Active Learning of Deterministic Timed Automata with Myhill-Nerode Style Characterization

Masaki Waga[*]

Graduate School of Informatics, Kyoto University, Kyoto, Japan

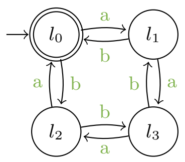
Abstract. We present an algorithm to learn a deterministic timed automaton (DTA) via membership and equivalence queries. Our algorithm is an extension of the L* algorithm with a Myhill-Nerode style characterization of recognizable timed languages, which is the class of timed languages recognizable by DTAs. We first characterize the recognizable timed languages with a Nerode-style congruence. Using it, we give an algorithm with a smart teacher answering *symbolic* membership queries in addition to membership and equivalence queries. With a symbolic membership query, one can ask the membership of a certain set of timed words at one time. We prove that for any recognizable timed language, our learning algorithm returns a DTA recognizing it. We show how to answer a symbolic membership query with finitely many membership queries. We also show that our learning algorithm requires a polynomial number of queries with a smart teacher and an exponential number of queries with a normal teacher. We applied our algorithm to various benchmarks and confirmed its effectiveness with a normal teacher.

Keywords: timed automata, active automata learning, recognizable timed languages, L* algorithm, observation table

1 Introduction

Active automata learning is a class of methods to infer an automaton recognizing an unknown target language $\mathcal{L}_{\mathrm{tgt}} \subseteq \Sigma^*$ through finitely many queries to a teacher. The L* algorithm [10], the best-known active DFA learning algorithm, infers the minimum DFA recognizing $\mathcal{L}_{\mathrm{tgt}}$ using *membership* and *equivalence* queries. In a membership query, the learner asks if a word $w \in \Sigma^*$ is in the target language $\mathcal{L}_{\mathrm{tgt}}$, which is used to obtain enough information to construct a hypothesis DFA $\mathcal{A}_{\mathrm{hyp}}$. Using an equivalence query, the learner checks if the hypothesis $\mathcal{A}_{\mathrm{hyp}}$ recognizes the target language $\mathcal{L}_{\mathrm{tgt}}$. If $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \neq \mathcal{L}_{\mathrm{tgt}}$, the teacher returns a counterexample $cex \in \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ differentiating the target language and the current hypothesis. The learner uses cex to update $\mathcal{A}_{\mathrm{hyp}}$ to

[*] This is the author (and extended) version of the manuscript of the same name published in the proceedings of the 35th International Conference on Computer Aided Verification (CAV 2023). The final version is available at www.springer.com.



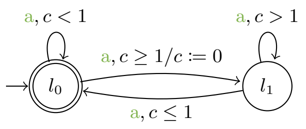
(a) A DFA $\mathcal{A}$

	ε	a
ε	$\top$	$\bot$
a	$\bot$	$\bot$
$\vdots$		
aa	$\bot$	$\bot$

$\xrightarrow{\text{add b to } S}$

	ε	a	b
ε	$\top$	$\bot$	$\bot$
a	$\bot$	$\bot$	$\top$
$\vdots$			
aa	$\bot$	$\bot$	$\bot$

(b) Intermediate observation tables for learning $\mathcal{A}$. a and aa are deemed equivalent with extensions $S = \{\varepsilon, \text{a}\}$ but distinguished with $S = \{\varepsilon, \text{a}, \text{b}\}$.



(c) A DTA $\mathcal{A}'$ with one clock variable c

	$\{\tau'_0 \mid \tau'_0 = 0\}$	$\{\tau'_0\text{a} \mid \tau'_0 \in (0,1)\}$
$\{\tau_0 \mid \tau_0 = 0\}$	$\top$	$\top$
$\{\tau_0 \mid \tau_0 \in (0,1)\}$	$\top$	$\tau_0 + \tau'_0 \in (0,1)$
$\vdots$		
$\{\tau_0\text{a}\tau_1 \mid \tau_0 \in (0,1), \tau_1 \in (0,1), \tau_0 + \tau_1 \in (0,1)\}(= p_1)$	$\top$	$\tau_0 + \tau_1 + \tau'_0 \in (0,1)$
$\vdots$		
$\{\tau_0\text{a}\tau_1\text{a}\tau_2 \mid \tau_0 \in (1,2), \tau_1 \in (0,1), \tau_2 \in (0,1), \tau_1 + \tau_2 \in (0,1)\}(= p_2)$	$\top$	$\tau_1 + \tau_2 + \tau'_0 \in (0,1)$
$\vdots$		

(d) Timed observation table for learning $\mathcal{A}'$. Each cell is indexed by a pair $(p, s) \in P \times S$ of elementary languages. The cell indexed by (p, s) shows a constraint Λ such that $w \in p \cdot s$ satisfies $w \in \mathcal{L}_{\text{tgt}}$ if and only if Λ holds. Elementary languages p_1 and p_2 are deemed equivalent with the equation $\tau_0^1 + \tau_1^1 = \tau_1^2 + \tau_2^2$, where τ_i^j represents τ_i in p_j.

Fig. 1: Illustration of observation tables in the L* algorithm for DFA learning (Fig. 1b) and our algorithm for DTA learning (Fig. 1d)

classify *cex* correctly. Such a learning algorithm has been combined with formal verification, e. g., for testing [32,30,38,35] and controller synthesis [40].

Most of the DFA learning algorithms rely on the characterization of regular languages by *Nerode's congruence*. For a language $\mathcal{L}$, words p and p' are equivalent if for any extension s, $p \cdot s \in \mathcal{L}$ if and only if $p' \cdot s \in \mathcal{L}$. It is well known that if $\mathcal{L}$ is regular, such an equivalence relation has finite classes, corresponding to the locations of the minimum DFA recognizing $\mathcal{L}$ (known as *Myhill-Nerode theorem*; see, e. g., [26]). Moreover, for any regular language $\mathcal{L}$, there are finite extensions S such that p and p' are equivalent if and only if for any $s \in S$, $p \cdot s \in \mathcal{L}$ if and only if $p' \cdot s \in \mathcal{L}$. Therefore, one can learn the minimum DFA by learning such finite extensions S and the finite classes induced by Nerode's congruence.

The L* algorithm learns the minimum DFA recognizing the target language $\mathcal{L}_{\text{tgt}}$ using a 2-dimensional array called an *observation table*. Fig. 1b illustrates observation tables. The rows and columns of an observation table are indexed with finite sets of words P and S, respectively. Each cell indexed by $(p, s) \in P \times S$ shows if $p \cdot s \in \mathcal{L}_{\text{tgt}}$. The column indices S are the current extensions approximating Nerode's congruence. The L* algorithm increases P and S until: 1) the equivalence relation defined by S converges to Nerode's congruence and 2) P covers all the classes induced by the congruence. The equivalence between $p, p' \in P$ under S can be checked by comparing the rows in the observation table indexed with p and p'. For example, Fig. 1b shows that a and aa are deemed equivalent with extensions $S = \{\varepsilon, \text{a}\}$ but distinguished by adding b to S. The refinement of P and S is driven by certain conditions to validate the DFA construction and by addressing the counterexample obtained by an equivalence query.

Timed words are extensions of conventional words with real-valued dwell time between events. *Timed* languages, sets of timed words, are widely used to formalize real-time systems and their properties, e. g., for formal verification. Among various formalisms representing timed languages, *timed automata (TAs)* [6] is one of the widely used formalisms. A TA is an extension of an NFA with finitely many clock variables to represent timing constraints. Fig. 1c shows an example.

Despite its practical relevance, learning algorithms for TAs are only available for limited subclasses of TAs, e. g., real-time automata [8,9], event-recording automata [21,22], event-recording automata with unobservable reset [24], and one-clock deterministic TAs [7,39]. Timing constraints representable by these classes are limited, e. g., by restricting the number of clock variables or by restricting the edges where a clock variable can be reset. Such restriction simplifies the inference of timing constraints in learning algorithms.

Contributions In this paper, we propose an active learning algorithm for *deterministic* TAs (DTAs). The languages recognizable by DTAs are called *recognizable timed languages* [29]. Our strategy is as follows: first, we develop a Myhill-Nerode style characterization of recognizable timed languages; then, we extend the L* algorithm for recognizable timed languages using the similarity of the Myhill-Nerode style characterization.

Due to the continuity of dwell time in timed words, it is hard to characterize recognizable timed languages by a Nerode-style congruence between timed words. For example, for the DTA in Fig. 1c, for any $\tau, \tau' \in [0, 1)$ satisfying $\tau < \tau'$, $(1 - \tau')\mathrm{a}$ distinguishes τ and τ' because $\tau(1 - \tau')\mathrm{a}$ leads to l_0 while $\tau(1 - \tau)\mathrm{a}$ leads to l_1. Therefore, such a congruence can make *infinitely* many classes.

Instead, we define a Nerode-style congruence between sets of timed words called *elementary languages* [29]. An elementary language is a timed language defined by a word with a conjunction of inequalities constraining the time difference between events. We also use an equality constraint, which we call, a *renaming equation* to define the congruence. Intuitively, a renaming equation bridges the time differences in an elementary language and the clock variables in a TA. We note that there can be multiple renaming equations showing the equivalence of two elementary languages.

Example 1. Let p_1 and p_2 be elementary languages $p_1 = \{\tau_0^1 \mathrm{a} \tau_1^1 \mid \tau_0^1 \in (0, 1), \tau_1^1 \in (0, 1), \tau_0^1 + \tau_1^1 \in (0, 1)\}$ and $p_2 = \{\tau_0^2 \mathrm{a} \tau_1^2 \mathrm{a} \tau_2^2 \mid \tau_0^2 \in (1, 2), \tau_1^2 \in (0, 1), \tau_2^2 \in (0, 1), \tau_1^2 + \tau_2^2 \in (0, 1)\}$. For the DTA in Fig. 1c, p_1 and p_2 are equivalent with the renaming equation $\tau_0^1 + \tau_1^1 = \tau_1^2 + \tau_2^2$ because for any $w_1 = \tau_0^1 \mathrm{a} \tau_1^1 \in p_1$ and $w_2 = \tau_0^2 \mathrm{a} \tau_1^2 \mathrm{a} \tau_2^2 \in p_2$: 1) we reach l_0 after reading either of w_1 and w_2 and 2) the values of c after reading w_1 and w_2 are $\tau_0^1 + \tau_1^1$ and $\tau_1^2 + \tau_2^2$, respectively.

We characterize recognizable timed languages by the finiteness of the equivalence classes defined by the above congruence. We also show that for any recognizable timed language, there is a finite set S of elementary languages such that the equivalence of any prefixes can be checked by the extensions S.

By using the above congruence, we extend the L* algorithm for DTAs. The high-level idea is the same as the original L* algorithm: 1) the learner makes membership queries to obtain enough information to construct a hypothesis DTA $\mathcal{A}_{\mathrm{hyp}}$ and 2) the learner makes an equivalence query to check if $\mathcal{A}_{\mathrm{hyp}}$ recognizes the target language. The largest difference is in the cells of an observation table. Since the concatenation $p \cdot s$ of an index pair $(p, s) \in P \times S$ is not a timed word but a set of timed words, its membership is not defined as a Boolean value. Instead, we introduce the notion of *symbolic* membership and use it as the value of each cell of the *timed* observation table. Intuitively, the symbolic membership is the constraint representing the subset of $p \cdot s$ included by $\mathcal{L}_{\mathrm{tgt}}$. Such a constraint can be constructed by finitely many (non-symbolic) membership queries.

Example 2. Fig. 1d illustrates a *timed* observation table. The equivalence between $p_1, p_2 \in P$ under S can be checked by comparing the cells in the rows indexed with p_1 and p_2 with renaming equations. For the cells in rows indexed by p_1 and p_2, their constraints are the same by replacing $\tau_0 + \tau_1$ with $\tau_1 + \tau_2$ and vice versa. Thus, p_1 and p_2 are equivalent with the current extensions S.

Once the learner obtains enough information, it constructs a DTA via the monoid-based representation of recognizable timed languages [29]. We show that for any recognizable timed language, our algorithm terminates and returns a DTA recognizing it. We also show that the number of the necessary queries is polynomial to the size of the equivalence class defined by the Nerode-style congruence if symbolic membership queries are allowed and, otherwise, exponential to it. Moreover, if symbolic membership queries are not allowed, the number of the necessary queries is at most doubly exponential to the number of the clock variable of a DTA recognizing the target language and singly exponential to the number of locations of a DTA recognizing the target language. This worst-case complexity is the same as the one-clock DTA learning algorithm in [39].

We implemented our DTA learning algorithm in a prototype library LearnTA. Our experiment results show that it is efficient enough for some benchmarks taken from practical applications, e. g., the FDDI protocol. This suggests the practical relevance of our algorithm.

The following summarizes our contribution.

- We characterize recognizable timed languages by a Nerode-style congruence.
- Using the above characterization, we give an active DTA learning algorithm.
- Our experiment results suggest its practical relevance.

