma 3.5 in [33], $q\cdot\boldsymbol{\sigma}_r$ and $q\cdot\boldsymbol{\sigma}'_r$ must witness the same transition sequence of $\mathcal{A}$ and reach to the same symbolic state. Hence, $\mathit{resets}(\boldsymbol{\sigma}_r) = \mathit{resets}(\boldsymbol{\sigma}'_r)$ and $\mathit{sift}(q\cdot\boldsymbol{\sigma}_r) = \mathit{sift}(q\cdot\boldsymbol{\sigma}'_r)$ according to Theorem 1. If $\mathit{vs}_{\mathcal{A}}(q, [\![vw(\boldsymbol{\sigma}_r)]\!]) = \emptyset$, according to the construction method for a leaf node jump,

**Algorithm 1** Learning DTA via timed classification tree.

```
Input  : timed classification tree $\mathcal{TR} = (\Sigma, V, Q, E, f_{tr}^E, g_{tr}^E, T_Q)$; the number of clocks $|\mathcal{C}|$.
Output: a DTA $\mathcal{H}$ recognizing the target language $\mathcal{L}$.
$\mathcal{TR} \leftarrow$ initialize();                                        // initialization
$\mathcal{M} \leftarrow$ build_DFA($\mathcal{TR}$);                                // transforming $\mathcal{TR}$ to a DFA $\mathcal{M}$
$\mathcal{H} \leftarrow$ build_hypothesis($\mathcal{M}$);              // constructing a hypothesis $\mathcal{H}$ from $\mathcal{M}$ [33]
$equivalent$, $\omega_r \leftarrow$ equivalence_query($\mathcal{H}$);
while $equivalent = \perp$ do
    $\mathcal{TR} \leftarrow$ process_counterexample($\omega_r$);
    $\mathcal{M} \leftarrow$ build_DFA($\mathcal{TR}$);
    $\mathcal{H} \leftarrow$ build_hypothesis($\mathcal{M}$);
    $equivalent$, $\omega_r \leftarrow$ equivalence_query($\mathcal{H}$);
return $\mathcal{H}$;
```

$resets(\boldsymbol{\sigma}_r) = resets(\boldsymbol{\sigma}'_r) = \{\top\}^{|\mathcal{C}|}$ and $q \cdot \boldsymbol{\sigma}_r$, $q \cdot \boldsymbol{\sigma}'_r$ are invalid. Therefore, for each $\xi \in V$, $f_{tr}(q \cdot \boldsymbol{\sigma}_r, \xi) = f_{tr}(q \cdot \boldsymbol{\sigma}'_r, \xi) = -$ and $g_{tr}(q \cdot \boldsymbol{\sigma}_r, \xi) = g_{tr}(q \cdot \boldsymbol{\sigma}'_r, \xi) = \{\top\}^{|\xi| \times |\mathcal{C}|}$. Hence, $sift(q \cdot \boldsymbol{\sigma}_r) = sift(q \cdot \boldsymbol{\sigma}'_r)$.

**Split.** For a reset-clocked words $\gamma_r$ that is sifted to the leaf node $q$ in $\mathcal{TR}$, if there exists a region word $\xi$ such that $f_{tr}(\gamma_r, \xi) \neq f_{tr}(q, \xi)$ or $g_{tr}(\gamma_r, \xi) \neq g_{tr}(q, \xi)$, it means that $\gamma_r$ and $q$ can be distinguished by $\xi$. In this case, the learner uses $\xi$ to split $q$ as follows.

(1) Record the parent of $q$ as $d_0$, introduce a new internal node containing $\xi$ instead of $q$ as the child of $d_0$, modify the edge $(d_0, q)$ to $(d_0, \xi)$, and let $f_{tr}^E((d_0, \xi)) = f_{tr}^E((d_0, q))$ and $g_{tr}^E((d_0, \xi)) = g_{tr}^E((d_0, q))$.

(2) Construct a new leaf node $\gamma_r$ and let $q$ and $\gamma_r$ be two children of $\xi$. Then add an edge $(\xi, \gamma_r)$ in $E$, compute $f_{tr}(\gamma_r, \xi)$ and $g_{tr}(\gamma_r, \xi)$ to label the edge by $f_{tr}^E((\xi, \gamma_r))$ and $g_{tr}^E((\xi, \gamma_r))$. Similarly, add an edge $(\xi, q)$ in $E$ and calculate $f_{tr}(q, \xi)$ and $g_{tr}(q, \xi)$ to label the edge with $f_{tr}^E((\xi, q))$ and $g_{tr}^E((\xi, q))$.

(3) For each leaf node jump $(u, \boldsymbol{\sigma}_r, q)$ in $T_Q$ pointing to $q$, change it to $(u, \boldsymbol{\sigma}_r, sift(u \cdot \boldsymbol{\sigma}_r))$.

Note that since $sift(u \cdot \boldsymbol{\sigma}_r)$ has already been computed during constructing the leaf node jump $(u, \boldsymbol{\sigma}_r, q)$, here $sift(u \cdot \boldsymbol{\sigma}_r)$ can skip the re-computation w.r.t the existed internal nodes in the second step of the sift operation and compute $f_{tr}(u \cdot \boldsymbol{\sigma}_r, \xi)$ and $g_{tr}(u \cdot \boldsymbol{\sigma}_r, \xi)$ according to the new internal node $\xi$ directly.

The split operation adds a new internal node $\xi$ to distinguish two leaf nodes. With respect to $\xi$, we make membership queries only for each leaf node $u$ pointing to $q$ in the third step, instead of all leaf nodes. This is analogous to adding a new suffix to the set $\boldsymbol{E}$ of an observation table, while membership queries should be performed for every prefix in the table.

**DFA construction.** Given $\mathcal{TR} = (\Sigma, V, Q, E, f_{tr}^E, g_{tr}^E, T_Q)$, we can construct a deterministic finite state automaton (DFA) $\mathcal{M} = (L_M, l_M^0, F_M, \Sigma_M, \Delta_M)$ where

- the finite abstract alphabet $\Sigma_M = \{\boldsymbol{\sigma}_r \mid (q, \boldsymbol{\sigma}_r, q') \in T_Q\}$;
- the finite set of locations $L_M = \{l_q \mid q \in Q\}$;
- the initial location $l_M^0 = l_\epsilon$ for $\epsilon \in Q$;
- the finite set of accepting locations $F_M = \{l_q \mid \mathsf{MQ}_{\mathcal{A}}(q) = +, q \in Q\}$;
- the finite set of transitions $\Delta_M = \{(l_q, \boldsymbol{\sigma}_r, l_{q'}) \mid (q, \boldsymbol{\sigma}_r, q') \in T_Q\}$.

According to Theorem 2 and the method for constructing $\mathcal{M}$ from $\mathcal{TR}$, for each $l \in L_M$, $\sigma \in \Sigma$ in $\mathcal{M}$, if there are two transitions $(l, (\sigma, \boldsymbol{v}, \boldsymbol{b}), l')$ and $(l, (\sigma, \boldsymbol{v}', \boldsymbol{b}'), l'')$ such that $[\![\boldsymbol{v}]\!] = [\![\boldsymbol{v}']\!]$, $l' = l''$, and $\boldsymbol{b} = \boldsymbol{b}'$.

## 4 Learning DTA via timed classification tree

Based on a timed classification tree, we design a novel learning algorithm, as shown in Algorithm 1. First, the learner initializes the classification tree $\mathcal{TR} = (\Sigma, V, Q, E, f_{tr}^E, g_{tr}^E, T_Q)$, where $V = \{\epsilon\}$. Performing the operation $sift(\epsilon)$, a leaf node $\epsilon$ is added to $Q$. Next, for each $\sigma \in \Sigma$, the learner constructs a leaf node jump from the leaf node $\epsilon$ by performing $(\sigma, 0^{|\mathcal{C}|})$ and then adds this jump to $T_Q$. An abstract DFA $\mathcal{M}$ is constructed based on $\mathcal{TR}$. Based on the partition function proposed in [33], $\mathcal{M}$ can be transformed into a CTA $\mathcal{H}$ as a hypothesis. An equivalence query is issued to determine whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H})$. If the answer is 'no', a reset-timed word $\omega_r$ is returned as a counterexample. Subsequently, the counterexample analysis and processing are performed to modify $\mathcal{TR}$ according to $\omega_r$. The whole procedure repeats until the teacher gives a positive answer to an equivalence query. Before proving the termination and correctness of Algorithm 1, we present the counterexample analysis and processing in detail.