Related work Among various characterization of timed languages [6,16,12,29,14,13], the characterization by *recognizability* [29] is closest to our Myhill-Nerode-style characterization. Both of them use finite sets of prefix elementary languages for characterization. Their main difference is that [29] proposes a formalism to define a timed language by relating prefixes by a morphism, whereas we propose a technical gadget to define an equivalence relation over timed words with respect to suffixes using symbolic membership. This difference makes our definition suitable for an L*-style algorithm, where the original L* algorithm is based on Nerode's congruence, which defines an equivalence relation over words with respect to suffixes using conventional membership.

As we have discussed so far, active TA learning [21,22,7,39,24] has been studied mostly for limited subclasses of TAs, where the number of the clock variables or the clock variables reset at each edge is fixed. In contrast, our algorithm infers both of the above information. Another difference is in the technical strategy. Most of the existing algorithms are related to the active learning of *symbolic automata* [20,11], enhancing the languages with clock valuations. In contrast, we take a more semantic approach via the Nerode-style congruence.

Another recent direction is to use a *genetic algorithm* to infer TAs in passive [36] or active [5] learning. This differs from our learning algorithm based on a formal characterization of timed languages. Moreover, these algorithms may not converge to the correct automaton due to a genetic algorithm.

2 Preliminaries

For a set X, its powerset is denoted by $\mathcal{P}(X)$. We denote the empty sequence by ε. For sets X, Y, we denote their symmetric difference by $X \triangle Y = \{x \mid x \in X \land x \notin Y\} \cup \{y \mid y \in Y \land y \notin X\}$.

2.1 Timed words and timed automata

Definition 3 (timed word). *For a finite alphabet Σ, a* timed word *w is an alternating sequence $\tau_0 a_1 \tau_1 a_2 \dots a_n \tau_n$ of Σ and $\mathbb{R}_{\geq 0}$. The set of timed words over Σ is denoted by $\mathcal{T}(\Sigma)$. A* timed language *$\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ is a set of timed words.*

For timed words $w = \tau_0 a_1 \tau_1 a_2 \dots a_n \tau_n$ and $w' = \tau'_0 a'_1 \tau'_1 a'_2 \dots a'_{n'} \tau'_{n'}$, their concatenation $w \cdot w'$ is $w \cdot w' = \tau_0 a_1 \tau_1 a_2 \dots a_n (\tau_n + \tau'_0) a'_1 \tau'_1 a'_2 \dots a'_{n'} \tau'_{n'}$. The concatenation is naturally extended to timed languages: for a timed word w and timed languages $\mathcal{L}, \mathcal{L}'$, we let $w \cdot \mathcal{L} = \{w \cdot w_{\mathcal{L}} \mid w_{\mathcal{L}} \in \mathcal{L}\}$, $\mathcal{L} \cdot w = \{w_{\mathcal{L}} \cdot w \mid w_{\mathcal{L}} \in \mathcal{L}\}$, and $\mathcal{L} \cdot \mathcal{L}' = \{w_{\mathcal{L}} \cdot w_{\mathcal{L}'} \mid w_{\mathcal{L}} \in \mathcal{L}, w_{\mathcal{L}'} \in \mathcal{L}'\}$. For timed words w and w', w is a *prefix* of w' if there is a timed word w'' satisfying $w \cdot w'' = w'$. A timed language $\mathcal{L}$ is *prefix-closed* if for any $w \in \mathcal{L}$, $\mathcal{L}$ contains all the prefixes of w.

For a finite set C of clock variables, a *clock valuation* is a function $\nu \in (\mathbb{R}_{\geq 0})^C$. We let $\mathbf{0}_C$ be the clock valuation $\mathbf{0}_C \in (\mathbb{R}_{\geq 0})^C$ satisfying $\mathbf{0}_C(c) = 0$ for any $c \in C$. For a clock valuation $\nu \in (\mathbb{R}_{\geq 0})^C$ over C and $\tau \in \mathbb{R}_{\geq 0}$, we let $\nu + \tau$ be the clock valuation satisfying $(\nu + \tau)(c) = \nu(c) + \tau$ for any $c \in C$. For a clock valuation $\nu \in (\mathbb{R}_{\geq 0})^C$ and $\rho \subseteq C$, we let $\nu[\rho := 0]$ be the clock valuation satisfying $(\nu[\rho := 0])(x) = 0$ for $c \in \rho$ and $(\nu[\rho := 0])(c) = \nu(c)$ for $c \notin \rho$. We let $\mathcal{G}_C$ be the set of constraints defined by a finite conjunction of inequalities $c \bowtie d$, where $c \in C$, $d \in \mathbb{N}$, and $\bowtie \in \{>, \geq, \leq, <\}$. We let $\mathcal{C}_C$ be the set of constraints defined by a finite conjunction of inequalities $c \bowtie d$ or $c - c' \bowtie d$, where $c, c' \in C$, $d \in \mathbb{N}$, and $\bowtie \in \{>, \geq, \leq, <\}$. We denote $\bigwedge \emptyset$ by $\top$. For a clock valuation $\nu \in (\mathbb{R}_{\geq 0})^C$ and $\varphi \in \mathcal{C}_C \cup \mathcal{G}_C$, we denote $\nu \models \varphi$ if ν satisfies φ.

Definition 4 (timed automaton). *A* timed automaton *(TA)*[1] *is a 7-tuple* $(\Sigma, L, l_0, C, I, \Delta, F)$*, where:* Σ *is the finite alphabet,* L *is the finite set of locations,* $l_0 \in L$ *is the initial location,* C *is the finite set of clock variables,* $I\colon L \to \mathcal{C}_C$ *is the invariant of each location,* $\Delta \subseteq L \times \mathcal{G}_C \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{P}(C) \times L$ *is the set of edges, and* $F \subseteq L$ *is the accepting locations.*

An edge is *unobservable* if it is labeled with ε, and otherwise, it is *observable*.

Definition 5 (DTA). *A TA* $\mathcal{A} = (\Sigma, L, l_0, C, I, \Delta, F)$ *is* deterministic *if we have the following.*

- *For any* $a \in \Sigma$ *and* $(l, g, a, \rho, l'), (l, g', a, \rho', l'') \in \Delta$*,* $g \wedge g'$ *is unsatisfiable.*
- *For any* $(l, g, \varepsilon, \rho, l') \in \Delta$*,* $g \wedge I(l)$ *is at most a singleton.*

Fig. 1c shows a deterministic TA (DTA).

The semantics of a TA is defined by a *timed transition system (TTS).*

Definition 6 (semantics of TAs). *For a TA* $\mathcal{A} = (\Sigma, L, l_0, C, I, \Delta, F)$*, the* timed transition system (TTS) *is a 4-tuple* $\mathcal{S} = (Q, q_0, Q_F, \to)$*, where:*

- $Q = L \times (\mathbb{R}_{\geq 0})^C$ *is the set of* (concrete) states*,*
- $q_0 = (l_0, \mathbf{0}_C)$ *is the* initial state*,*
- $Q_F = \{(l, \nu) \in Q \mid l \in F\}$ *is the set of* accepting states*, and*
- $\to \subseteq Q \times Q$ *is the* transition relation *consisting of the following*[2]*.*
  - *For each* $(l, \nu) \in Q$ *and* $\tau \in \mathbb{R}_{>0}$*, we have* $(l, \nu) \xrightarrow{\tau} (l, \nu + \tau)$ *if* $\nu + \tau' \models I(l)$ *holds for each* $\tau' \in [0, \tau)$*.*
  - *For each* $(l, \nu), (l', \nu') \in Q$*,* $a \in \Sigma$*, and* $(l, g, a, \rho, l') \in \Delta$*, we have* $(l, \nu) \xrightarrow{a} (l', \nu')$ *if we have* $\nu \models g$ *and* $\nu' = \nu[\rho := 0]$*.*
  - *For each* $(l, \nu), (l', \nu') \in Q$*,* $\tau \in \mathbb{R}_{>0}$*, and* $(l, g, \varepsilon, \rho, l') \in \Delta$*, we have* $(l, \nu) \xrightarrow{\varepsilon, \tau} (l', \nu' + \tau)$ *if we have* $\nu \models g$*,* $\nu' = \nu[\rho := 0]$*, and* $\forall \tau' \in [0, \tau).\, \nu' + \tau' \models I(l')$*.*

A *run* of a TA $\mathcal{A}$ is an alternating sequence $q_0, \to_1, q_1, \dots, \to_n, q_n$ of states $q_i \in Q$ and transitions $\to_i \in \to$ satisfying $q_{i-1} \to_i q_i$ for any $i \in \{1, 2, \dots, n\}$. A run $q_0, \to_1, q_1, \dots, \to_n, q_n$ is accepting if $q_n \in Q_F$. Given such a run, the associated timed word is the concatenation of the labels of the transitions. The timed *language* $\mathcal{L}(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of timed words associated with some accepting run of $\mathcal{A}$.

[1] Unlike [29], we define the acceptance by the accepting *locations* rather than accepting *conditions*. This choice does not affect the expressive power because accepting conditions can be encoded with unobservable edges and invariants.

[2] We use $\xrightarrow{\varepsilon, \tau}$ to avoid the discussion with an arbitrary small dwell time in [29].

2.2 Recognizable timed languages

Here, we review the *recognizability* [29] of timed languages.

Definition 7 (timed condition). *For a set $\mathbb{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$ of ordered variables, a* timed condition *Λ is a finite conjunction of inequalities $\mathbb{T}_{i,j} \bowtie d$, where $\mathbb{T}_{i,j} = \sum_{k=i}^{j} \tau_k$, $\bowtie \in \{>, \geq, \leq, <\}$, and $d \in \mathbb{N}$.*

A timed condition Λ is *simple*[3] if for each $\mathbb{T}_{i,j}$, Λ contains $d < \mathbb{T}_{i,j} < d+1$ or $d \leq \mathbb{T}_{i,j} \wedge \mathbb{T}_{i,j} \leq d$ for some $d \in \mathbb{N}$. A timed condition Λ is *canonical* if we cannot strengthen or add any inequality $\mathbb{T}_{i,j} \bowtie d$ to Λ without changing its semantics.

Definition 8 (elementary language). *A timed language $\mathcal{L}$ is* elementary *if there are $u = a_1, a_2, \dots, a_n \in \Sigma^*$ and a timed condition Λ over $\{\tau_0, \tau_1, \dots, \tau_n\}$ satisfying $\mathcal{L} = \{\tau_0 a_1 \tau_1 a_2 \dots a_n \tau_n \mid \tau_0, \tau_1, \dots, \tau_n \models \Lambda\}$, and the set of valuations of $\{\tau_0, \tau_1, \dots, \tau_n\}$ defined by Λ is bounded. We denote such an elementary language $\mathcal{L}$ by (u, Λ). We let $\mathcal{E}(\Sigma)$ be the set of elementary languages over Σ.*

For elementary languages $p, p' \in \mathcal{E}(\Sigma)$, p is a *prefix* of p' if for any $w' \in p'$, there is a prefix $w \in p$ of w', and for any $w \in p$, there is $w' \in p'$ such that w is a prefix of w'. For any elementary language, the number of its prefixes is finite. For a set of elementary languages, the notions of *prefix-closedness* is defined based on the above definition of prefixes.

An elementary language (u, Λ) is *simple* if there is a simple and canonical timed condition Λ' satisfying $(u, \Lambda) = (u, \Lambda')$. We let $\mathcal{SE}(\Sigma)$ be the set of simple elementary languages over Σ. Without loss of generality, we assume that for any $(u, \Lambda) \in \mathcal{SE}(\Sigma)$, Λ is simple and canonical. We remark that any DTA cannot distinguish timed words in a simple elementary language, i. e., for any simple elementary language $p \in \mathcal{SE}(\Sigma)$ and a DTA $\mathcal{A}$, we have either $p \subseteq \mathcal{L}(\mathcal{A})$ or $p \cap \mathcal{L}(\mathcal{A}) = \emptyset$. We can decide if $p \subseteq \mathcal{L}(\mathcal{A})$ or $p \cap \mathcal{L}(\mathcal{A}) = \emptyset$ by taking some $w \in p$ and checking if $w \in \mathcal{L}(\mathcal{A})$.

Definition 9 (immediate exterior). *Let $\mathcal{L} = (u, \Lambda)$ be an elementary language. For $a \in \Sigma$, the* discrete immediate exterior $\mathrm{ext}^a(\mathcal{L})$ *of $\mathcal{L}$ is $\mathrm{ext}^a(\mathcal{L}) = (u \cdot a, \Lambda \cup \{\tau_{|u|+1} = 0\})$. The* continuous immediate exterior $\mathrm{ext}^t(\mathcal{L})$ *of $\mathcal{L}$ is $\mathrm{ext}^t(\mathcal{L}) = (u, \Lambda^t)$, where Λ^t is the timed condition such that each inequality $\mathbb{T}_{i,|u|} = d$ in Λ is replaced with $\mathbb{T}_{i,|u|} > d$ if such an inequality exists, and otherwise, the inequality $\mathbb{T}_{i,|u|} < d$ in Λ with the smallest index i is replaced with $\mathbb{T}_{i,|u|} = d$. The* immediate exterior $\mathrm{ext}(\mathcal{L})$ *of $\mathcal{L}$ is $\mathrm{ext}(\mathcal{L}) = \bigcup_{a \in \Sigma} \mathrm{ext}^a(\mathcal{L}) \cup \mathrm{ext}^t(\mathcal{L})$.*

Example 10. For a word $u = \mathrm{a} \cdot \mathrm{a}$ and a timed condition $\Lambda = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} \in (1,2) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} = 0\}$, we have $1.3 \cdot \mathrm{a} \cdot 0.5 \cdot \mathrm{a} \cdot 0 \in (u, \Lambda)$ and $1.7 \cdot \mathrm{a} \cdot 0.5 \cdot \mathrm{a} \cdot 0 \notin (u, \Lambda)$. The discrete and continuous immediate exteriors of (u, Λ) are $\mathrm{ext}^{\mathrm{a}}((u, \Lambda)) = (u \cdot \mathrm{a}, \Lambda^{\mathrm{a}})$ and $\mathrm{ext}^t((u, \Lambda)) = (u, \Lambda^t)$, where $\Lambda^{\mathrm{a}} = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} \in (1,2) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} = \mathbb{T}_{3,3} = 0\}$ and $\Lambda^t = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} \in (1,2) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} > 0\}$.