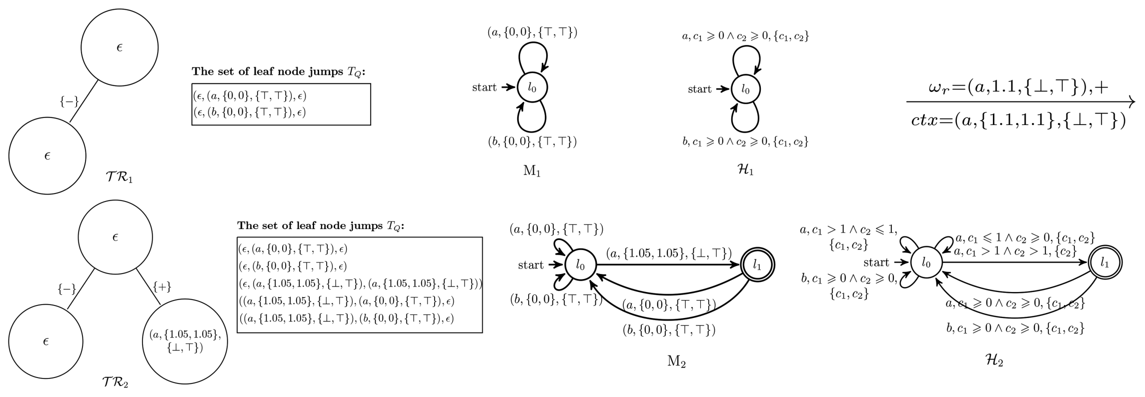

$\epsilon$
$\{-\}$
$\epsilon$
$\mathcal{TR}_1$
The set of leaf node jumps $T_Q$:
$(\epsilon, (a, \{0,0\}, \{\top,\top\}), \epsilon)$
$(\epsilon, (b, \{0,0\}, \{\top,\top\}), \epsilon)$
$(a, \{0,0\}, \{\top,\top\})$
start
$l_0$
$(b, \{0,0\}, \{\top,\top\})$
$\mathrm{M}_1$
$a, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
start
$l_0$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$\mathcal{H}_1$
$\omega_r = (a, 1.1, \{\bot,\top\}), +$
$ctx = (a, \{1.1, 1.1\}, \{\bot,\top\})$
$\epsilon$
$\{-\}$
$\{+\}$
$\epsilon$
$(a, \{1.05, 1.05\}, \{\bot,\top\})$
$\mathcal{TR}_2$
The set of leaf node jumps $T_Q$:
$(\epsilon, (a, \{0,0\}, \{\top,\top\}), \epsilon)$
$(\epsilon, (b, \{0,0\}, \{\top,\top\}), \epsilon)$
$(\epsilon, (a, \{1.05, 1.05\}, \{\bot,\top\}), (a, \{1.05, 1.05\}, \{\bot,\top\}))$
$((a, \{1.05, 1.05\}, \{\bot,\top\}), (a, \{0,0\}, \{\top,\top\}), \epsilon)$
$((a, \{1.05, 1.05\}, \{\bot,\top\}), (b, \{0,0\}, \{\top,\top\}), \epsilon)$
$(a, \{0,0\}, \{\top,\top\})$
start
$l_0$
$(a, \{1.05, 1.05\}, \{\bot,\top\})$
$l_1$
$(b, \{0,0\}, \{\top,\top\})$
$(a, \{0,0\}, \{\top,\top\})$
$(b, \{0,0\}, \{\top,\top\})$
$\mathrm{M}_2$
$a, c_1 > 1 \wedge c_2 \leqslant 1, \{c_1, c_2\}$
$a, c_1 \leqslant 1 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$a, c_1 > 1 \wedge c_2 > 1, \{c_2\}$
start
$l_0$
$l_1$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$a, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$b, c_1 \geqslant 0 \wedge c_2 \geqslant 0, \{c_1, c_2\}$
$\mathcal{H}_2$

**Figure 3** An example for the prefix analysis and the counterexample processing I.

## 4.1 Counterexample analysis and processing

Given a counterexample $\omega_r$, we transform it into the corresponding reset-clocked word $ctx = \Gamma(\omega_r)$. Let $ctx[{:}\ i]$ and $ctx[i\ {:}]$ be the prefix before and the suffix after the $i$-th position (including the $i$-th position) of $ctx$, respectively. Let $ctx[i]$ be the reset-clocked action at the $i$-th position of $ctx$. To identify an erroneous position $EI$ in $ctx$, the learner sequentially examines each position $i$ of $ctx$ from the front to the end, through the prefix analysis and the suffix analysis in turn. Upon identifying an $EI$, the learner modifies $\mathcal{TR}$ based on the conflict between $\mathcal{H}$ and $\mathcal{A}$ induced by $EI$.

**Prefix analysis.** Given a position $i$, the learner finds the corresponding reset-clocked word $\gamma_r$ of the clocked word $vw(ctx[{:}\ i])$ according to $\mathcal{H}$. If $\gamma_r \neq ctx[{:}\ i]$, it means that $vw(ctx[{:}\ i])$ corresponds to different reset-clocked words in $\mathcal{H}$ and $\mathcal{A}$, respectively. Therefore, $i$ is an erroneous position, e.g., $EI = i$.

**Counterexample processing I.** If $EI$ is found during the prefix analysis, $ctx$ is processed as follows. Let $l_u$ be the location reached by executing $ctx[{:}\ EI - 1]$ in the hypothesis $\mathcal{H}$ and $u$ be the corresponding leaf node in $\mathcal{TR}$. Then construct a region action $[\![vw(ctx[EI])]\!]$ and find a valid successor $e_r \in vs_{\mathcal{A}}(u, [\![vw(ctx[EI])]\!])$. If there is such $e_r$, then $e_r$ is a reset-clocked action and let $\boldsymbol{\sigma}_r = e_r$. Otherwise, construct a reset-clocked action $\boldsymbol{\sigma}_r = (vw(ctx[EI]), \{\top\}^{|\mathcal{C}|})$. After that, add a new leaf node jump $(u, \boldsymbol{\sigma}_r, \mathit{sift}(u \cdot \boldsymbol{\sigma}_r))$ to $T_Q$. If $\mathit{sift}(u \cdot \boldsymbol{\sigma}_r)$ returns a new leaf node, the leaf node jump construction is conducted. For each $\sigma \in \Sigma$, a jump from the new leaf node by performing $(\sigma, 0^{|\mathcal{C}|})$ is added to $T_Q$.

**Example 2.** Figure 3 gives an example for the prefix analysis and the counterexample processing I. Suppose the DTA $\mathcal{A}$ in Figure 1 is the target DTA. In Figure 3, the classification tree $\mathcal{TR}_1$ is obtained by initialization, and $\mathcal{M}_1$ and $\mathcal{H}_1$ are the DFA and hypothesis corresponding to $\mathcal{TR}_1$, respectively. By initiating an equivalence query on $\mathcal{H}_1$, a counterexample $\omega_r = (a, 1.1, \{\bot, \top\})$ is returned. We have $ctx = \Gamma(\omega_r) = (a, \{1.1, 1.1\}, \{\bot, \top\})$. To analyse the first position in $ctx$, $vw(ctx[{:}\ 1]) = (a, \{1.1, 1.1\})$, the learner finds the corresponding reset-clocked word $\gamma_r = (a, \{1.1, 1.1\}, \{\top, \top\})$ of $vw(ctx[{:}\ 1])$ in $\mathcal{H}_1$, which is not equal to $ctx[{:}\ 1]$. Hence, let $EI = 1$. To process $ctx$, since $ctx[{:}\ EI - 1] = \epsilon$, $l_0$ is found and the leaf node $\epsilon$ in $\mathcal{TR}_1$ corresponds to $l_0$. After that, the learner finds $e_r = (a, \{1.05, 1.05\}, \{\bot, \top\}) \in vs_{\mathcal{A}}(\epsilon, [\![(a, \{1.1, 1.1\})]\!])$ and thus lets $\boldsymbol{\sigma}_r = e_r$. After that, a new leaf node jump $(\epsilon, (a, \{1.05, 1.05\}, \{\bot, \top\}), \mathit{sift}((a, \{1.05, 1.05\}, \{\bot, \top\})))$ is added in $T_Q$. In this process, a new leaf node $(a, \{1.05, 1.05\}, \{\bot, \top\})$ is created through the operation $\mathit{sift}((a, \{1.05, 1.05\}, \{\bot, \top\}))$. Hence, two leaf node jumps from $(a, \{1.05, 1.05\}, \{\bot, \top\})$ by performing $(a, \{0, 0\}, \{\top, \top\})$ and $(b, \{0, 0\}, \{\top, \top\})$ are added to $T_Q$, respectively.

**Suffix analysis.** If $i$ is not an erroneous position in the prefix analysis, a suffix analysis starting from $i$ is conducted as follows. Let $l_q$ denote the location reached by executing $ctx[{:}\ i]$ in the hypothesis $\mathcal{H}$ and $q$ be the leaf node corresponding to $l_q$. Next, the learner converts $ctx[i + 1\ {:}]$ to a region word $[\![vw(ctx[i + 1\ {:}])]\!]$. After that, the learner computes $f_{tr}(q, [\![vw(ctx[i + 1\ {:}])]\!])$ and $g_{tr}(q, [\![vw(ctx[i + 1\ {:}])]\!])$. If $f_{tr}(q, [\![vw(ctx[i + 1\ {:}])]\!]) \neq \mathsf{MQ}_{\mathcal{A}}(ctx)$ or $g_{tr}(q, [\![vw(ctx[i + 1\ {:}])]\!]) \neq \mathit{resets}(ctx[{:}\ i + 1])$, then let $EI = i$.

**Counterexample processing II.** If $EI$ is found during the suffix analysis, the learner processes $ctx$ as follows. Let $l_u$ be the location reached by executing $ctx[{:}\ EI - 1]$ in the hypothesis, and $u$ denotes the leaf node corresponding to $l_u$. Convert $ctx[EI]$ to a region word $[\![vw(ctx[EI])]\!]$ and then find a valid successor $e_r \in vs_{\mathcal{A}}(u, [\![vw(ctx[EI])]\!])$. If there exists such $e_r$, let $\boldsymbol{\sigma}_r = e_r$. Otherwise, construct a reset-clocked action $\boldsymbol{\sigma}_r = (vw(ctx[EI]), \{\top\}^{|\mathcal{C}|})$. Then perform a sift operation to obtain a leaf node $u' = \mathit{sift}(u \cdot \boldsymbol{\sigma}_r)$.

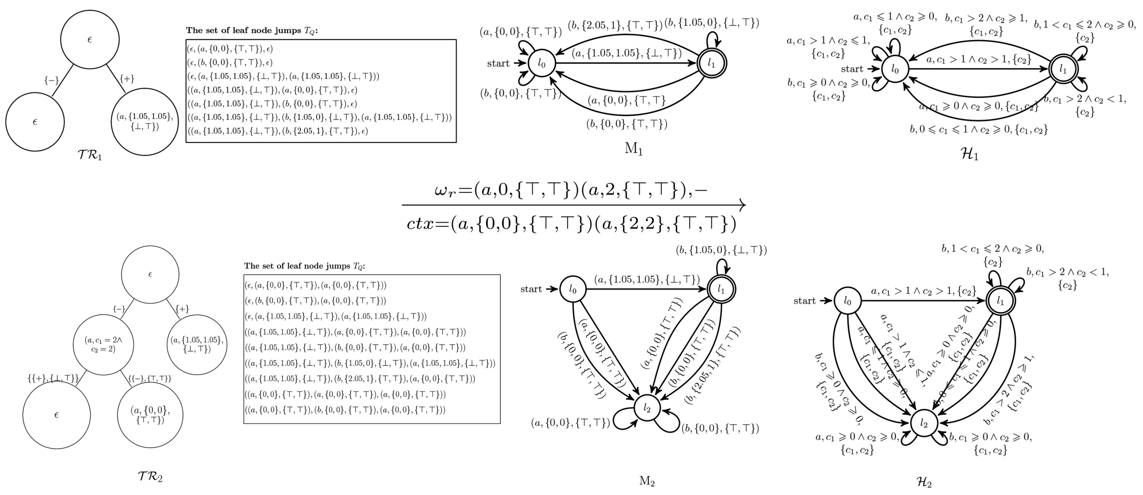

$\mathcal{TR}_1$
The set of leaf node jumps $T_Q$:
$(\epsilon,(a,\{0,0\},\{\top,\top\}),\epsilon)$
$(\epsilon,(b,\{0,0\},\{\top,\top\}),\epsilon)$
$(\epsilon,(a,\{1.05,1.05\},\{\bot,\top\}),(a,\{1.05,1.05\},\{\bot,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(a,\{0,0\},\{\top,\top\}),\epsilon)$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{0,0\},\{\top,\top\}),\epsilon)$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{1.05,0\},\{\bot,\top\}),(a,\{1.05,1.05\},\{\bot,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{2.05,1\},\{\top,\top\}),\epsilon)$
$\mathrm{M}_1$
$\mathcal{H}_1$
$\omega_r=(a,0,\{\top,\top\})(a,2,\{\top,\top\}),-$
$ctx=(a,\{0,0\},\{\top,\top\})(a,\{2,2\},\{\top,\top\})$
$\mathcal{TR}_2$
The set of leaf node jumps $T_Q$:
$(\epsilon,(a,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$(\epsilon,(b,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$(\epsilon,(a,\{1.05,1.05\},\{\bot,\top\}),(a,\{1.05,1.05\},\{\bot,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(a,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{1.05,0\},\{\bot,\top\}),(a,\{1.05,1.05\},\{\bot,\top\}))$
$((a,\{1.05,1.05\},\{\bot,\top\}),(b,\{2.05,1\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$((a,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$((a,\{0,0\},\{\top,\top\}),(b,\{0,0\},\{\top,\top\}),(a,\{0,0\},\{\top,\top\}))$
$\mathrm{M}_2$
$\mathcal{H}_2$

**Figure 4** An example for the suffix analysis and the counterexample processing II.

• If $u'=q$, it means that $u\cdot\boldsymbol{\sigma}_r$ is sifted to $q$. Since the $(EI-1)$-th position is not an erroneous position, we have $\mathsf{MQ}_{\mathcal{A}}(ctx)=f_{tr}(u,[\![vw(ctx[EI:])]\!])=f_{tr}(u\cdot\boldsymbol{\sigma}_r,[\![vw(ctx[EI+1:])]\!])$ and $resets(ctx[EI:])=g_{tr}(u,[\![vw(ctx[EI:])]\!])$. It follows that $resets(ctx[EI+1:])=g_{tr}(u\cdot\boldsymbol{\sigma}_r,[\![vw(ctx[EI+1:])]\!])$. In addition, the suffix analysis shows that $f_{tr}(q,[\![vw(ctx[EI+1:])]\!])\neq\mathsf{MQ}_{\mathcal{A}}(ctx)=f_{tr}(u\cdot\boldsymbol{\sigma}_r,[\![vw(ctx[EI+1:])]\!])$ or $g_{tr}(q,[\![vw(ctx[EI+1:])]\!])\neq resets(ctx[:EI+1])=g_{tr}(u\cdot\boldsymbol{\sigma}_r,[\![vw(ctx[EI+1:])]\!])$. Therefore, $[\![vw(ctx[EI+1:])]\!]$ can be used to distinguish $u\cdot\boldsymbol{\sigma}_r$ and $q$. The learner performs the split operation to split $q$ into $u\cdot\boldsymbol{\sigma}_r$ and $q$, adds an internal node $[\![vw(ctx[EI+1:])]\!]$ as the parent of $u\cdot\boldsymbol{\sigma}_r$ and $q$ and re-assigns the target leaf node of the leaf node jump that originally pointed to $q$.

• If $u'\neq q$, it means that in the hypothesis $\mathcal{H}$, the transition from the location $l_u$ by executing $\boldsymbol{\sigma}_r$ arrives at the incorrect location $l_q$. This is because that there is an error clock constraint on this transition. In this case, a leaf node jump $(u,\boldsymbol{\sigma}_r,u')$ is added to the set of leaf nodes relationship $T_Q$. The clock valuation in $\boldsymbol{\sigma}_r$ can be used to further refine the clock constraint.

**Example 3.** Figure 4 gives an example of the suffix analysis and the counterexample processing II. Suppose DTA $\mathcal{A}$ in Figure 1 is the target DTA. In Figure 4, $\mathcal{TR}_1$ is a classification tree generated in the learning process, and $\mathcal{M}_1$ and $\mathcal{H}_1$ are the corresponding DFA and hypothesis, respectively. An equivalence query for $\mathcal{H}_1$ yields a counterexample $\omega_r=(a,0,\{\top,\top\})(a,2,\{\top,\top\})$. Then the learner gets the corresponding reset-clocked word $ctx=(a,\{0,0\},\{\top,\top\})(a,\{2,2\},\{\top,\top\})$. Given a position $i=1$, according to the suffix analysis, the location reached by executing $(a,\{0,0\},\{\top,\top\})$ on $\mathcal{H}_1$ is $l_0$ and the leaf node corresponding to $l_0$ is $\epsilon$. Since there is a valid successor $e_r=(a,\{2,2\},\{\bot,\top\})\in vs_{\mathcal{A}}(\epsilon,[\![(a,\{2,2\})]\!])$ and $\mathsf{MQ}_{\mathcal{A}}(\epsilon\cdot(a,\{2,2\},\{\bot,\top\}))=+$, the learner gets $g_{tr}(\epsilon,[\![(a,\{2,2\})]\!])=\{\bot,\top\}$ and $f_{tr}(\epsilon,[\![(a,\{2,2\})]\!])=+$. Since $f_{tr}(\epsilon,[\![(a,\{2,2\})]\!])\neq\mathsf{MQ}_{\mathcal{A}}(ctx)$ and $g_{tr}(\epsilon,[\![(a,\{2,2\})]\!])\neq resets((a,\{2,2\},\{\top,\top\}))$, we have $EI=i=1$. Then the learner finds that the location reached by executing $ctx[EI-1]$ in $\mathcal{H}_1$ is $l_0$ and the leaf node corresponding to $l_0$ is $\epsilon$. Accordingly, the learner finds $e_r=((a,\{0,0\},\{\top,\top\}))\in vs_{\mathcal{A}}(\epsilon,[\![(a,\{0,0\})]\!])$ and then sets $\boldsymbol{\sigma}_r=e_r$. Since $sift(\epsilon\cdot(a,\{0,0\},\{\top,\top\}))=\epsilon$, the learner splits the leaf node $\epsilon$ into the leaf nodes $(a,\{0,0\},\{\top,\top\})$ and $\epsilon$, adds an internal node $[\![(a,\{2,2\})]\!]$ as the parent of $(a,\{0,0\},\{\top,\top\})$ and $\epsilon$, and then re-assigns the target leaf node of the leaf node jump that originally pointed to $\epsilon$. Since a new leaf node $(a,\{0,0\},\{\top,\top\})$ is generated in the above process, the learner adds two leaf node jumps from this new leaf node. In this way, the learner obtains a classification tree $\mathcal{TR}_2$.