[3] The notion of simplicity is taken from [21]

Definition 11 (chronometric timed language). *A timed language $\mathcal{L}$ is* chronometric *if there is a finite set $\{(u_1, \Lambda_1), (u_2, \Lambda_2), \dots, (u_m, \Lambda_m)\}$ of disjoint elementary languages satisfying $\mathcal{L} = \bigcup_{i \in \{1,2,\dots,m\}} (u_i, \Lambda_i)$.*

For any elementary language $\mathcal{L}$, its immediate exterior $\mathrm{ext}(\mathcal{L})$ is chronometric. We naturally extend the notion of exterior to chronometric timed languages, i. e., for a chronometric timed language $\mathcal{L} = \bigcup_{i \in \{1,2,\dots,m\}} (u_i, \Lambda_i)$, we let $\mathrm{ext}(\mathcal{L}) = \bigcup_{i \in \{1,2,\dots,m\}} \mathrm{ext}((u_i, \Lambda_i))$, which is also chronometric. For a timed word $w = \tau_0 a_1 \tau_1 a_2 \dots a_n \tau_n$, we denote the valuation of $\tau_0, \tau_1, \dots, \tau_n$ by $\kappa(w) \in (\mathbb{R}_{\geq 0})^{\mathbb{T}}$.

Chronometric relational morphism [29] relates any timed word to a timed word in a certain set P, which is later used to define a timed language. Intuitively, the tuples in Φ specify a mapping from timed words immediately out of P to timed words in P. By inductively applying it, any timed word is mapped to P.

Definition 12 (chronometric relational morphism). *Let P be a chronometric and prefix-closed timed language. Let $(u, \Lambda, u', \Lambda', R)$ be a 5-tuple such that $(u, \Lambda) \subseteq \mathrm{ext}(P)$, $(u', \Lambda') \subseteq P$, and R is a finite conjunction of equations of the form $\mathbb{T}_{i,|u|} = \mathbb{T}'_{j,|u'|}$, where $i \leq |u|$ and $j \leq |u'|$. For such a tuple, we let $[\![(u, \Lambda, u', \Lambda', R)]\!] \subseteq (u, \Lambda) \times (u', \Lambda')$ be the relation such that $(w, w') \in [\![(u, \Lambda, u', \Lambda', R)]\!]$ if and only if $\kappa(w), \kappa(w') \models R$. For a finite set Φ of such tuples $(u, \Lambda, u', \Lambda', R)$, the* chronometric relational morphism *$[\![\Phi]\!] \subseteq \mathcal{T}(\Sigma) \times P$ is the relation inductively defined as follows: 1) for $w \in P$, we have $(w, w) \in [\![\Phi]\!]$; 2) for $w \in \mathrm{ext}(P)$ and $w' \in P$, we have $(w, w') \in [\![\Phi]\!]$ if we have $(w, w') \in [\![(u, \Lambda, u', \Lambda', R)]\!]$ for one of the tuples $(u, \Lambda, u', \Lambda', R) \in \Phi$; 3) for $w \in \mathrm{ext}(P)$, $w' \in \mathcal{T}(\Sigma)$, and $w'' \in P$, we have $(w \cdot w', w'') \in [\![\Phi]\!]$ if there is $w''' \in \mathcal{T}(\Sigma)$ satisfying $(w, w''') \in [\![\Phi]\!]$ and $(w''' \cdot w', w'') \in [\![\Phi]\!]$. We also require that all (u, Λ) in the tuples in Φ must be disjoint and the union of each such (u, Λ) is $\mathrm{ext}(P) \setminus P$.*

A chronometric relational morphism $[\![\Phi]\!]$ is *compatible* with $F \subseteq P$ if for each tuple $(u, \Lambda, u', \Lambda', R)$ defining $[\![\Phi]\!]$, we have either $(u', \Lambda') \subseteq F$ or $(u', \Lambda') \cap F = \emptyset$.

Definition 13 (recognizable timed language). *A timed language $\mathcal{L}$ is* recognizable *if there is a chronometric prefix-closed set P, a chronometric subset F of P, and a chronometric relational morphism $[\![\Phi]\!] \subseteq \mathcal{T}(\Sigma) \times P$ compatible with F satisfying $\mathcal{L} = \{w \mid \exists w' \in F, (w, w') \in [\![\Phi]\!]\}$.*

It is known that for any recognizable timed language $\mathcal{L}$, we can construct a DTA $\mathcal{A}$ recognizing $\mathcal{L}$, and vice versa [29].

2.3 Distinguishing extensions and active DFA learning

Among the various characterization of regular languages, most DFA learning algorithms are based on the characterization with *Nerode's congruence* [26]. For a (not necessarily regular) language $\mathcal{L} \subseteq \Sigma^*$, Nerode's congruence $\equiv_{\mathcal{L}} \subseteq \Sigma^* \times \Sigma^*$ is the equivalence relation satisfying $w \equiv_{\mathcal{L}} w'$ if and only if for any $w'' \in \Sigma^*$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$.

Algorithm 1: Outline of an L*-style active DFA learning algorithm

```
1  $P \leftarrow \{\varepsilon\}$; $S \leftarrow \{\varepsilon\}$
2  while $\top$ do
3      while the observation table is not closed or consistent do
4          update $P$ and $S$ so that the observation table is closed and consistent
5      $\mathcal{A}_{\mathrm{hyp}} \leftarrow$ ConstructDFA($P$, $S$, $T$)
6      switch $\mathtt{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}})$ do
7          case $\top$ do
8              return $\mathcal{A}_{\mathrm{hyp}}$
9          case cex do
10             Update $P$ and/or $S$ using cex
```

Generally, we cannot decide if $w \equiv_{\mathcal{L}} w'$ by testing because it requires infinitely many membership checking. However, if $\mathcal{L}$ is regular, thanks to the Myhill-Nerode theorem, we can decide whether $w \equiv_{\mathcal{L}} w'$ only by finitely many membership checking. More precisely, for any regular language $\mathcal{L}$, there is a finite set of suffixes $S \subseteq \Sigma^*$ called *distinguishing extensions* satisfying $\equiv_{\mathcal{L}} = \sim^S_{\mathcal{L}}$, where $\sim^S_{\mathcal{L}}$ is the equivalence relation satisfying $w \sim^S_{\mathcal{L}} w'$ if and only if for any $w'' \in S$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$. Thus, the minimum DFA recognizing $\mathcal{L}_{\mathrm{tgt}}$ can be learned by[4]: i) identifying distinguishing extensions S satisfying $\equiv_{\mathcal{L}_{\mathrm{tgt}}} = \sim^S_{\mathcal{L}_{\mathrm{tgt}}}$ and ii) constructing the minimum DFA $\mathcal{A}$ corresponding to $\sim^S_{\mathcal{L}_{\mathrm{tgt}}}$.

The *L^* algorithm* [10] is an algorithm to learn the minimum DFA $\mathcal{A}_{\mathrm{hyp}}$ recognizing the target regular language $\mathcal{L}_{\mathrm{tgt}}$ with finitely many *membership* and *equivalence* queries to the teacher. In a membership query, the learner asks if $w \in \Sigma^*$ belongs to the target language $\mathcal{L}_{\mathrm{tgt}}$ i.e., $w \in \mathcal{L}_{\mathrm{tgt}}$. In an equivalence query, the learner asks if the hypothesis DFA $\mathcal{A}_{\mathrm{hyp}}$ recognizes the target language $\mathcal{L}_{\mathrm{tgt}}$ i.e., $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$, where $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ is the language of the hypothesis DFA $\mathcal{A}_{\mathrm{hyp}}$. When we have $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \neq \mathcal{L}_{\mathrm{tgt}}$, the teacher returns a counterexample $cex \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \triangle \mathcal{L}_{\mathrm{tgt}}$. The information obtained via queries is stored in a 2-dimensional array called an *observation table*. See Fig. 1b for an illustration. For finite index sets $P, S \subseteq \Sigma^*$, for each pair $(p, s) \in (P \cup P \cdot \Sigma) \times S$, the observation table stores whether the concatenation $p \cdot s$ is a member of the target language $\mathcal{L}_{\mathrm{tgt}}$. Formally, an observation table is a 3-tuple (P, S, T) of a prefix-closed finite set P of words, a suffix-closed finite set S of words, and a function $T \colon (P \cup P \cdot \Sigma) \times S \to \{\top, \bot\}$ assigning a pair (p, s) of words to the membership $p \cdot s \in \mathcal{L}_{\mathrm{tgt}}$ of their concatenation to the target language. S is the current candidate of the distinguishing extensions, and P represents $\Sigma^* / \sim^S_{\mathcal{L}_{\mathrm{tgt}}}$. Since P and S are finite, one can fill the observation table using finite membership queries.

Algorithm 1 outlines an L*-style algorithm. We start from $P = S = \{\varepsilon\}$ and incrementally increase them. To construct a hypothesis DFA $\mathcal{A}_{\mathrm{hyp}}$, the observation table must be *closed* and *consistent*. Intuitively, the closedness assures that

[4] The distinguishing extensions S can be defined locally. For example, the TTT algorithm [27] is optimized with *local* distinguishing extensions for some prefixes $w \in \Sigma^*$. Nevertheless, we use the global distinguishing extensions for simplicity.

the target state of each transition is in P, and the consistency assures that all the states with the same row behave in the same way. Formally, an observation table is *closed* if, for each $p \in P \cdot \Sigma$, there is $p' \in P$ satisfying $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$. An observation table is *consistent* if, for any $p, p' \in P \cup P \cdot \Sigma$ and $a \in \Sigma$, $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$ implies $p \cdot a \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p' \cdot a$.

Once the observation table becomes closed and consistent, the learner constructs a hypothesis DFA $\mathcal{A}_{\mathrm{hyp}}$ and checks if $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$ by an equivalence query. If $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$ holds, $\mathcal{A}_{\mathrm{hyp}}$ is the resulting DFA. Otherwise, the teacher returns a counterexample $cex \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \triangle \mathcal{L}_{\mathrm{tgt}}$, which is used to refine the observation table. There are several variants of the refinement. In the L* algorithm, all the prefixes of cex are added to P. In the Rivest-Schapire algorithm [33,28], an extension s strictly refining S is obtained by an analysis of cex, and such s is added to S.

3 A Myhill-Nerode style characterization of recognizable timed languages with elementary languages

Unlike the case of regular languages, any finite set of timed words cannot correctly distinguish recognizable timed languages due to the infiniteness of dwell time in timed words. Instead, we use a finite set of *elementary languages* to define a Nerode-style congruence. To define the Nerode-style congruence, we extend the notion of membership to elementary languages.

Definition 14 (symbolic membership). *For a timed language $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ and an elementary language $(u, \Lambda) \in \mathcal{E}(\Sigma)$, the* symbolic membership $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda))$ *of (u, Λ) to $\mathcal{L}$ is the strongest constraint such that for any $w \in (u, \Lambda)$, we have $w \in \mathcal{L}$ if and only if $\kappa(w) \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}(\mathcal{L})$.*

We discuss how to obtain symbolic membership in Section 4.5. We define a Nerode-style congruence using symbolic membership. A naive idea is to distinguish two elementary languages (u, Λ) and (u', Λ') by the equivalence of their symbolic membership $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda))$ and $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda'))$. However, this does not capture the semantics of TAs. For example, for the DTA $\mathcal{A}$ in Fig. 1c, for any timed word w, we have $1.3 \cdot \mathrm{a} \cdot 0.4 \cdot w \in \mathcal{L}(\mathcal{A}) \iff 0.3 \cdot \mathrm{a} \cdot 1.0 \cdot \mathrm{a} \cdot 0.4 \cdot w \in \mathcal{L}(\mathcal{A})$, while they have different symbolic membership. This is because symbolic membership distinguishes the *position* in timed words where each clock variable is reset, which must be ignored. We use *renaming equations* to abstract such positional information of the clock reset in symbolic membership. Note that $\mathbb{T}_{i,n} = \sum_{k=i}^{n} \tau_k$ corresponds to the value of the clock variable reset at τ_i.