## 4.2 Termination, correctness, and complexity

Supposing that $\mathcal{L}$ is a target timed language accepted by a DTA $\mathcal{A}=(\Sigma,L,l_0,F,\mathcal{C},\Delta)$, let $n=|L|$ be the number of locations of $\mathcal{A}$, $m=|\Sigma|$ be the size of the alphabet, $h$ be the maximum length of all counterexamples obtained by the learner, $\kappa$ be a function mapping each clock $c\in\mathcal{C}$ to the largest integer appearing in the constraints over $c$, $\Lambda=|\mathcal{C}|!\cdot 2^{|\mathcal{C}|}\cdot\prod_{c\in\mathcal{C}}(2\kappa(c)+2)$ be the bound of the number of regions.

**Theorem 3** (Termination and correctness)**.** Algorithm 1 can terminate and return a CTA $\mathcal{H}$ which recognizes the target timed language $\mathcal{L}$.

*Proof.* First, the correctness of the algorithm is guaranteed by the equivalence query. Now we prove termination. According to Theorem 1, we know that different reset-clocked words in different leaf nodes reach different symbolic states of $\mathcal{A}$. Hence, the number of leaf nodes in $\mathcal{TR}$ is bounded by the number of symbolic states of $\mathcal{A}$, which is $n \cdot \Lambda$. Since the locations in the hypothesis are mapped from the leaf nodes in the tree, the number of locations in the hypothesis is bounded by $n \cdot \Lambda$.

Now let us compute the number of transitions. Consider two leaf node jumps $(q, \boldsymbol{\sigma}_r, \mathit{sift}(q \cdot \boldsymbol{\sigma}_r))$ and $(q, \boldsymbol{\sigma}'_r, \mathit{sift}(q \cdot \boldsymbol{\sigma}'_r))$ from a leaf node $q$. According to Theorem 2, if $[\![vw(\boldsymbol{\sigma}_r)]\!] = [\![vw(\boldsymbol{\sigma}'_r)]\!]$, $\mathit{resets}(\boldsymbol{\sigma}_r) = \mathit{resets}(\boldsymbol{\sigma}'_r)$ and these two jumps reach the same target leaf node. So, according to the method for constructing a DFA $\mathcal{M}$ from the tree, for each $l \in L_M$ and $\sigma \in \Sigma$ in DFA $\mathcal{M}$, if there are two transitions $(l, (\sigma, \boldsymbol{v}, \boldsymbol{b}), l')$ and $(l, (\sigma, \boldsymbol{v}', \boldsymbol{b}'), l'')$ such that $[\![\boldsymbol{v}]\!] = [\![\boldsymbol{v}']\!]$, we have $l' = l''$ and $\boldsymbol{b} = \boldsymbol{b}'$. Hence, according to the method for converting $\mathcal{M}$ to $\mathcal{H}$, we know that the number of transitions from a location by performing an action in $\mathcal{H}$ is bounded by $\Lambda$. Since the number of locations in $\mathcal{H}$ and the number of actions in $\mathcal{H}$ are finite, the total number of transitions in $\mathcal{H}$ is finite. Since each iteration of the algorithm either adds at least one leaf node to $\mathcal{TR}$ or refines at least one of the partitions along with transitions of $\mathcal{H}$, or both, the algorithm is guaranteed to terminate.

**Theorem 4** (Complexity)**.** The number of equivalence queries is bounded by $mn\Lambda^2$, the number of membership queries is bounded by $mn\Lambda^2(h + n\Lambda + mn\Lambda^2)$ and the number of reset information queries is bounded by $hmn\Lambda^2(h + n\Lambda + mn\Lambda^2) + mn\Lambda$.

*Proof.* According to the proof of Theorem 3, the number of transitions in $\mathcal{H}$ is bounded by $mn\Lambda^2$. Since every counterexample adds at least one fresh transition to $\mathcal{H}$ if considering each final timing constraint of the partition corresponds to a transition (when a location is added to $\mathcal{H}$, at least $m$ transitions are added to $\mathcal{H}$), the number of both counterexamples and equivalence queries are bounded by $mn\Lambda^2$.

For each counterexample $\omega_r$, the learner converts $\omega_r$ to the corresponding reset-clocked word $\mathit{ctx} = \Gamma(\omega_r)$ and performs prefix analysis and suffix analysis for *ctx*. Let $h$ denote the maximum length of all counterexamples. To find the erroneous position in *ctx*, the learner needs to perform at most $h$ prefix analyses and $h$ suffix analyses. In a prefix analysis, the learner does not need to issue a membership query. In a suffix analysis, a membership query needs to be initiated. Hence, the number of membership queries generated to analyse *ctx* is at most $h$. After obtaining the erroneous position, to process *ctx*, the sift operation needs to be done. According to the proof of Theorem 3, there exists at most $n \cdot \Lambda$ leaf nodes in the classification tree. Hence, there are at most $n \cdot \Lambda$ internal nodes in the classification tree. Therefore, to perform a sift operation, at most $n \cdot \Lambda$ membership queries need to be issued. If the erroneous position of *ctx* is found during a prefix analysis, one sift operation is first required to process *ctx*. In this process, if a new leaf node is generated, $m$ sift operations are needed to construct leaf node jumps from the generated leaf node. Consider the case where the erroneous position of *ctx* is found during a suffix analysis. In this case, to process *ctx*, a sift operation is first required and $n\Lambda$ membership queries need to be issued. Next, in the worst case, the learner needs to split a leaf node. If so, the learner needs to reassign the target nodes in at most $mn\Lambda^2$ leaf node jumps and this process requires at most $mn\Lambda^2$ membership queries. Therefore, to analyse and process a counterexample, $h + n\Lambda + mn\Lambda^2$ membership queries are needed. Hence, the total number of membership queries is bounded by $mn\Lambda^2(h+n\Lambda+mn\Lambda^2)$.

According to [33], for a reset-clocked word $\gamma_r$ and a region word $\xi$, to find $e_r \in vs_{\mathcal{A}}(\gamma_r, \xi)$, at most $|\xi|$ reset information queries are needed. Hence, in each membership query, the number of reset information queries depends on the length of the region word stored in the internal node. Since internal nodes are always generated when processing a counterexample and are constructed from a part of the counterexample, the maximum length of the region word in an internal node is $h$. Therefore, the number of reset information queries for membership queries is bounded by $hmn\Lambda^2(h+n\Lambda+mn\Lambda^2)$. Moreover, when a new leaf node $\gamma_r$ is generated, to construct leaf node jumps from $\gamma_r$, $m$ reset information queries are needed. Hence, the total number of reset information queries is bounded by $hmn\Lambda^2(h + n\Lambda + mn\Lambda^2) + mn\Lambda$.

The bound for the equivalence queries of our algorithm is identical to that of the algorithm proposed in [33]. The number of membership queries and reset information queries required by in [33] are bounded by $n\Lambda \cdot (n\Lambda + hmn\Lambda^2 + mn\Lambda + (n\Lambda)^2)$ and $n^2\Lambda^2 \cdot (n\Lambda + hmn\Lambda^2 + mn\Lambda + (n\Lambda)^2) + mn\Lambda$, respectively. Considering such loose upper bounds, both our method and the learning algorithm in [33] exhibit exponential complexity. However, based on the tree operations and counterexample processing outlined in the paper, our method effectively reduces numerous unnecessary membership queries and also reduces equivalence queries during the learning process, a phenomenon also substantiated by our experiments.

Below we analyze how the depth of the timed classification tree impacts learning efficiency. Suppose we know that the depth of the final tree corresponding to the correct hypothesis is $D$. We first consider the effect of $D$ on the sift operation. During one sift operation, let the root node be the current node $x$. Next, the learner needs to issue a membership query to find a child $y$ of $x$ and sets $y$ to be $x$. In the worst case, this process is repeated until $x$ is a leaf node. Hence, the number of required membership queries in one sift operation is bounded by $D$. Then, consider the processes of counterexample analysis and processing. As mentioned above, for a counterexample $\omega_r$, to find the erroneous position in the corresponding reset-clocked word *ctx*, the number of membership queries generated to analyse *ctx* is at most $h$, not $D$. Consider the more complex case where the erroneous position of *ctx* is found during a suffix analysis. To process *ctx*, a sift operation is first required and the number of required membership queries is bounded by $D$. Next, in the worst case, the learner splits a leaf node. In this case, the learner needs to reassign the target nodes in at most $mn\Lambda^2$ leaf node jumps. The process of reassigning target nodes requires at most $mn\Lambda^2$ membership queries. There are at most $mn\Lambda^2$ counterexamples. Hence, the number of membership queries required for the above process is bounded by $mn\Lambda^2(h + D + mn\Lambda^2)$. Note that during the whole learning process, when a sift operation or a split operation occurs, a new leaf node may be generated. Since the number of new leaf nodes is bounded by $n\Lambda$ and each new node needs $m$ sift operations to construct leaf node jumps, the number of required membership queries in this process is bounded by $n\Lambda \cdot mD$, which is not enough to influence the complexity.