Definition 15 (renaming equation). *Let $\mathbb{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$ and $\mathbb{T}' = \{\tau'_0, \tau'_1, \dots, \tau'_{n'}\}$ be sets of ordered variables. A* renaming equation *R over $\mathbb{T}$ and $\mathbb{T}'$ is a finite conjunction of equations of the form $\mathbb{T}_{i,n} = \mathbb{T}'_{i',n'}$, where $i \in \{0, 1, \dots, n\}$, $i' \in \{0, 1, \dots, n'\}$, $\mathbb{T}_{i,n} = \sum_{k=i}^{n} \tau_k$ and $\mathbb{T}'_{i',n'} = \sum_{k=i'}^{n'} \tau'_k$.*

Definition 16 ($\sim_{\mathcal{L}}^{S}$)**.** *Let $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ be a timed language, let $(u, \Lambda), (u', \Lambda'), (u'', \Lambda'') \in \mathcal{E}(\Sigma)$ be elementary languages, and let R be a renaming equation over $\mathbb{T}$ and $\mathbb{T}'$, where $\mathbb{T}$ and $\mathbb{T}'$ are the variables of Λ and Λ', respectively. We let $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{(u'',\Lambda''),R} (u', \Lambda')$ if we have the following: for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ satisfying $\kappa(w), \kappa(w') \models R$; $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda'$ is equivalent to $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u', \Lambda') \cdot (u'', \Lambda'')) \land R \land \Lambda$. We let $(u, \Lambda) \sim_{\mathcal{L}}^{(u'',\Lambda''),R} (u', \Lambda')$ if we have $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{(u'',\Lambda''),R} (u', \Lambda')$ and $(u', \Lambda') \sqsubseteq_{\mathcal{L}}^{(u'',\Lambda''),R} (u, \Lambda)$. Let $S \subseteq \mathcal{E}(\Sigma)$. We let $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$ if for any $(u'', \Lambda'') \in S$, we have $(u, \Lambda) \sim_{\mathcal{L}}^{(u'',\Lambda''),R} (u', \Lambda')$. We let $(u, \Lambda) \sim_{\mathcal{L}}^{S} (u', \Lambda')$ if $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$ for some renaming equation R.*

Example 17. Let $\mathcal{A}$ be the DTA in Fig. 1c and let (u, Λ), (u', Λ'), and (u'', Λ'') be elementary languages, where $u = \mathrm{a}$, $\Lambda = \{\tau_0 \in (1, 2) \land \tau_0 + \tau_1 \in (1, 2) \land \tau_1 \in (0, 1)\}$, $u' = \mathrm{a} \cdot \mathrm{a}$, $\Lambda' = \{\tau'_0 \in (0, 1) \land \tau'_0 + \tau'_1 \in (1, 2) \land \tau'_1 + \tau'_2 \in (1, 2) \land \tau'_2 \in (0, 1)\}$, $u'' = \mathrm{a}$, and $\Lambda'' = \{\tau_0 \in (0, 1) \land \tau_1 = 0\}$. We have $\mathtt{mem}_{\mathcal{L}(\mathcal{A})}^{\mathtt{sym}}((u, \Lambda) \cdot (u'', \Lambda'')) = \Lambda \land \Lambda'' \land \tau_1 + \tau''_0 \leq 1$ and $\mathtt{mem}_{\mathcal{L}(\mathcal{A})}^{\mathtt{sym}}((u', \Lambda') \cdot (u'', \Lambda'')) = \Lambda' \land \Lambda'' \land \tau'_2 + \tau''_0 \leq 1$. Therefore, for the renaming equation $\mathbb{T}_{1,1} = \mathbb{T}'_{2,2}$, we have $(u, \Lambda) \sim_{\mathcal{L}}^{(u'',\Lambda''),\mathbb{T}_{1,1}=\mathbb{T}'_{2,2}} (u', \Lambda')$.

An algorithm to check if $(u, \Lambda) \sim_{\mathcal{L}}^{S} (u', \Lambda')$ is shown in Appendix B.2.

Intuitively, $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{s,R} (u', \Lambda')$ shows that any $w \in (u, \Lambda)$ can be "simulated" by some $w' \in (u', \Lambda')$ with respect to s and R. Such intuition is formalized as follows.

Theorem 18. *For any $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ and $(u, \Lambda), (u', \Lambda'), (u'', \Lambda'') \in \mathcal{E}(\Sigma)$ satisfying $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{(u'',\Lambda'')} (u', \Lambda')$, for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $w'' \in (u'', \Lambda'')$, $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$ holds.* □

Proof (sketch). Let R be a renaming equation satisfying $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{S,R} (u', \Lambda')$. By the definition of $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{S,R} (u', \Lambda')$, for any $(u'', \Lambda'') \in S$, $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u', \Lambda') \cdot (u'', \Lambda'')) \land R \land \Lambda$. Therefore, any suffix $(u'', \Lambda'') \in S$ cannot distinguish $w \in (u, \Lambda)$ and $w' \in (u', \Lambda')$ if we have $\kappa(w), \kappa(w') \models R$. Since $(u, \Lambda) \sqsubseteq_{\mathcal{L}}^{S,R} (u', \Lambda')$, For any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ satisfying $\kappa(w), \kappa(w') \models R$. Overall, for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $(u'', \Lambda'') \in S$ and for any $w'' \in (u'', \Lambda'')$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$. □

By $\bigcup_{(u,\Lambda) \in \mathcal{E}(\Sigma)} (u, \Lambda) = \mathcal{T}(\Sigma)$, we have the following as a corollary.

Corollary 19. *For any timed language $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ and for any elementary languages $(u, \Lambda), (u', \Lambda') \in \mathcal{E}(\Sigma)$, $(u, \Lambda) \sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} (u', \Lambda')$ implies the following.*

- *For any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $w'' \in \mathcal{T}(\Sigma)$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$.*
- *For any $w' \in (u', \Lambda')$, there is $w \in (u, \Lambda)$ such that for any $w'' \in \mathcal{T}(\Sigma)$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$.* □

The following theorem characterizes recognizable timed languages with $\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}$.

Theorem 20 (Myhill-Nerode style characterization). *A timed language $\mathcal{L}$ is recognizable if and only if the quotient set $\mathcal{SE}(\Sigma)/\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}$ is finite.* □

By Theorem 20, we always have a finite set S of distinguishing extensions.

Theorem 21. *For any recognizable timed language $\mathcal{L}$, there is a finite set S of elementary languages satisfying $\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} = \sim_{\mathcal{L}}^{S}$.* □

4 Active learning of deterministic timed automata

We show our L*-style active learning algorithm for DTAs with the Nerode-style congruence in Section 3. We let $\mathcal{L}_{\mathrm{tgt}} \subseteq \mathcal{T}(\Sigma)$ be the target timed language in learning.

For simplicity, we first present our learning algorithm with a smart teacher answering the following three kinds of queries: membership query $\mathtt{mem}_{\mathcal{L}_{\mathrm{tgt}}}(w)$ asking whether $w \in \mathcal{L}_{\mathrm{tgt}}$, symbolic membership query asking $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u, \Lambda))$, and equivalence query $\mathtt{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}})$ asking whether $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$. If $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$, $\mathtt{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}}) = \top$, and otherwise, $\mathtt{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}})$ is a timed word $\mathit{cex} \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \triangle \mathcal{L}_{\mathrm{tgt}}$. Later in Section 4.5, we show how to answer a symbolic membership query with finitely many membership queries. Our task is to construct a DTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}_{\mathrm{tgt}}$ with finitely many queries.

4.1 Successors of simple elementary languages

Similarly to the L* algorithm in Section 2.3, we learn a DTA with an observation table. Reflecting the extension of the underlying congruence, we use sets of simple elementary languages for the indices. To define the extensions, $P \cup (P \cdot \Sigma)$ in the L* algorithm, we introduce *continuous* and *discrete successors* for simple elementary languages, which are inspired by *regions* [6]. We note that immediate exteriors do not work for this purpose.

Example 22. Let $(u, \Lambda) = (\varepsilon, 0 < \tau_0 \land \tau_0 < 1)$. Since we have $\mathrm{ext}^t((u, \Lambda)) = (\varepsilon, 0 < \tau_0 \land \tau_0 = 1)$, $\mathrm{ext}^t((u, \Lambda))$ is empty. Let $(u', \Lambda') = (a, \mathbb{T}'_{0,1} < 2 \land \mathbb{T}'_{1,1} < 1)$ and $w = 0.7 \cdot a \cdot 0.9$. We have $w \in (u', \Lambda')$. Since we have $\mathrm{ext}^t((u', \Lambda')) = (a, \mathbb{T}'_{0,1} = 2 \land \mathbb{T}'_{1,1} < 1)$, there is no $t > 0$ satisfying $w \cdot t \in \mathrm{ext}^t((u', \Lambda'))$.

For any simple elementary language $(u, \Lambda) \in \mathcal{SE}(\Sigma)$, we let $\Theta_{(u,\Lambda)}$ be the total order over 0 and the fractional parts $\mathrm{frac}(\mathbb{T}_{0,n}), \mathrm{frac}(\mathbb{T}_{1,n}), \dots, \mathrm{frac}(\mathbb{T}_{n,n})$ of $\mathbb{T}_{0,n}, \mathbb{T}_{1,n}, \dots, \mathbb{T}_{n,n}$. Such an order is uniquely defined because Λ is simple and canonical (Proposition 36).

Definition 23 (successor). *Let $p = (u, \Lambda) \in \mathcal{SE}(\Sigma)$ be a simple elementary language. The* discrete successor $\mathrm{succ}^a(p)$ *of p is* $\mathrm{succ}^a(p) = (u \cdot a, \Lambda \land \tau_{n+1} =$

0). *The* continuous successor $\mathrm{succ}^t(p)$ *of* p *is* $\mathrm{succ}^t(p) = (u, \Lambda^t)$*, where* Λ^t *is defined as follows: if there is an equation* $\mathbb{T}_{i,n} = d$ *in* Λ*, all such equations are replaced with* $\mathbb{T}_{i,n} \in (d, d+1)$*; otherwise, for each greatest* $\mathbb{T}_{i,n}$ *in terms of* $\Theta_{(u,\Lambda)}$*, we replace* $\mathbb{T}_{i,n} \in (d, d+1)$ *with* $\mathbb{T}_{i,n} = d+1$*. We let* $\mathrm{succ}(p) = \bigcup_{a \in \Sigma} \mathrm{succ}^a(p) \cup \mathrm{succ}^t(p)$*. For a set* $P \subseteq \mathcal{SE}(\Sigma)$ *of simple elementary languages, we let* $\mathrm{succ}(P) = \bigcup_{p \in P} \mathrm{succ}(p)$*.*

One can easily check that both discrete and continuous successors of a simple elementary language are also simple.

Example 24. Let $u = aa$, $\Lambda = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} \in (1,2) \wedge \mathbb{T}_{1,1} \in (0,1) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} = 0\}$. The order $\Theta_{(u,\Lambda)}$ is such that $0 = \mathrm{frac}(\mathbb{T}_{2,2}) < \mathrm{frac}(\mathbb{T}_{1,2}) < \mathrm{frac}(\mathbb{T}_{0,2})$. The continuous successor of (u, Λ) is $\mathrm{succ}^t((u,\Lambda)) = (u, \Lambda^t)$, where $\Lambda^t = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} \in (1,2) \wedge \mathbb{T}_{1,1} \in (0,1) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} \in (0,1)\}$. The continuous successor of (u, Λ^t) is $\mathrm{succ}^t((u,\Lambda^t)) = (u, \Lambda^{tt})$, where $\Lambda^{tt} = \{\mathbb{T}_{0,0} \in (1,2) \wedge \mathbb{T}_{0,1} \in (1,2) \wedge \mathbb{T}_{0,2} = 2 \wedge \mathbb{T}_{1,1} \in (0,1) \wedge \mathbb{T}_{1,2} \in (0,1) \wedge \mathbb{T}_{2,2} \in (0,1)\}$.

4.2 Timed observation table for active DTA learning

We extend the observation table with (simple) elementary languages and symbolic membership to learn a *recognizable timed language* rather than a *regular language.*

Definition 25 (timed observation table). *A* timed observation table *for DTA learning is a 3-tuple* (P, S, T) *such that:* P *is a prefix-closed finite set of simple elementary languages,* S *is a finite set of elementary languages, and* T *is a function mapping* $(p, s) \in (P \cup \mathrm{succ}(P)) \times S$ *to the symbolic membership* $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}(p \cdot s)$*.*

Fig. 2 illustrates timed observation tables: each cell indexed by (p, s) show the symbolic membership $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}(p \cdot s)$. In the original L* algorithm in Section 2.3, closedness and consistency are defined by the equality of the rows of T. For timed observation tables, we extend them with the congruence $\sim^S_{\mathcal{L}_{\mathrm{tgt}}}$ we introduced in Section 3. We note that consistency is defined only for discrete successors. This is because a timed observation table does not always become "consistent" for continuous successors. See Appendix C for a detailed discussion. We also require *exterior-consistency* since we construct an exterior from a successor.