## 5 Illustrative case

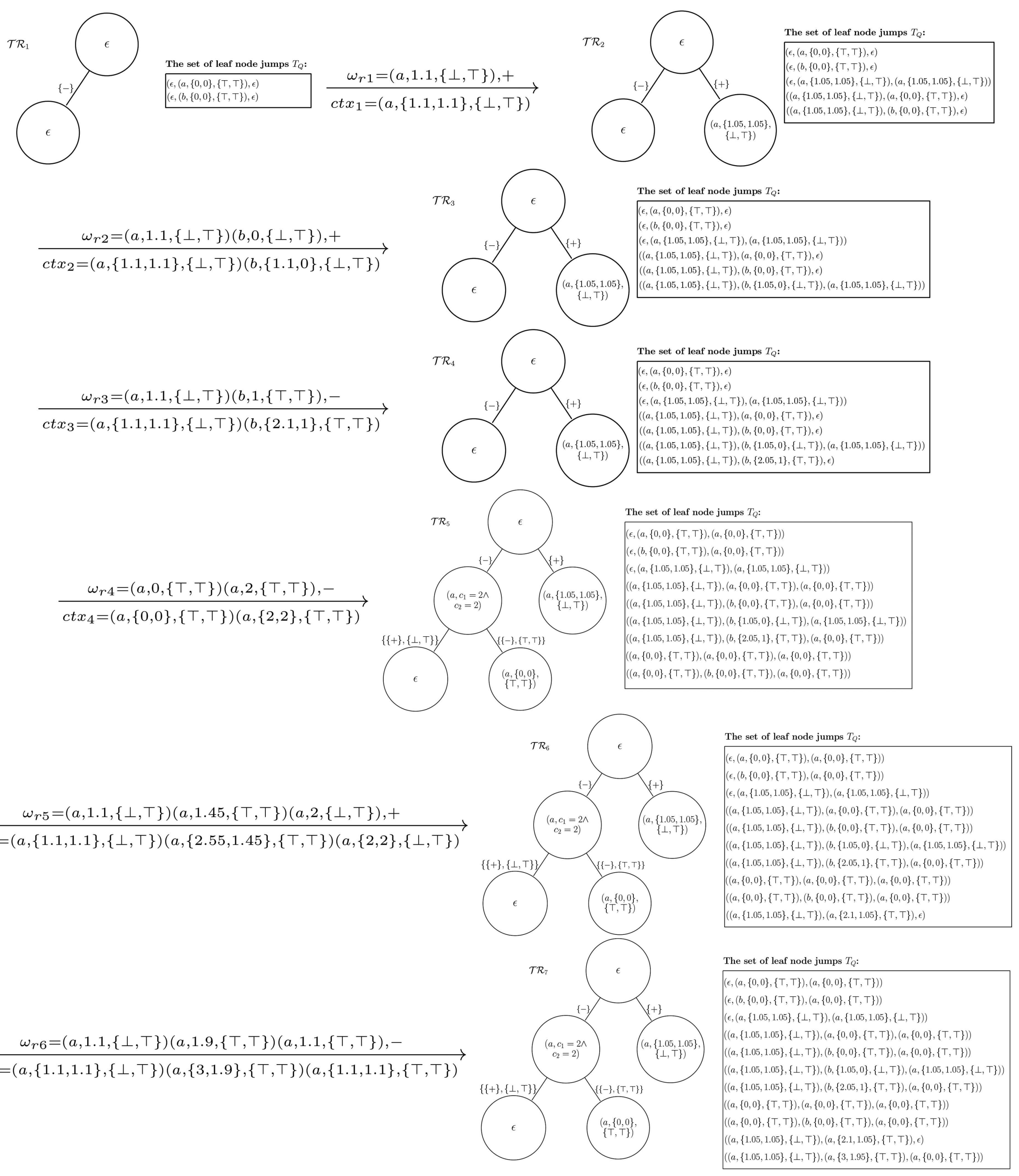
In this section, we use the DTA $\mathcal{A}$ in Figure 1 as a target automaton and show the learning process for it by using our algorithm. In the DTA $\mathcal{A}$, the initial location is $l_0$, the accepting location is $l_1$ and the alphabet $\Sigma = \{a, b\}$. Figure 5 shows the changes in the classification tree $\mathcal{TR}$ during the learning process, and Figure 6 shows the changes in the intermediate DFA $\mathcal{M}$ and hypothesis $\mathcal{H}$.

First, the learner obtains $\mathcal{TR}_1$ by the initialization operation. Then according to $\mathcal{TR}_1$, the learner builds $\mathcal{M}_1$ and $\mathcal{H}_1$ and then issues an equivalence query to get $ctx_1 = (a, \{1.1, 1.1\}, \{\bot, \top\})$. After prefix analyzing and processing $ctx_1$ through the operations in Example 2, $\mathcal{TR}_2$ is obtained. According to $\mathcal{TR}_2$, the learner builds $\mathcal{M}_2$ and $\mathcal{H}_2$. Similarly, the learner gets $ctx_2$ and $ctx_3$ by the equivalence queries and constructs $\mathcal{TR}_3$ and $\mathcal{TR}_4$, respectively. According to $\mathcal{TR}_4$, the learner builds $\mathcal{M}_4$ and $\mathcal{H}_4$ and then issues an equivalence query to acquire $ctx_4 = (a, \{0, 0\}, \{\top, \top\})\ (a, \{2, 2\}, \{\top, \top\})$. After suffix analyzing and processing $ctx_4$ through the operations in Example 3, $\mathcal{TR}_5$ is obtained and $\mathcal{M}_5$ and $\mathcal{H}_5$ are constructed.

After an equivalence query, a new counterexample $ctx_5 = (a, \{1.1, 1.1\}, \{\bot, \top\})(a, \{2.55, 1.45\}, \{\top, \top\})$ $(a, \{2, 2\}, \{\bot, \top\})$ is obtained. According to the suffix analysis for $ctx_5$, the learner finds $EI = 2$. The computation details are as follows. The location reached by executing $(a, \{1.1, 1.1\}, \{\bot, \top\})(a, \{2.55, 1.45\}, \{\top, \top\})$ in $\mathcal{H}_5$ is $l_2$ and it corresponds to the leaf node $(a, \{0, 0\}, \{\top, \top\})$. However, $f_{tr}((a, \{0, 0\}, \{\top, \top\}), [\![(a, \{2, 2\})]\!]) \neq \mathsf{MQ}_{\mathcal{A}}(ctx_5)$ and $g_{tr}((a, \{0, 0\}, \{\top, \top\}), [\![(a, \{2, 2\})]\!]) \neq resets((a, \{2, 2\}, \{\bot, \top\}))$. Hence, $EI = 2$. To process $ctx_5$, the learner finds that the location reached by executing $ctx_5[EI - 1]$ in $\mathcal{H}_5$ is $l_1$ and the leaf node corresponding to $l_1$ is $(a, \{1.05, 1.05\}, \{\bot, \top\})$. Then the learner finds a valid successor $e_r = ((a, \{2.1, 1.05\}, \{\top, \top\})) \in vs_{\mathcal{A}}((a, \{1.05, 1.05\}, \{\bot, \top\}), [\![(a, \{2, 55, 1.45\})]\!])$. Since $sift((a, \{1.05, 1.05\}, \{\bot, \top\})((a, \{2.1, 1.05\}, \{\top, \top\}))) = \epsilon \neq (a, \{0, 0\}, \{\top, \top\})$, a new leaf node jump $((a, \{1.05, 1.05\}, \{\bot, \top\}), (a, \{2.1, 1.05\}, \{\top, \top\}), \epsilon)$ is added to $T_Q$. Similarly, whenever the learner gets a counterexample through the equivalence query, the counterexample needs to be analyzed and processed to get a new classification tree. According to the new tree, a new DFA and a new hypothesis need to be constructed to issue an equivalence query again. The learning algorithm terminates until the teacher gives a positive answer to the equivalence query.

## 6 Experiments

We implemented a prototype named DTAL-Tree in JAVA for our learning algorithm. We compared it with an active learning algorithm DTAL-Table proposed in [33] for learning DTAs from a powerful teacher, an active learning algorithm DOTAL proposed in [28] for learning deterministic one-clock timed automata (DOTAs) from a powerful teacher, a passive learning algorithm GenProg based on genetic programming in [12] and an algorithm AcGenProg in [42] improving on GenProg, respectively. All evaluations were carried out on an Intel Core-i7 processor with 32 GB RAM.