Definition 26 (closedness, consistency, exterior-consistency, cohesion). *Let* $O = (P, S, T)$ *be a timed observation table.* O *is* closed *if, for each* $p \in \mathrm{succ}(P) \setminus P$*, there is* $p' \in P$ *satisfying* $p \sim^S_{\mathcal{L}_{\mathrm{tgt}}} p'$*.* O *is* consistent *if, for each* $p, p' \in P$ *and for each* $a \in \Sigma$*,* $p \sim^S_{\mathcal{L}_{\mathrm{tgt}}} p'$ *implies* $\mathrm{succ}^a(p) \sim^S_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$*.* O *is* exterior-consistent *if for any* $p \in P$*,* $\mathrm{succ}^t(p) \notin P$ *implies* $\mathrm{succ}^t(p) \subseteq \mathrm{ext}^t(p)$*.* O *is* cohesive *if it is closed, consistent, and exterior-consistent.*

Algorithm 2: DTA construction from a timed observation table

```
Input  : A cohesive timed observation table $(P, S, T)$
Output : A DTA $\mathcal{A}_{\mathrm{hyp}}$ row-faithful to the given timed observation table
1 Function MakeDTA$(P, S, T)$:
2     $\Phi \leftarrow \emptyset$; $F \leftarrow \{(u, \Lambda) \in P \mid T((u, \Lambda), (\varepsilon, \tau'_0 = 0)) = \{\Lambda \wedge \tau'_0 = 0\}\}$
3     for $p \in P$ such that $\mathrm{succ}^t(p) \notin P$ (resp. $\mathrm{succ}^a(p) \notin P$) do
          // Construct $(u, \Lambda, u', \Lambda', R)$ for some $p' \in P$ and $R$
          // Such $R$ is chosen using an exhaustive search
4         pick $p' \in P$ and $R$ such that $\mathrm{succ}^t(p) \sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} p'$ (resp. $\mathrm{succ}^a(p) \sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} p'$)
5         add $(u, \Lambda, u', \Lambda', R)$ to $\Phi$, where $(u, \Lambda) = \mathrm{ext}^t(p)$ (resp. $\mathrm{ext}^a(p)$) and
            $(u', \Lambda') = p'$
6     return the DTA $\mathcal{A}_{\mathrm{hyp}}$ obtained from $(P, F, \Phi)$ by the construction in [29]
```

From a cohesive timed observation table (P, S, T), we can construct a DTA as outlined in Algorithm 2. We construct a DTA via a recognizable timed language. The main ideas are as follows: 1) we approximate $\sim^{\mathcal{E}(\Sigma),R}_{\mathcal{L}_{\mathrm{tgt}}}$ by $\sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}}$; 2) we decide the equivalence class of an exterior $\mathrm{ext}(p) \in \mathrm{ext}(P) \setminus P$ in $\mathcal{E}(\Sigma)$ from successors. Notice that there can be multiple renaming equations R showing $\sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}}$. We use one of them found by an exhaustive search in Appendix B.2. The latter estimation is justified by the exterior-consistency of the timed observation table. We denote the resulting DTA of this construction by MakeDTA(P, S, T).

The DTA obtained by MakeDTA is faithful to the timed observation table in *rows*, i. e., for any $p \in P \cup \mathrm{succ}(P)$, $\mathcal{L}_{\mathrm{tgt}} \cap p = \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \cap p$. However, unlike the original L* algorithm, this does not (at least likely) hold for each *cell*, i. e., there may be $p \in P \cup \mathrm{succ}(P)$ and $s \in S$ satisfying $\mathcal{L}_{\mathrm{tgt}} \cap (p \cdot s) \neq \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \cap (p \cdot s)$. This is because we do not (and actually cannot) enforce consistency for continuous successors. See Appendix C for a discussion. It is future work to find a concrete example violating the faithfulness for each cell. Nevertheless, this does not affect the correctness of our DTA learning algorithm partly by Theorem 28.

Theorem 27 (row faithfulness). *For any cohesive timed observation table (P, S, T), for any $p \in P \cup \mathrm{succ}(P)$, $\mathcal{L}_{\mathrm{tgt}} \cap p = \mathcal{L}(\mathtt{MakeDTA}(P, S, T)) \cap p$ holds.* □

Theorem 28. *For any cohesive timed observation table (P, S, T), $\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} = \sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$ implies $\mathcal{L}_{\mathrm{tgt}} = \mathcal{L}(\mathtt{MakeDTA}(P, S, T))$.*

4.3 Counterexample analysis

We analyze the counterexample *cex* obtained by an equivalence query to refine the set S of suffixes in a timed observation table. Our analysis, outlined in Algorithm 3, is inspired by the Rivest-Schapire algorithm [33,28]. The idea is to reduce the counterexample *cex* using the mapping defined by the congruence $\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}}$ (lines 5–7), much like Φ in recognizable timed languages, and to find a suffix s strictly refining S (line 9), i. e., satisfying $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$ and $p \not\sim^{S \cup \{s\}}_{\mathcal{L}_{\mathrm{tgt}}} p'$ for some $p \in \mathrm{succ}(P)$ and $p' \in P$.

Algorithm 3: Counterexample analysis in our DTA learning algorithm

```
1  Function AnalyzeCEX($cex$):
2      $i \leftarrow 1$; $w_0 \leftarrow cex$
3      while $\nexists p \in P . w_i \in p$ do
4          $i \leftarrow i + 1$
5          split $w_{i-1}$ into $w'_i \cdot w''_i$ such that $w'_i \in p'_i$ for some $p'_i \in \mathrm{succ}(P) \setminus P$
6          let $p_i \in P$ and $R_i$ be such that $p'_i \sim^{S,R_i}_{\mathcal{L}_{\mathrm{tgt}}} p_i$
7          let $\overline{w}_i \in p_i$ be such that $\kappa(w'_i), \kappa(\overline{w}_i) \models R_i$
8          $w_i \leftarrow \overline{w}_i \cdot w''_i$
9      find $j \in \{1, 2, \dots, i\}$ such that $w_{j-1} \in \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ and
        $w_j \notin \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$
       // We use a binary search with membership queries for $\lceil \log(i) \rceil$
          times.
10     return the simple elementary language including $w''_j$
```

By definition of cex, we have $cex = w_0 \in \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. By Theorem 27, $w_n \notin \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ holds, where n is the final value of i. By construction of $\mathcal{A}_{\mathrm{hyp}}$ and w_i, for any $i \in \{1, 2, \dots, n\}$, we have $w_0 \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \iff w_i \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. Therefore, there is $i \in \{1, 2, \dots, n\}$ satisfying $w_{i-1} \in \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ and $w_i \notin \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. For such i, since we have $w_{i-1} = w'_i \cdot w''_i \in \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$, $w_i = \overline{w}_i \cdot w''_i \notin \mathcal{L}_{\mathrm{tgt}} \triangle \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$, and $\kappa(w'_i), \kappa(\overline{w}_i) \models R_i$, such w''_i is a witness of $p'_i \not\sim^{\mathcal{E}(\Sigma),R_i}_{\mathcal{L}_{\mathrm{tgt}}} p_i$. Therefore, S can be refined by the simple elementary language $s \in \mathcal{SE}(\Sigma)$ including w''_i.

4.4 L*-style learning algorithm for DTAs

Algorithm 4 outlines our active DTA learning algorithm. At line 1, we initialize the timed observation table with $P = \{(\varepsilon, \tau_0 = 0)\}$ and $S = \{(\varepsilon, \tau'_0 = 0)\}$. In the loop in lines 2–15, we refine the timed observation table (P, S, T) until the hypothesis DTA $\mathcal{A}_{\mathrm{hyp}}$ recognizes the target language $\mathcal{L}_{\mathrm{tgt}}$, which is checked by equivalence queries. The refinement finishes when the equivalence relation $\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}}$ defined by the suffixes S converges to $\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$, and the prefixes P covers $\mathcal{SE}(\Sigma)/\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$.

In the loop in lines 3–11, we make the timed observation table cohesive. If the timed observation table is not closed, we move the incompatible row in $\mathrm{succ}(P) \setminus P$ to P (line 5). If the timed observation table is inconsistent, we concatenate an event $a \in \Sigma$ in front of some of the suffixes in S (line 8). If the timed observation table is not exterior-consistent, we move the boundary $\mathrm{succ}^t(p) \in \mathrm{succ}^t(P) \setminus P$ satisfying $\mathrm{succ}^t(p) \nsubseteq \mathrm{ext}^t(p)$ to P (line 10). Once we obtain a cohesive timed observation table, we construct a DTA $\mathcal{A}_{\mathrm{hyp}} =$ `MakeDTA(`P, S, T`)` and make an equivalence query (line 12). If we have $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$, we return $\mathcal{A}_{\mathrm{hyp}}$. Otherwise, we have a timed word cex witnessing the difference between the language of the hypothesis DTA $\mathcal{A}_{\mathrm{hyp}}$ and the target language $\mathcal{L}_{\mathrm{tgt}}$. We refine the timed observation table using Algorithm 3.

Algorithm 4: Outline of our L*-style algorithm for DTA learning

```
$P \leftarrow \{(\varepsilon, \tau_0 = 0)\};\ S \leftarrow \{(\varepsilon, \tau'_0 = 0)\}$
while $\top$ do
    while $(P, S, T)$ is not cohesive do
        if $\exists p \in \mathrm{succ}(P) \setminus P.\, \nexists p' \in P.\, p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$ then        // $(P, S, T)$ is not closed
            $P \leftarrow P \cup \{p\}$                                // Add such $p$ to $P$
        else if $\exists p, p' \in P, a \in \Sigma.\, p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p' \wedge \mathrm{succ}^a(p) \not\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$ then
            // $(P, S, T)$ is inconsistent due to $a$
            let $S' \subseteq S$ be a minimal set such that $p \not\sim^{S \cup \{\{a \cdot w \mid w \in s\} \mid s \in S'\}}_{\mathcal{L}_{\mathrm{tgt}}} p'$
            $S \leftarrow S \cup \{\{a \cdot w \mid w \in s\} \mid s \in S'\}$
        else                        // $(P, S, T)$ is not exterior-consistent
            $P \leftarrow P \cup \{p' \in \mathrm{succ}^t(P) \setminus P \mid \exists p \in P.\, p' = \mathrm{succ}^t(p) \wedge p' \not\subseteq \mathrm{ext}^t(p)\}$
        fill $T$ using symbolic membership queries
    $\mathcal{A}_{\mathrm{hyp}} \leftarrow$ MakeDTA($P$, $S$, $T$)
    if $cex = \mathrm{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}})$ then
        add AnalyzeCEX($cex$) to $S$
    else return $\mathcal{A}_{\mathrm{hyp}}$              // It returns $\mathcal{A}_{\mathrm{hyp}}$ if $\mathrm{eq}_{\mathcal{L}_{\mathrm{tgt}}}(\mathcal{A}_{\mathrm{hyp}}) = \top$.
```

Example 29. Let $\mathcal{L}_{\mathrm{tgt}}$ be the timed language recognized by the DTA in Fig. 1c. We start from $P = \{(\varepsilon, \tau_0 = 0)\}$ and $S = \{(\varepsilon, \tau'_0 = 0\}$. Fig. 2a shows the initial timed observation table O_1. Since the timed observation table O_1 in Fig. 2a is cohesive, we construct a hypothesis DTA $\mathcal{A}^1_{\mathrm{hyp}}$. The hypothesis recognizable timed language is (P_1, F_1, Φ_1) is such that $P_1 = F_1 = \{(\varepsilon, \tau_0 = 0)\}$ and $\Phi_1 = \{(\varepsilon, \tau_0 > 0, \varepsilon, \tau_0, \top), (a, \tau_0 = \tau_0 + \tau_1 = \tau_1 = 0, \varepsilon, \tau_0, \top)\}$. Fig. 2b shows the first hypothesis DTA $\mathcal{A}^1_{\mathrm{hyp}}$.

We have $\mathcal{L}(\mathcal{A}^1_{\mathrm{hyp}}) \neq \mathcal{L}_{\mathrm{tgt}}$, and the learner obtains a counterexample, e. g., $cex = 1.0 \cdot a \cdot 0$, with an equivalence query. In Algorithm 3, we have $w_0 = cex$, $w_1 = 0.5 \cdot a \cdot 0$, $w_2 = 0 \cdot a \cdot 0$, and $w_3 = 0$. We have $w_0 \notin \mathcal{L}(\mathcal{A}^1_{\mathrm{hyp}}) \triangle \mathcal{L}_{\mathrm{tgt}}$ and $w_1 \in \mathcal{L}(\mathcal{A}^1_{\mathrm{hyp}}) \triangle \mathcal{L}_{\mathrm{tgt}}$, and the suffix to distinguish w_0 and w_1 is $0.5 \cdot a \cdot 0$. Thus, we add $s_1 = (a, \tau'_1 = 0 < \tau'_0 = \tau'_0 + \tau'_1 < 1)$ to S_1 (Fig. 2d).

In Fig. 2d, we observe that $T_2(p_1, s_1)$ is more strict than $T_2(p_0, s_1)$, and we have $p_1 \not\sim^{S_2}_{\mathcal{L}_{\mathrm{tgt}}} p_0$. To make (P_2, S_2, T_2) closed, we add p_1 to P_2. By repeating similar operations, we obtain the timed observation table $O_3 = (P_3, S_3, T_3)$ in Fig. 2e, which is cohesive. Fig. 2c shows the DTA $\mathcal{A}^3_{\mathrm{hyp}}$ constructed from O_3. Since $\mathcal{L}(\mathcal{A}^3_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$ holds, Algorithm 4 finishes returning $\mathcal{A}^3_{\mathrm{hyp}}$.

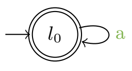
By the use of equivalence queries, Algorithm 4 returns a DTA recognizing the target language if it terminates, which is formally as follows.

Theorem 30 (correctness). *For any target timed language $\mathcal{L}_{\mathrm{tgt}}$, if Algorithm 4 terminates, for the resulting DTA $\mathcal{A}_{\mathrm{hyp}}$, $\mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) = \mathcal{L}_{\mathrm{tgt}}$ holds.* □

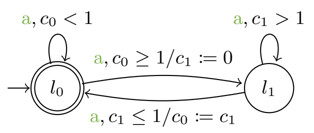
Moreover, Algorithm 4 terminates for any recognizable timed language $\mathcal{L}_{\mathrm{tgt}}$. This is essentially because of the finiteness of $\mathcal{SE}(\Sigma)/\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$.

	$(\varepsilon, \tau'_0 = 0)$
$(\varepsilon, \tau_0 = 0)$	$\tau_0 = \tau'_0 = 0$
$(\varepsilon, \tau_0 \in (0, 1))$	$0 = \tau'_0 < \tau_0 < 1$
$(\mathrm{a}, \tau_0 = \tau_1 = 0)$	$\tau_0 = \tau_1 = \tau'_0 = 0$

(a) The initial timed observation table $O_1 = (P_1, S_1, T_1)$



(b) DTA $\mathcal{A}^1_{\mathrm{hyp}}$ constructed from O_1



(c) DTA $\mathcal{A}^3_{\mathrm{hyp}}$ constructed from O_3

	$s_0 = (\varepsilon, \tau'_0 = 0)$	$s_1 = (\mathrm{a}, \tau'_1 = 0 < \tau'_0 < 1)$
$p_0 = (\varepsilon, \tau_0 = 0)$	$\top$	$\top$
$p_1 = (\varepsilon, \tau_0 \in (0, 1))$	$\top$	$\tau_0 + \tau'_0 \in (0, 1)$
$p_2 = (\mathrm{a}, \tau_0 = \tau_0 + \tau_1 = \tau_1 = 0)$	$\top$	$\top$

(d) Timed observation table $O_2 = (P_2, S_2, T_2)$ after processing *cex*

	$(\varepsilon, \tau'_0 = 0)$	$(\mathrm{a}, \tau'_1 = 0 < \tau'_0 < 1)$
$(\varepsilon, \tau_0 = 0)$	$\top$	$\top$
$(\varepsilon, \tau_0 \in (0, 1))$	$\top$	$\tau_0 + \tau'_0 \in (0, 1)$
$(\varepsilon, \tau_0 = 1)$	$\top$	$\bot$
$(\mathrm{a}, \tau_0 = \tau_0 + \tau_1 = 1 \land \tau_1 = 0)$	$\bot$	$\top$
$(\mathrm{a}, \tau_0 = 1 \land \tau_1 \in (0, 1))$	$\bot$	$\tau_1 + \tau'_0 \in (0, 1]$
$(\mathrm{a}, \tau_0 = \tau_1 = 1 \land \tau_0 + \tau_1 = 2)$	$\bot$	$\bot$
$(\mathrm{a}, \tau_0 = \tau_0 + \tau_1 = \tau_1 = 0)$	$\top$	$\top$
$(\mathrm{a}, \tau_0 = \tau_0 + \tau_1 \in (0, 1) \land \tau_1 = 0)$	$\top$	$\tau_0 + \tau_1 + \tau'_0 \in (0, 1)$
$(\varepsilon, \tau_0 \in (1, 2))$	$\top$	$\bot$
$(\mathrm{aa}, \tau_0 = \tau_0 + \tau_1 = \tau_0 + \tau_1 + \tau_2 = 1 \land \tau_1 = \tau_2 = \tau_1 + \tau_2 = 0)$	$\top$	$\top$
$(\mathrm{aa}, \tau_0 = 1 \land \tau_1 = \tau_1 + \tau_2 \in (0, 1) \land \tau_0 + \tau_1 = \tau_0 + \tau_1 + \tau_2 \in (1, 2) \land \tau_2 = 0)$	$\top$	$\tau_1 + \tau_2 + \tau'_0 \in (0, 1)$
$(\mathrm{a}, \tau_0 = 1 < \tau_1 < 2 < \tau_0 + \tau_1 < 3)$	$\bot$	$\bot$
$(\mathrm{aa}, \tau_0 = \tau_1 = \tau_1 + \tau_2 = 1 \land \tau_0 + \tau_1 = \tau_0 + \tau_1 + \tau_2 = 2 \land \tau_2 = 0)$	$\top$	$\bot$

(e) The final observation table $O_3 = (P_3, S_3, T_3)$

Fig. 2: Timed observation tables O_1, O_2, O_3, and the DTAs $\mathcal{A}^1_{\mathrm{hyp}}$ and $\mathcal{A}^3_{\mathrm{hyp}}$ made from O_1 and O_3, respectively. In O_2 and O_3, we only show the constraints non-trivial from p and s. The DTAs are simplified without changing the language. The use of clock assignments, which does not change the expressiveness, is from [29].

Theorem 31 (termination). *For any recognizable timed language $\mathcal{L}_{\mathrm{tgt}}$, Algorithm 4 terminates and returns a DTA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}_{\mathrm{tgt}}$.*

Proof (Theorem 31). By the recognizability of $\mathcal{L}_{\mathrm{tgt}}$ and Theorem 20, $\mathcal{SE}(\Sigma)/{\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}}$ is finite. Let $N = |\mathcal{SE}(\Sigma)/{\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}}|$. Since each execution of line 5 adds p to P, where p is such that for any $p' \in P$, $p \not\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}} p'$ holds, it is executed at most N times. Since each execution of line 8 strictly refines S, i. e., it increases $|\mathcal{SE}(\Sigma)/{\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}}}|$, line 8 is executed at most N times. For any simple elementary language $(u, \Lambda) \in \mathcal{SE}(\Sigma)$, if Λ contains $\mathbb{T}_{i,|u|} = d$ for some $i \in \{0, 1, \dots, |u|\}$ and $d \in \mathbb{N}$, we have $\mathrm{succ}^t((u, \Lambda)) \subseteq \mathrm{ext}^t((u, \Lambda))$. Therefore, line 10 is executed at most N times. Since S is strictly refined in line 14, i. e., it increases $|\mathcal{SE}(\Sigma)/{\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}}}|$, line 14 is executed at most N times. By Theorem 28, once $\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}}$ saturates to $\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$, `MakeDTA` returns the correct DTA. Overall, Algorithm 4 terminates. $\square$

4.5 Learning with a normal teacher

We briefly show how to learn a DTA only with membership and equivalence queries. See Appendix B.1 for detail. We reduce a symbolic membership query to finitely many membership queries, which can be answered by a normal teacher.

Let (u, Λ) be the elementary language given in a symbolic membership query. Since Λ is bounded, we can construct a finite and disjoint set of simple and canonical timed conditions $\Lambda'_1, \Lambda'_2, \dots, \Lambda'_n$ satisfying $\bigvee_{1 \leq i \leq n} \Lambda'_i = \Lambda$ by a simple enumeration. For any simple elementary language $(u', \Lambda') \in \mathcal{SE}(\Sigma)$ and timed words $w, w' \in (u', \Lambda')$, we have $w \in \mathcal{L} \iff w' \in \mathcal{L}$. Thus, we can construct $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda))$ by making a membership query $\mathtt{mem}_{\mathcal{L}}(w)$ for each such $(u', \Lambda') \subseteq (u, \Lambda)$ and for some $w \in (u', \Lambda')$. We need such an exhaustive search, instead of a binary search, because $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda))$ may be non-convex.

Assume Λ is a canonical timed condition. Let M be the size of the variables in Λ and I be the largest difference between the upper bound and the lower bound for some $\mathbb{T}_{i,j}$ in Λ. The size n of the above decomposition is bounded by $(2 \times I + 1)^{1/2 \times M \times (M+1)}$, which exponentially blows up with respect to M.

In our algorithm, we only make symbolic membership queries with elementary languages of the form $p \cdot s$, where p and s are simple elementary languages. Therefore, I is at most 2. However, even with such an assumption, the number of the necessary membership queries blows up exponentially to the size of the variables in Λ.

4.6 Complexity analysis

After each equivalence query, our DTA learning algorithm strictly refines S or terminates. Thus, the number of equivalence queries is at most N. In the proof of Theorem 31, we observe that the size of P is at most $2N$. Therefore, the number $(|P| + |\mathrm{succ}(P)|) \times |S|$ of the cells in the timed observation table is at most $(2N + 2N \times (|\Sigma| + 1)) \times N = 2N^2|\Sigma| + 2$. Let J be the upper bound of i in the analysis of *cex* returned by equivalence queries (Algorithm 3). For each equivalence query, the number of membership queries in Algorithm 3 is bounded by $\lceil \log J \rceil$, and thus, it is, in total, bounded by $N \times \lceil \log J \rceil$. Therefore, if the learner can use symbolic membership queries, the total number of queries is bounded by a polynomial of N and J. In Section 4.5, we observe that the number of membership queries to implement a symbolic membership query is at most exponential to M. Since P is prefix-closed, M is at most N. Overall, if the learner cannot use symbolic membership queries, the total number of queries is at most exponential to N.

Let $\mathcal{A}_{\mathrm{tgt}} = (\Sigma, L, l_0, C, I, \Delta, F)$ be a DTA recognizing $\mathcal{L}_{\mathrm{tgt}}$. As we observe in the proof of Lemma 33, N is bounded by the size of the state space of the RA of $\mathcal{A}_{\mathrm{tgt}}$, and N is at most $|C|! \times 2^{|C|} \times \prod_{c \in C}(2K_c + 2) \times |L|$, where K_c is the largest constant compared with $c \in C$ in $\mathcal{A}_{\mathrm{tgt}}$. Thus, without symbolic membership queries, the total number of queries is at most doubly-exponential to $|C|$ and singly exponential to $|L|$. We remark that when $|C| = 1$, the total number of queries is at most singly exponential to $|L|$ and K_c, which coincides with the worst-case complexity of the one-clock DTA learning algorithm in [39].

Table 1: Summary of the results for Random. Each row index $|L|\_|\Sigma|\_K_C$ shows the number of locations, the alphabet size, and the upper bound of the maximum constant in the guards, respectively. The row "count" shows the number of instances finished in 3 hours. Cells with the best results are highlighted.

		# of Mem. queries			# of Eq. queries			Exec. time [sec.]			count
		max	mean	min	max	mean	min	max	mean	min	
3_2_10	LearnTA	35,268	14,241	2,830	**11**	**6**	**4**	2.32e+00	6.68e-01	**4.50e-02**	**10/10**
	OneSMT	**468**	**205**	**32**	13	8	5	**9.58e-01**	**2.89e-01**	6.58e-02	**10/10**
4_2_10	LearnTA	194,442	55,996	10,619	**14**	**7**	**4**	2.65e+01	7.98e+00	4.88e-01	**10/10**
	OneSMT	**985**	**451**	**255**	16	12	7	**3.53e-01**	**2.09e-01**	**1.27e-01**	**10/10**
4_4_20	LearnTA	1,681,769	858,759	248,399	**21**	**15**	**10**	8.34e+03	1.41e+03	3.23e+01	8/10
	OneSMT	**5,329**	**3,497**	**1,740**	42	32	26	**2.19e+00**	**1.42e+00**	**8.27e-01**	**10/10**
5_2_10	LearnTA	627,980	119,906	8,121	**19**	**8**	**5**	1.67e+02	2.28e+01	**1.96e-01**	**10/10**
	OneSMT	**1,332**	**876**	**359**	22	16	12	**5.20e-01**	**3.66e-01**	2.58e-01	**10/10**
6_2_10	LearnTA	555,939	106,478	2,912	**14**	**9**	**6**	2.44e+02	2.81e+01	**4.40e-02**	**10/10**
	OneSMT	**3,929**	**1,894**	**104**	35	20	11	**1.72e+00**	**8.01e-01**	1.73e-01	**10/10**

5 Experiments

We experimentally evaluated our DTA learning algorithm using our prototype library LearnTA[5] implemented in C++. In LearnTA, the equivalence queries are answered by a zone-based reachability analysis using the fact that DTAs are closed under complement [6]. We pose the following research questions.

RQ1 How is the scalability of LearnTA with respect to the language complexity?
RQ2 How is the efficiency of LearnTA for practical benchmarks?

For the benchmarks with one clock variable, we compared LearnTA with one of the latest one-clock DTA learning algorithms [39,1], which we call OneSMT. OneSMT is implemented in Python with Z3 [31] for constraint solving.