𝒯ℛ1
ϵ
{−}
ϵ
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),ϵ)
(ϵ,(b,{0,0},{⊤,⊤}),ϵ)
ω_r1=(a,1.1,{⊥,⊤}),+
ctx1=(a,{1.1,1.1},{⊥,⊤})
𝒯ℛ2
{−}
{+}
(a,{1.05,1.05},{⊥,⊤})
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),ϵ)
(ϵ,(b,{0,0},{⊤,⊤}),ϵ)
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),ϵ)
ω_r2=(a,1.1,{⊥,⊤})(b,0,{⊥,⊤}),+
ctx2=(a,{1.1,1.1},{⊥,⊤})(b,{1.1,0},{⊥,⊤})
𝒯ℛ3
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),ϵ)
(ϵ,(b,{0,0},{⊤,⊤}),ϵ)
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(b,{1.05,0},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
ω_r3=(a,1.1,{⊥,⊤})(b,1,{⊤,⊤}),−
ctx3=(a,{1.1,1.1},{⊥,⊤})(b,{2.1,1},{⊤,⊤})
𝒯ℛ4
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),ϵ)
(ϵ,(b,{0,0},{⊤,⊤}),ϵ)
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(b,{1.05,0},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{2.05,1},{⊤,⊤}),ϵ)
ω_r4=(a,0,{⊤,⊤})(a,2,{⊤,⊤}),−
ctx4=(a,{0,0},{⊤,⊤})(a,{2,2},{⊤,⊤})
𝒯ℛ5
(a, c1 = 2∧ c2 = 2)
{{+},{⊥,⊤}}
{{−},{⊤,⊤}}
(a,{0,0},{⊤,⊤})
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{1.05,0},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{2.05,1},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
ω_r5=(a,1.1,{⊥,⊤})(a,1.45,{⊤,⊤})(a,2,{⊥,⊤}),+
ctx5=(a,{1.1,1.1},{⊥,⊤})(a,{2.55,1.45},{⊤,⊤})(a,{2,2},{⊥,⊤})
𝒯ℛ6
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{1.05,0},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{2.05,1},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{2.1,1.05},{⊤,⊤}),ϵ)
ω_r6=(a,1.1,{⊥,⊤})(a,1.9,{⊤,⊤})(a,1.1,{⊤,⊤}),−
ctx6=(a,{1.1,1.1},{⊥,⊤})(a,{3,1.9},{⊤,⊤})(a,{1.1,1.1},{⊤,⊤})
𝒯ℛ7
The set of leaf node jumps T_Q:
(ϵ,(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
(ϵ,(a,{1.05,1.05},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{1.05,0},{⊥,⊤}),(a,{1.05,1.05},{⊥,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(b,{2.05,1},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{0,0},{⊤,⊤}),(b,{0,0},{⊤,⊤}),(a,{0,0},{⊤,⊤}))
((a,{1.05,1.05},{⊥,⊤}),(a,{2.1,1.05},{⊤,⊤}),ϵ)
((a,{1.05,1.05},{⊥,⊤}),(a,{3,1.95},{⊤,⊤}),(a,{0,0},{⊤,⊤}))

**Figure 5** Iterations of the timed classification tree during the learning process of DTA $\mathcal{A}$ in Figure 1.

## 6.1 Compared with DTAL-Table

We first generated 100 DTAs with 2 clocks randomly as the targets, evenly distributed across 10 groups based on varying numbers of locations, alphabet sizes, and maximum constants in constraints[1]. These three parameters constitute the case ID. Table 1 provides a summary of the experimental results. #Membership, #Equivalence, and #Reset denote the average number of membership queries, equivalence queries, and reset information queries, respectively. #M_opt represents the average number of membership queries conducted in tree operations or table operations, while #M_ctx represents the average number of membership queries conducted during counterexample analysis. $n$ denotes the average number of locations in the learned automata. Time (s) denotes the average runtime for learning a DTA, with a time limit of 300 s per automaton. #Learnt represents the total number of learned automata.

In comparison with DTAL-Table, our method DTAL-Tree necessitates notably fewer membership queries and equivalence queries. We provide a detailed breakdown of the membership queries employed in both

1) This experiment is available at https://github.com/Tongji-lab/DTA-Learn.

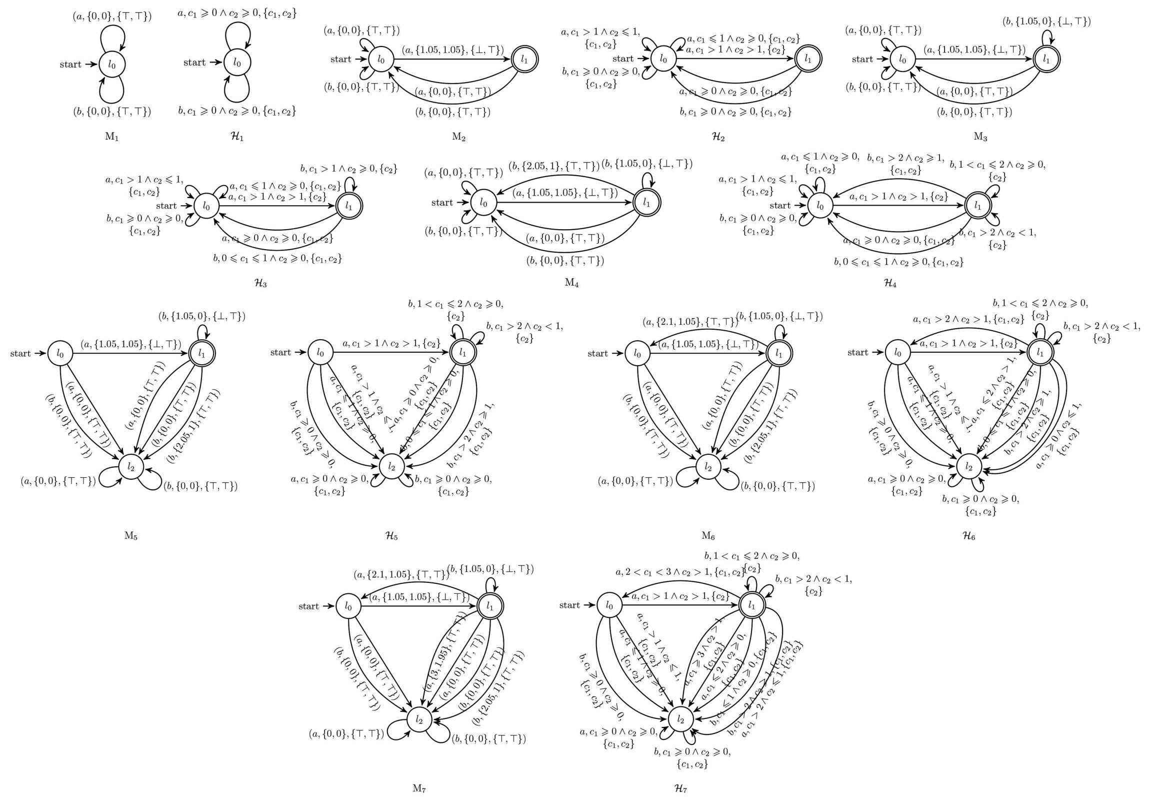


**Figure 6** Iterations of intermediate DFA $\mathcal{M}$ and hypothesis $\mathcal{H}$ during the learning process of DTA $\mathcal{A}$ in Figure 1.

**Table 1** Experimental results on learning DTAs. Each row represents one DTA group where $|L|$ is the number of locations, $|\Sigma|$ is the alphabet size, and $\kappa(\mathcal{C})$ is the maximum constant in the clock constraints. The best results are in bold.