For each execution, we measured

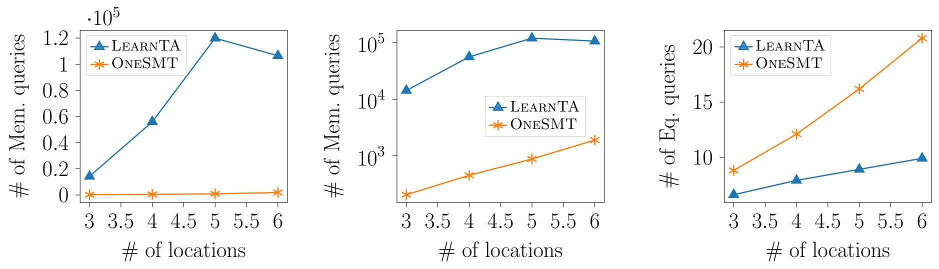
- the number of queries and
- the total execution time, including the time to answer the queries.

For the number of queries, we report the number with memoization, i. e., we count the number of the queried timed words (for membership queries) and the counterexamples (for equivalence queries). We conducted all the experiments on a computing server with Intel Core i9-10980XE 125 GiB RAM that runs Ubuntu 20.04.5 LTS. We used 3 hours as the timeout.

5.1 RQ1: Scalability with respect to the language complexity

To evaluate the scalability of LearnTA, we used randomly generated DTAs from [7] (denoted as Random) and our original DTAs (denoted as Unbalanced). Random consists of five classes: 3_2_10, 4_2_10, 4_4_20, 5_2_10, and 6_2_10, where

[5] LearnTA is publicly available at https://github.com/masWag/LearnTA. The artifact of the experiments is available at https://doi.org/10.5281/zenodo.7875383.



(a) Membership queries (b) Membership queries (log scale) (c) Equivalence queries

Fig. 3: The number of locations and the number of queries for $|L|$_2_10 in Random, where $|L| \in \{3, 4, 5, 6\}$

Table 2: Summary of the target DTAs and the results for Unbalanced. $|L|$ is the number of locations, $|\Sigma|$ is the alphabet size, $|C|$ is the number of clock variables, and K_C is the maximum constant in the guards in the DTA.

	$\lvert L\rvert$	$\lvert\Sigma\rvert$	$\lvert C\rvert$	K_C	# of Mem. queries	# of Eq. queries	Exec. time [sec.]
Unbalanced:1 LearnTA	5	1	1	2	51	2	2.00e-03
Unbalanced:2 LearnTA	5	1	2	4	576,142	3	3.64e+01
Unbalanced:3 LearnTA	5	1	3	4	403,336	4	2.24e+01
Unbalanced:4 LearnTA	5	1	4	6	4,142,835	5	2.40e+02
Unbalanced:5 LearnTA	5	1	5	6	10,691,400	5	8.68e+02

each value of $|L|\_|\Sigma|\_K_C$ is the number of locations, the alphabet size, and the upper bound of the maximum constant in the guards in the DTAs, respectively. Each class consists of 10 randomly generated DTAs. Unbalanced is our original benchmark inspired by the "unbalanced parentheses" timed language from [12]. Unbalanced consists of five DTAs with different complexity of timing constraints. Table 2 summarizes their complexity.

Table 1 and Fig. 3 summarize the results for Random, and Table 2 summarizes the results for Unbalanced. Table 1 shows that LearnTA requires more membership queries than OneSMT. This is likely because of the difference in the definition of prefixes and successors: OneSMT's definitions are discrete (e. g., prefixes are only with respect to events with time elapse), whereas ours are both continuous and discrete (e. g., we also consider prefixes by trimming the dwell time in the end); Since our definition makes significantly more prefixes, LearnTA tends to require much more membership queries. Another, more high-level reason is that LearnTA learns a DTA without knowing the number of the clock variables, and many more timed words are potentially helpful for learning. Table 1 shows that LearnTA requires significantly many membership queries for 4_4_20. This is likely because of the exponential blowup with respect to K_C, as discussed in Section 4.6. In Fig. 3, we observe that for both LearnTA and OneSMT, the number of membership queries increases nearly exponentially to the number of locations. This coincides with the discussion in Section 4.6.

In contrast, Table 1 shows that LearnTA requires fewer equivalence queries than OneSMT. This suggests that the cohesion in Definition 26 successfully detected contradictions in observation before generating a hypothesis, whereas

Table 3: Summary of the target DTA and the results for practical benchmarks. The columns are the same as Table 2. Cells with the best results are highlighted.

		$\|L\|$	$\|\Sigma\|$	$\|C\|$	K_C	# of Mem. queries	# of Eq. queries	Exec. time [sec.]
AKM	LearnTA	17	12	1	5	12,263	**11**	**5.85e-01**
	OneSMT	17	12	1	5	**3,453**	49	7.97e+00
CAS	LearnTA	14	10	1	27	66,067	**17**	**4.65e+00**
	OneSMT	14	10	1	27	**4,769**	18	9.58e+01
Light	LearnTA	5	5	1	10	3,057	**7**	**3.30e-02**
	OneSMT	5	5	1	10	**210**	7	9.32e-01
PC	LearnTA	26	17	1	10	245,134	**23**	**6.49e+01**
	OneSMT	26	17	1	10	**10,390**	29	1.24e+02
TCP	LearnTA	22	13	1	2	11,300	**15**	**3.82e-01**
	OneSMT	22	13	1	2	**4,713**	32	2.20e+01
Train	LearnTA	6	6	1	10	13,487	**8**	**1.72e-01**
	OneSMT	6	6	1	10	**838**	13	1.13e+00
FDDI	LearnTA	16	5	7	6	**9,986,271**	**43**	**3.00e+03**

OneSMT mines timing constraints mainly by equivalence queries and tends to require more equivalence queries. In Fig. 3c, we observe that for both LearnTA and OneSMT, the number of equivalence queries increases nearly linearly to the number of locations. This also coincides with the complexity analysis in Section 4.6. Fig. 3c also shows that the number of equivalence queries increases faster in OneSMT than in LearnTA.

Table 2 also suggests a similar tendency: the number of membership queries rapidly increases to the complexity of the timing constraints; In contrast, the number of equivalence queries increases rather slowly. Moreover, LearnTA is scalable enough to learn a DTA with five clock variables within 15 minutes.

Table 1 also suggests that LearnTA does not scale well to the maximum constant in the guards, as observed in the analysis in Section 4.6. However, we still observe that LearnTA requires fewer equivalence queries than OneSMT. Overall, compared with OneSMT, LearnTA has better scalability in the number of equivalence queries and worse scalability in the number of membership queries.

5.2 RQ2: Performance on practical benchmarks

To evaluate the practicality of LearnTA, we used seven benchmarks: AKM, CAS, Light, PC, TCP, Train, and FDDI. Table 3 summarizes their complexity. All the benchmarks other than FDDI are taken from [39] (or its implementation [1]). FDDI is taken from TChecker [2]. We use the instance of FDDI with two processes.

Table 3 summarizes the results for the benchmarks from practical applications. We observe, again, that LearnTA requires more membership queries and fewer equivalence queries than OneSMT. However, for these benchmarks, the difference in the number of membership queries tends to be much smaller than in Random. This is because these benchmarks have simpler timing constraints than Random for the exploration by LearnTA. In AKM, Light, PC, TCP, and Train, the clock variable can be reset at every edge without changing the language, i. e., there are *real-time automata* [19] recognizing the same language. For such a DTA, all simple elementary languages are equivalent in terms of the Nerode-style congruence in Section 3 if we have the same edge at their last event and the same dwell time after it. If two simple elementary languages are equivalent, LearnTA explores the successors of only one of them, and the exploration is relatively efficient. We have a similar situation in CAS. Moreover, in many of these DTAs, only a few edges have guards. Overall, despite the large number of locations and alphabets, these languages' complexities are mild for LearnTA.

We also observe that, surprisingly, for all of these benchmarks, LearnTA took a shorter time for DTA learning than OneSMT. This is partly because of the difference in the implementation language (i. e., C++ vs. Python) but also because of the small number of equivalence queries and the mild number of membership queries. Moreover, although it requires significantly more queries, LearnTA successfully learned FDDI with seven clock variables. Overall, such efficiency on benchmarks from practical applications suggests the potential usefulness of LearnTA in some realistic scenarios.

6 Conclusions and future work

Extending the L* algorithm, we proposed an active learning algorithm for DTAs. Our extension is by our Nerode-style congruence for recognizable timed languages. We proved the termination and the correctness of our algorithm. We also proved that our learning algorithm requires a polynomial number of queries with a smart teacher and an exponential number of queries with a normal teacher. Our experiment results also suggest the practical relevance of our algorithm.

One of the future directions is to extend more recent automata learning algorithms (e. g., TTT algorithm [27] to improve the efficiency) to DTA learning. Another direction is constructing a *passive* DTA learning algorithm based on our congruence and an existing passive DFA learning algorithm. It is also a future direction to apply our learning algorithm for practical usage, e. g., identification of black-box systems and testing black-box systems with black-box checking [32,30,38]. Optimization of the algorithm, e. g., by incorporating clock information is also a future direction.

Acknowledgements. This work is partially supported by JST ACT-X Grant No. JPMJAX200U, JST PRESTO Grant No. JPMJPR22CA, JST CREST Grant No. JPMJCR2012, and JSPS KAKENHI Grant No. 22K17873.

References

1. GitHub: Leslieaj/DOTALearningSMT. https://github.com/Leslieaj/DOTALearningSMT, accessed: 2023-01-10
2. Github: ticktac-project/tchecker. https://github.com/ticktac-project/tchecker, accessed: 2023-01-20
3. Aichernig, B.K., Auer, J., Jöbstl, E., Korosec, R., Krenn, W., Schlick, R., Schmidt, B.V.: Model-based mutation testing of an industrial measurement device. In: Seidl, M., Tillmann, N. (eds.) Tests and Proofs - 8th International Conference, TAP@STAF 2014, York, UK, July 24-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8570, pp. 1–19. Springer (2014). https://doi.org/10.1007/978-3-319-09099-3_1, https://doi.org/10.1007/978-3-319-09099-3_1
4. Aichernig, B.K., Brandl, H., Jöbstl, E., Krenn, W.: UML in action: a two-layered interpretation for testing. ACM SIGSOFT Softw. Eng. Notes **36**(1), 1–8 (2011). https://doi.org/10.1145/1921532.1921559, https://doi.org/10.1145/1921532.1921559
5. Aichernig, B.K., Pferscher, A., Tappler, M.: From passive to active: Learning timed automata efficiently. In: Lee, R., Jha, S., Mavridou, A. (eds.) NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11-15, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12229, pp. 1–19. Springer (2020). https://doi.org/10.1007/978-3-030-55754-6_1, https://doi.org/10.1007/978-3-030-55754-6_1
6. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8, http://dx.doi.org/10.1016/0304-3975(94)90010-8
7. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12078, pp. 444–462. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_25, https://doi.org/10.1007/978-3-030-45190-5_25
8. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. Sci. China Inf. Sci. **64**(9) (2021). https://doi.org/10.1007/s11432-019-2767-4, https://doi.org/10.1007/s11432-019-2767-4
9. An, J., Zhan, B., Zhan, N., Zhang, M.: Learning nondeterministic real-time automata. ACM Trans. Embed. Comput. Syst. **20**(5s), 99:1–99:26 (2021). https://doi.org/10.1145/3477030, https://doi.org/10.1145/3477030
10. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6, https://doi.org/10.1016/0890-5401(87)90052-6
11. Argyros, G., D'Antoni, L.: The learnability of symbolic automata. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 427–445. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_23, https://doi.org/10.1007/978-3-319-96145-3_23

12. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM **49**(2), 172–206 (2002). https://doi.org/10.1145/506147.506151, http://doi.acm.org/10.1145/506147.506151
13. Bersani, M.M., Rossi, M., San Pietro, P.: A logical characterization of timed regular languages. Theor. Comput. Sci. **658**, 46–59 (2017). https://doi.org/10.1016/j.tcs.2016.07.020, https://doi.org/10.1016/j.tcs.2016.07.020
14. Bojanczyk, M., Lasota, S.: A machine-independent characterization of timed languages. In: Czumaj, A., Mehlhorn, K., Pitts, A.M., Wattenhofer, R. (eds.) Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II. Lecture Notes in Computer Science, vol. 7392, pp. 92–103. Springer (2012). https://doi.org/10.1007/978-3-642-31585-5_12, https://doi.org/10.1007/978-3-642-31585-5_12
15. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable timed automata. Theor. Comput. Sci. **321**(2-3), 291–345 (2004). https://doi.org/10.1016/j.tcs.2004.04.003, https://doi.org/10.1016/j.tcs.2004.04.003
16. Bouyer, P., Petit, A., Thérien, D.: An algebraic characterization of data and timed languages. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2154, pp. 248–261. Springer (2001). https://doi.org/10.1007/3-540-44685-0_17, https://doi.org/10.1007/3-540-44685-0_17
17. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA. Lecture Notes in Computer Science, vol. 1066, pp. 208–219. Springer (1995). https://doi.org/10.1007/BFb0020947, https://doi.org/10.1007/BFb0020947
18. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings. Lecture Notes in Computer Science, vol. 407, pp. 197–212. Springer (1989). https://doi.org/10.1007/3-540-52148-8_17, http://dx.doi.org/10.1007/3-540-52148-8_17
19. Dima, C.: Real-time automata. J. Autom. Lang. Comb. **6**(1), 3–23 (2001). https://doi.org/10.25596/jalc-2001-003, https://doi.org/10.25596/jalc-2001-003
20. Drews, S., D'Antoni, L.: Learning symbolic automata. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 173–189 (2017). https://doi.org/10.1007/978-3-662-54577-5_10, https://doi.org/10.1007/978-3-662-54577-5_10
21. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theor. Comput. Sci. **411**(47), 4029–4054 (2010). https://doi.org/10.1016/j.tcs.2010.07.008, https://doi.org/10.1016/j.tcs.2010.07.008