| $\lvert L\rvert$ | $\lvert\Sigma\rvert$ | $\kappa(\mathcal{C})$ | $\lvert\mathcal{C}\rvert$ | Algorithm | #Membership | #M_opt | #M_ctx | #Equivalence | #Reset | $n$ | Time (s) | #Learnt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 1 | 8 | 2 | DTAL-Tree | **45.5** | **16.3** | **22.3** | **6.4** | 62.0 | **5.4** | **0.7** | 10/10 |
| | | | | DTAL-Table | 52.4 | 24.3 | 26.1 | 8.9 | **49.0** | 5.5 | 1.0 | 10/10 |
| 8 | 1 | 8 | 2 | DTAL-Tree | **81.2** | **32.0** | **40.2** | **9.2** | 138.1 | 8.3 | **1.3** | 10/10 |
| | | | | DTAL-Table | 130.9 | 53.6 | 75.3 | 15.5 | **134.1** | **7.4** | 2.9 | 10/10 |
| 8 | 2 | 8 | 2 | DTAL-Tree | **109.0** | **50.0** | **48.7** | **11.4** | **180.1** | 7.8 | **2.1** | 10/10 |
| | | | | DTAL-Table | 195.2 | 85.9 | 106.3 | 17.3 | 242.2 | **7.1** | 7.2 | 10/10 |
| 8 | 2 | 16 | 2 | DTAL-Tree | **130.9** | **52.5** | **60.2** | **14.4** | 218.1 | 9.8 | **2.9** | 10/10 |
| | | | | DTAL-Table | 180.3 | 79.6 | 97.7 | 19.3 | **180.3** | **7.3** | 4.9 | 10/10 |
| 8 | 3 | 8 | 2 | DTAL-Tree | **241.2** | **111.7** | **93.2** | **24.1** | **366.2** | 11.1 | **3.9** | 10/10 |
| | | | | DTAL-Table | 379.3 | 171.0 | 204.3 | 30.0 | 495.2 | **8.4** | 18.8 | 10/10 |
| 10 | 2 | 8 | 2 | DTAL-Tree | **174.0** | **71.8** | **81.6** | **15.1** | **455.1** | 10.7 | **4.9** | 10/10 |
| | | | | DTAL-Table | 430.7 | 160.8 | 266.9 | 24.7 | 558.2 | **9.4** | 38.5 | 10/10 |
| 10 | 2 | 16 | 2 | DTAL-Tree | **181.0** | **67.4** | **95.4** | **19.8** | 408.6 | 10.8 | **6.5** | 10/10 |
| | | | | DTAL-Table | 312.8 | 106.7 | 203.1 | 29.8 | **347.7** | **8.4** | 73.2 | 9/10 |
| 10 | 3 | 8 | 2 | DTAL-Tree | **256.2** | **115.7** | **113.9** | **27.7** | 414.2 | 13.6 | **5.9** | 10/10 |
| | | | | DTAL-Table | 407.1 | 173.0 | 230.1 | 36.3 | **392.4** | **9.6** | 42.7 | 10/10 |
| 12 | 2 | 8 | 2 | DTAL-Tree | **276.9** | **104.5** | **148.5** | **25.2** | 720.1 | 14.1 | **7.5** | 10/10 |
| | | | | DTAL-Table | 418.7 | 136.7 | 279.0 | 30.7 | **542.0** | **10.7** | 36.0 | 9/10 |
| 14 | 2 | 8 | 2 | DTAL-Tree | **225.7** | **108.5** | **100.8** | **22.4** | 519.0 | 13.4 | **9.3** | 10/10 |
| | | | | DTAL-Table | 450.2 | 173.8 | 273.4 | 28.5 | **509.1** | **10.8** | 122.2 | 10/10 |

operations and counterexample processing. Remarkably, our approach reduces the requisite membership queries in both procedures, and avoids time-consuming checking of "closed" and "consistent", resulting in substantial enhancements in terms of runtime. Regarding the number of reset information queries, our

**Table 2** Experimental results on learning DTAs.

| $\|L\|$ | $\|\Sigma\|$ | $\kappa(\mathcal{C})$ | $\|\mathcal{C}\|$ | Algorithm | #Membership | #M_opt | #M_ctx | #Equivalence | #Reset | $n$ | Time (s) | #Learnt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 1 | 8 | 3 | DTAL-Tree | 45.7 | 18.1 | 19.5 | 6.2 | 62.9 | 6.6 | 10.5 | 10/10 |
| | | | | DTAL-Table | 35.0 | 24.0 | 9.0 | 7 | 27 | 5 | 1.8 | 1/10 |
| 6 | 2 | 8 | 3 | DTAL-Tree | 76.8 | 36.1 | 31.1 | 9.0 | 98.3 | 8.9 | 4.8 | 7/10 |
| 6 | 1 | 16 | 3 | DTAL-Tree | 58.3 | 24.5 | 26.8 | 7.9 | 69.8 | 8.8 | 6.2 | 8/10 |
| 8 | 1 | 8 | 3 | DTAL-Tree | 90.0 | 31.0 | 48.4 | 13.0 | 128.3 | 9.7 | 33.1 | 10/10 |

**Table 3** Experimental results on learning practical models in [31, 32]. 'TO' represents a timeout (300 s).

| | $\|L\|$ | $\|\Sigma\|$ | $\kappa(\mathcal{C})$ | $\|\mathcal{C}\|$ | Algorithm | #Membership | #M_opt | #M_ctx | #Equivalence | #Reset | $n$ | Time (s) | #Learnt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AKM | 17 | 12 | 5 | 1 | DTAL-Tree | **515** | **337** | **138** | **33** | **695** | **13** | **2.1** | 1/1 |
| | | | | | DTAL-Table | 2256 | 1934 | 309 | 36 | 2976 | 13 | 96.9 | 1/1 |
| TCP | 22 | 13 | 2 | 1 | DTAL-Tree | **589** | **389** | **155** | 29 | **912** | **21** | **2.3** | 1/1 |
| | | | | | DTAL-Table | 4592 | 4298 | 280 | **26** | 6013 | 21 | 235.2 | 1/1 |
| CAS | 14 | 10 | 27 | 1 | DTAL-Tree | **339** | **232** | **82** | **14** | **541** | **15** | **0.6** | 1/1 |
| | | | | | DTAL-Table | 1650 | 1536 | 103 | 14 | 1800 | 15 | 23.0 | 1/1 |
| PC | 26 | 17 | 10 | 1 | DTAL-Tree | **745** | **600** | **104** | **26** | **1004** | **16** | **3.7** | 1/1 |
| | | | | | DTAL-Table | TO | TO | TO | TO | TO | TO | TO | 0/1 |

**Table 4** Experimental results on learning DOTAs.

| $\|L\|$ | $\|\Sigma\|$ | $\kappa(\mathcal{C})$ | Algorithm | #Membership | #M_opt | #M_ctx | #Equivalence | #Reset | $n$ | Time (s) | #Learnt |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 20 | DTAL-Tree | 220.6 | 142.9 | 72.3 | 28.2 | 219.5 | 5.0 | 0.3 | 10/10 |
| | | | DOTAL | 245.0 | 120.1 | 120.4 | 30.1 | – | 5.5 | 25.8 | 10/10 |
| 7 | 4 | 10 | DTAL-Tree | 401.4 | 275.7 | 119.9 | 41.6 | 465.7 | 8.0 | 0.7 | 10/10 |
| | | | DOTAL | 921.7 | 530.9 | 386.8 | 50.9 | – | 10.3 | 37.8 | 10/10 |
| 7 | 4 | 20 | DTAL-Tree | 437.0 | 295.0 | 135.0 | 43.3 | 472.6 | 8.0 | 0.4 | 10/10 |
| | | | DOTAL | 634.5 | 386.8 | 243.0 | 44.7 | – | 8.8 | 43.3 | 10/10 |
| 7 | 6 | 10 | DTAL-Tree | 587.8 | 416.0 | 160.0 | 55.9 | 671.6 | 8.0 | 0.8 | 10/10 |
| | | | DOTAL | 1183.4 | 708.0 | 469.2 | 70.5 | – | 10.5 | 90.2 | 10/10 |
| 10 | 4 | 20 | DTAL-Tree | 766.1 | 550.6 | 209.7 | 66.6 | 824.1 | 11.0 | 1.5 | 10/10 |
| | | | DOTAL | 1580.9 | 983.7 | 592.7 | 73.1 | – | 12.7 | 166.5 | 10/10 |
| 14 | 4 | 20 | DTAL-Tree | 1134.7 | 798.0 | 329.3 | 99.6 | 1230.4 | 15.0 | 5.6 | 10/10 |
| | | | DOTAL | 1572.0 | 938.0 | 629.0 | 79.0 | – | 16.0 | 255.0 | 1/10 |

method aligns closely with DTAL-Table. Despite the searching strategy employed in DTAL-Table, which typically yields a smaller-sized learned automaton, the disparity is not exceedingly significant.

We then present the experimental results on randomly generated DTAs with more clocks. DTAL-Table is struggling to learning DTA with more clocks. It fails to learn out a DTA in most of the cases; thus we omit its experimental results in Table 2. We also compared two methods on learning practical models which have been utilized in [31, 32]. Table 3 shows that our method DTAL-Tree improves DTAL-Table significantly.

## 6.2 Compared with DOTAL

We also compared our method with the active learning algorithm DOTAL by learning DOTAs. We used the 60 randomly generated DOTAs from [28], which are divided into 6 groups according to different numbers of locations $|L|$, size of alphabet $|\Sigma|$, and the maximum constant appearing in the clock constraints $\kappa(\mathcal{C})$. #Membership, #Equivalence, #Reset, #M_opt, #M_ctx, $n$, Time (s) and #Learnt have the same meaning as mentioned above. The results are shown in Table 4. It shows that our method needs fewer membership queries. Furthermore, the runtime needed by using our method is much shorter. There are two reasons for this phenomenon. On the one hand, it is because our method avoids time-consuming checking of "closed" and "consistent" and reduces the membership queries. On the other hand, it is since our algorithm is implemented in Java, while DOTAL is implemented in Python.

**Table 5** Experimental results produced by comparing DTAL-Tree with GenProg.

| | $\|L\|$ | $\|\Sigma\|$ | $\kappa(\mathcal{C})$ | $\|\mathcal{C}\|$ | Algorithm | $n$ | Time (s) | #Learnt |
|---|---|---|---|---|---|---|---|---|
| Light | 5 | 5 | 10 | 1 | DTAL-Tree | **6** | **0.1** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| Train | 6 | 6 | 10 | 1 | DTAL-Tree | 7 | **0.1** | 1/1 |
| | | | | | GenProg | **6** | 115.3 | 1/1 |
| CAS | 14 | 10 | 27 | 1 | DTAL-Tree | **15** | **0.6** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| PC | 26 | 17 | 10 | 1 | DTAL-Tree | **16** | **3.7** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| – | 6 | 7 | 13 | 2 | DTAL-Tree | **9** | **1.5** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| – | 6 | 5 | 15 | 2 | DTAL-Tree | **7** | **0.6** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| – | 6 | 6 | 14 | 2 | DTAL-Tree | **14** | **3.9** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |
| – | 6 | 4 | 15 | 2 | DTAL-Tree | **7** | **3.4** | 1/1 |
| | | | | | GenProg | TO | TO | 0/1 |

**Table 6** Experimental results produced by comparing DTAL-Tree with AcGenProg.

| | $\|L\|$ | $\|\Sigma\|$ | $\kappa(\mathcal{C})$ | $\|\mathcal{C}\|$ | Algorithm | $n$ | Time(s) | #Learnt |
|---|---|---|---|---|---|---|---|---|
| Light | 5 | 5 | 10 | 1 | DTAL-Tree | 6 | **0.1** | 1/1 |
| | | | | | AcGenProg | **5** | 29.6 | 1/1 |
| CAS | 14 | 10 | 27 | 1 | DTAL-Tree | **15** | **0.6** | 1/1 |
| | | | | | AcGenProg | TO | TO | 0/1 |
| PC | 26 | 17 | 10 | 1 | DTAL-Tree | **16** | **3.7** | 1/1 |
| | | | | | AcGenProg | TO | TO | 0/1 |

### 6.3 Compared with learning algorithms based on genetic programming

Below we compare our algorithm with two learning algorithms based on genetic programming, rather than the $L^*$ framework. The first algorithm GenProg is a passive learning algorithm. The second learning algorithm AcGenProg improves on GenProg. We compared our method and GenProg by learning 4 practical models and 4 randomly generated DTAs containing 2 clocks. The results are shown in Table 5. Since GenProg does not involve queries, we only compared the runtime Time (s) and the number of locations $n$ in the learned automata. The experimental result shows that the size of the automata learned by two algorithms is similar, and our algorithm greatly reduces the runtime required for learning. In addition, we compared our method and AcGenProg by learning 3 practical models. The results are shown in Table 6 and we obtained the same conclusion as those of the previous set of comparative experiments.

## 7 Conclusion

We introduce the timed classification tree and devise an active learning algorithm for DTA based on it. Compared with the method in [33], our algorithm significantly decreases the number of queries and reduces the runtime. We plan to refine the algorithm further for learning from a normal teacher.

**Acknowledgements** This work was supported by Shanghai 2023 "Science and Technology Innovation Action Plan": Special Project for Key Technical Breakthrough of Blockchain (Grant No. 23511100800) and National Natural Science Foundation of China (Grant No. 62472316).

**References**

1 Vaandrager F. Model learning. Commun ACM, 2017, 60: 86–95
2 Gold E M. Complexity of automaton identification from given data. Inf Control, 1978, 37: 302–320
3 Oliveira A L, Silva J P M. Efficient algorithms for the inference of minimum size DFAs. Machine Learn, 2001, 44: 93–119
4 Avellaneda F, Petrenko A. Learning minimal DFA: taking inspiration from RPNI to improve SAT approach. In: Proceedings of International Conference on Software Engineering and Formal Methods, 2019. 243–256

5 Bohn L, Löding C. Passive learning of deterministic Büchi automata by combinations of DFAs. In: Proceedings of International Colloquium on Automata, Languages, and Programming, 2022. 1–20
6 Schmidt J, Ghorbani A, Hapfelmeier A, et al. Learning probabilistic real-time automata from multi-attribute event logs. Intell Data Anal, 2013, 17: 93–123
7 Cornanguer L, Largouët C, Rozé L, et al. TAG: learning timed automata from logs. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2022. 3949–3958
8 Verwer S, de Weerdt M, Witteveen C. One-clock deterministic timed automata are efficiently identifiable in the limit. In: Proceedings of International Conference on Language and Automata Theory and Applications, 2009. 740–751
9 Verwer S, Weerdt M, Witteveen C. The efficiency of identifying timed automata and the power of clocks. Inf Comput, 2011, 209: 606–625
10 Verwer S, de Weerdt M, Witteveen C. Efficiently identifying deterministic real-time automata from labeled data. Mach Learn, 2012, 86: 295–333
11 Dierl S, Howar F M, Kauffman S, et al. Learning symbolic timed models from concrete timed data. In: Proceedings of NASA Formal Methods Symposium, 2023. 104–121
12 Tappler M, Aichernig B K, Larsen K G, et al. Time to learn-learning timed automata from tests. In: Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems, 2019. 216–235
13 Tappler M, Aichernig B K, Lorber F. Timed automata learning via SMT solving. In: Proceedings of NASA Formal Methods Symposium, 2022. 489–507
14 Jin X, An J, Zhan B, et al. Inferring switched nonlinear dynamical systems. Form Asp Comput, 2021, 33: 385–406
15 Angluin D. Learning regular sets from queries and counterexamples. Inf Comput, 1987, 75: 87–106
16 Aarts F, Kuppens H, Tretmans J, et al. Improving active Mealy machine learning for protocol conformance testing. Mach Learn, 2014, 96: 189–224
17 Shahbaz M, Groz R. Inferring Mealy machines. In: Proceedings of FM, 2009. 207–222
18 Bollig B, Habermehl P, Kern C, et al. Angluin-style learning of NFA. In: Proceedings of International Joint Conference on Artificial Intelligence, 2009. 1004–1009
19 Howar F, Steffen B, Jonsson B, et al. Inferring canonical register automata. In: Proceedings of International Workshop on Verification, Model Checking, and Abstract Interpretation, 2012. 251–266
20 Cassel S, Howar F, Jonsson B, et al. Active learning for extended finite state machines. Form Asp Comput, 2016, 28: 233–263
21 Aarts F, Fiterau-Brostean P, Kuppens H, et al. Learning register automata with fresh value generation. In: Proceedings of International Colloquium on Theoretical Aspects of Computing, 2015. 165–183
22 Aarts F, Vaandrager F. Learning I/O automata. In: Proceedings of International Conference on Concurrency Theory, 2010. 71–85
23 Maler O, Mens I E. Learning regular languages over large alphabets. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2014. 485–499
24 Argyros G, D'Antoni L. The learnability of symbolic automata. In: Proceedings of International Conference on Computer Aided Verification, 2018. 427–445
25 Vaandrager F, Bloem R, Ebrahimi M. Learning Mealy machines with one timer. In: Proceedings of International Conference on Language and Automata Theory and Applications, 2021. 157–170
26 An J, Wang L T, Zhan B H, et al. Learning real-time automata. Sci China Inf Sci, 2021, 64: 192103
27 An J, Zhan B, Zhan N, et al. Learning nondeterministic real-time automata. ACM Trans Embed Comput Syst, 2021, 20: 1–26
28 An J, Chen M, Zhan B, et al. Learning one-clock timed automata. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2020. 444–462
29 Shen W, An J, Zhan B, et al. PAC learning of deterministic one-clock timed automata. In: Proceedings of International Conference on Formal Engineering Methods, 2020. 129–146
30 Tang X, Shen W, Zhang M, et al. Learning deterministic one-clock timed automata via mutation testing. In: Proceedings of International Symposium on Automated Technology for Verification and Analysis, 2022. 233–248
31 Xu R, An J, Zhan B. Active learning of one-clock timed automata using constraint solving. In: Proceedings of International Symposium on Automated Technology for Verification and Analysis, 2022. 249–265
32 Waga M. Active learning of deterministic timed automata with Myhill-Nerode style characterization. In: Proceedings of International Conference on Computer Aided Verification, 2023. 3–26
33 Teng Y, Zhang M, An J. Learning deterministic multi-clock timed automata. In: Proceedings of International Conference on Hybrid Systems: Computation and Control, 2024. 1–11
34 Isberner M, Howar F, Steffen B. The TTT algorithm: a redundancy-free approach to active automata learning. In: Proceedings of International Conference on Runtime Verification, 2014. 307–322
35 Li Y, Chen Y F, Zhang L, et al. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. Inf Computation, 2021, 281: 104678
36 Vaandrager F W, Garhewal B, Rot J, et al. A new approach for active automata learning based on apartness. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2022. 223–243
37 Dierl S, Fiterau-Brostean P, Howar F, et al. Scalable tree-based register automata learning. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2024. 87–108
38 Mi J, Xu J. The learning algorithm of real-time automata. In: Proceedings of International Conference on Computer and Communications, 2020. 2146–2150
39 Mi J, Zhang M, An J, et al. Learning deterministic one-clock timed automata based on timed classification tree. J Software, 2022, 33: 2797–2814
40 Alur R, Dill D L. A theory of timed automata. Theor Comput Sci, 1994, 126: 183–235
41 Maler O, Pnueli A. On recognizable timed languages. In: Proceedings of International Conference on Foundations of Software Science and Computation Structures, 2004. 348–362
42 Aichernig B K, Pferscher A, Tappler M. From passive to active: learning timed automata efficiently. In: Proceedings of NASA Formal Methods Symposium, 2020. 1–19