22. Grinchtein, O., Jonsson, B., Pettersson, P.: Inference of event-recording automata using timed decision trees. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4137, pp. 435–449. Springer (2006). https://doi.org/10.1007/11817949_29, https://doi.org/10.1007/11817949_29
23. Guha, S., Narayan, C., Arun-Kumar, S.: Reducing clocks in timed automata while preserving bisimulation. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8704, pp. 527–543. Springer (2014). https://doi.org/10.1007/978-3-662-44584-6_36, https://doi.org/10.1007/978-3-662-44584-6_36
24. Henry, L., Jéron, T., Markey, N.: Active learning of timed automata with unobservable resets. In: Bertrand, N., Jansen, N. (eds.) Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12288, pp. 144–160. Springer (2020). https://doi.org/10.1007/978-3-030-57628-8_9, https://doi.org/10.1007/978-3-030-57628-8_9
25. Hessel, A., Larsen, K.G., Nielsen, B., Pettersson, P., Skou, A.: Time-optimal real-time test case generation using uppaal. In: Petrenko, A., Ulrich, A. (eds.) Formal Approaches to Software Testing, Third International Workshop on Formal Approaches to Testing of Software, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Lecture Notes in Computer Science, vol. 2931, pp. 114–130. Springer (2003). https://doi.org/10.1007/978-3-540-24617-6_9, https://doi.org/10.1007/978-3-540-24617-6_9
26. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 3rd Edition. Pearson international edition, Addison-Wesley (2007)
27. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 307–322. Springer (2014). https://doi.org/10.1007/978-3-319-11164-3_26, https://doi.org/10.1007/978-3-319-11164-3_26
28. Isberner, M., Steffen, B.: An abstract framework for counterexample analysis in active automata learning. In: Clark, A., Kanazawa, M., Yoshinaka, R. (eds.) Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014. JMLR Workshop and Conference Proceedings, vol. 34, pp. 79–93. JMLR.org (2014), http://proceedings.mlr.press/v34/isberner14a.html
29. Maler, O., Pnueli, A.: On recognizable timed languages. In: Walukiewicz, I. (ed.) Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2987, pp. 348–362. Springer (2004). https://doi.org/10.1007/978-3-540-24727-2_25, https://doi.org/10.1007/978-3-540-24727-2_25
30. Meijer, J., van de Pol, J.: Sound black-box checking in the learnlib. Innov. Syst. Softw. Eng. **15**(3-4), 267–287 (2019).

https://doi.org/10.1007/s11334-019-00342-6, https://doi.org/10.1007/s11334-019-00342-6

31. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
32. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China. IFIP Conference Proceedings, vol. 156, pp. 225–240. Kluwer (1999)
33. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993). https://doi.org/10.1006/inco.1993.1021, https://doi.org/10.1006/inco.1993.1021
34. Saeedloei, N., Kluzniak, F.: Clock allocation in timed automata and graph colouring. In: Prandini, M., Deshmukh, J.V. (eds.) Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018. pp. 71–80. ACM (2018). https://doi.org/10.1145/3178126.3178138, https://doi.org/10.1145/3178126.3178138
35. Shijubo, J., Waga, M., Suenaga, K.: Efficient black-box checking via model checking with strengthened specifications. In: Feng, L., Fisman, D. (eds.) Runtime Verification - 21st International Conference, RV 2021, Virtual Event, October 11-14, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12974, pp. 100–120. Springer (2021). https://doi.org/10.1007/978-3-030-88494-9_6, https://doi.org/10.1007/978-3-030-88494-9_6
36. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: André, É., Stoelinga, M. (eds.) Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11750, pp. 216–235. Springer (2019). https://doi.org/10.1007/978-3-030-29662-9_13, https://doi.org/10.1007/978-3-030-29662-9_13
37. Vaandrager, F.W., Bloem, R., Ebrahimi, M.: Learning mealy machines with one timer. In: Leporati, A., Martín-Vide, C., Shapira, D., Zandron, C. (eds.) Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12638, pp. 157–170. Springer (2021). https://doi.org/10.1007/978-3-030-68195-1_13, https://doi.org/10.1007/978-3-030-68195-1_13
38. Waga, M.: Falsification of cyber-physical systems with robustness-guided black-box checking. In: Ames, A.D., Seshia, S.A., Deshmukh, J. (eds.) HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Con-

trol, Sydney, New South Wales, Australia, April 21-24, 2020. pp. 11:1–11:13. ACM (2020). https://doi.org/10.1145/3365365.3382193, https://doi.org/10.1145/3365365.3382193

39. Xu, R., An, J., Zhan, B.: Active learning of one-clock timed automata using constraint solving. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13505, pp. 249–265. Springer (2022). https://doi.org/10.1007/978-3-031-19992-9_16, https://doi.org/10.1007/978-3-031-19992-9_16
40. Zhang, H., Feng, L., Li, Z.: Control of black-box embedded systems by integrating automaton learning and supervisory control theory of discrete-event systems. IEEE Trans Autom. Sci. Eng. **17**(1), 361–374 (2020). https://doi.org/10.1109/TASE.2019.2929563, https://doi.org/10.1109/TASE.2019.2929563

A Omitted proofs

A.1 Proof of Theorem 18

Here, we prove Theorem 18. First, we show the following lemma.

Lemma 32. *Let $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ be a timed language and let $(u, \Lambda), (u', \Lambda'), (u'', \Lambda'') \in \mathcal{E}(\Sigma)$ be elementary languages. For any timed words $w \in (u, \Lambda)$, $w' \in (u', \Lambda')$, and $w'' \in (u'', \Lambda'')$, if we have $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda' \Rightarrow \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$, $\kappa(w), \kappa(w') \models R$, and $w \cdot w'' \in \mathcal{L}$, we also have $w' \cdot w'' \in \mathcal{L}$.*

Proof. Since we have $w \cdot w'' \in \mathcal{L}$, $\kappa(w), \kappa(w'') \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$ holds. Since we have $\kappa(w), \kappa(w') \models R$, we have $\kappa(w), \kappa(w'), \kappa(w'') \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R$. Since $w' \in (u', \Lambda')$, we have $\kappa(w') \models \Lambda'$, and we have $\kappa(w), \kappa(w'), \kappa(w'') \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda'$. Because of $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda' \Rightarrow \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$, we have $\kappa(w), \kappa(w'), \kappa(w'') \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$. Since $\kappa(w)$ is over $\mathbb{T}$ and $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$ is over $\mathbb{T}'$ and $\mathbb{T}''$, we have $\kappa(w'), \kappa(w'') \models \mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$. Therefore, we have $w' \cdot w'' \in \mathcal{L}$

The following proves Theorem 18.

Theorem 18 (recalled). *For any $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$ and $(u, \Lambda), (u', \Lambda'), (u'', \Lambda'') \in \mathcal{E}(\Sigma)$ satisfying $(u, \Lambda) \sqsubseteq^{(u'', \Lambda'')}_{\mathcal{L}} (u', \Lambda')$, for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $w'' \in (u'', \Lambda'')$, $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$ holds.*

Proof. Let R be the renaming equation satisfying $(u, \Lambda) \sqsubseteq^{(u'', \Lambda''), R}_{\mathcal{L}} (u', \Lambda')$. By the definition of $(u, \Lambda) \sqsubseteq^{(u'', \Lambda''), R}_{\mathcal{L}} (u', \Lambda')$, we have $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \land R \land \Lambda'$ if and only if $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \land R \land \Lambda$. By Lemma 32, for any $w \in (u, \Lambda)$, $w' \in (u', \Lambda')$, and $w'' \in (u'', \Lambda'')$ satisfying $\kappa(w), \kappa(w') \models R$, we have $w \cdot w'' \in \mathcal{L}$ if and only if $w' \cdot w'' \in \mathcal{L}$. By the definition of $(u, \Lambda) \sqsubseteq^{(u'', \Lambda''), R}_{\mathcal{L}} (u', \Lambda')$, for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ satisfying $\kappa(w), \kappa(w) \models R$. Therefore, for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $w'' \in (u'', \Lambda'')$, we have $w \cdot w'' \in \mathcal{L} \iff w' \cdot w'' \in \mathcal{L}$.

A.2 Proof of Theorem 20

Theorem 20 is immediately obtained from the following lemmas.

Lemma 33. *For any recognizable timed language $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$, the quotient set $\mathcal{SE}(\Sigma)/{\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}}}$ is finite.* ◻

Lemma 34. *For any timed language $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$, if the quotient set $\mathcal{SE}(\Sigma)/{\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}}}$ is finite, $\mathcal{L}$ is recognizable.*

Proof (sketch). Let $P \subseteq \mathcal{SE}(\Sigma)$ be a finite set of simple elementary languages representing $\mathcal{SE}(\Sigma)/{\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}}}$, i. e., for any $p \in \mathcal{SE}(\Sigma)$, there is $p' \in P$ satisfying $p \sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}} p'$. We construct $\tilde{P}$ by augmenting P as follows: 1) for each $(u, \Lambda) \in P$, we add all its prefixes to $\tilde{P}$ so that $\tilde{P}$ is prefix-closed; 2) for each $(u, \Lambda) \in P$ such that Λ is a simple and canonical timed condition if Λ does not contain equalities and P does not contain any of its time successors (u', Λ'), i. e., $(u', \Lambda') \neq (u, \Lambda)$ such that there are $w \in (u, \Lambda)$ and $t \in \mathbb{R}_{>0}$ satisfying $w \cdot t \in (u', \Lambda')$, we add such (u', Λ') with the shortest dwell time to $\tilde{P}$. We note that $\tilde{P}$ is also a finite set. Let $F = \{(u, \Lambda) \in \tilde{P} \mid (u, \Lambda) \subseteq \mathcal{L}\}$. Let $\Phi = \{(u, \Lambda, u', \Lambda', R) \mid (u, \Lambda) \in \mathrm{ext}(\tilde{P}), (u', \Lambda') \in \tilde{P}, (u, \Lambda) \sim_{\mathcal{L}}^{\mathcal{E}(\Sigma), R} (u', \Lambda')\}$. Then, $\mathcal{L}$ is equal to the recognizable language defined by $\tilde{P}$, F, and Φ.

Proof of Lemma 33 First, we review *regions* and *region automata* [6]. *Region abstraction* is an important theoretical gadget in the theory of timed automaton: it reduces the *infinite* state space Q of a TTS $\mathcal{S}$ to its *finite* abstraction, the latter being amenable to algorithmic treatments, e. g., reachability analysis with a breadth-first search. Specifically, it relies on an equivalence relation $\approx_K$ over clock valuations. Given a TA $\mathcal{A} = (\Sigma, L, l_0, C, I, \Delta, F)$, for each $c \in C$, let K_c denote the greatest number compared with c in $\mathcal{A}$. Writing $\mathrm{int}(t)$ and $\mathrm{frac}(t)$ for the integer and fractional parts of $t \in \mathbb{R}_{\geq 0}$, let $\approx_K$ be the equivalence relation over clock valuations ν, ν' defined as follows. We have $\nu \approx_K \nu'$ if:

- for each $c \in C$ we have $\mathrm{int}(\nu(c)) = \mathrm{int}(\nu'(c))$ or $(\nu(c) > K_c$ and $\nu'(c) > K_c)$;
- for any $c, c' \in C$ satisfying $\nu(c) \leq K_c$ and $\nu(c') \leq K_{c'}$, $\mathrm{frac}(\nu(c)) < \mathrm{frac}(\nu(c'))$ if and only if $\mathrm{frac}(\nu'(c)) < \mathrm{frac}(\nu'(c'))$; and
- for any $c \in C$ such that $\nu(c) \leq K_c$, $\mathrm{frac}(\nu(c)) = 0$ if and only if $\mathrm{frac}(\nu'(c)) = 0$.

The equivalence relation $\approx_K$ is extended over the states Q of a TTS $\mathcal{S}$ such that $(l, \nu) \approx_K (l', \nu')$ if we have $l = l'$ and $\nu \approx_K \nu'$. By merging the equivalent states with respect to $\approx_K$, we abstract a TTS and obtain the *region automaton* [6].

Definition 35 (region automata (RAs)). *For a TA $\mathcal{A}$ and its TTS $\mathcal{S} = (Q, q_0, Q_F, \to)$, the* region automaton *(RA) is the NFA $\mathcal{A}^{\mathrm{r}} = (\Sigma, Q^{\mathrm{r}}, q_0^{\mathrm{r}}, Q_F^{\mathrm{r}}, E)$ defined as follows: $Q^{\mathrm{r}} = Q/{\approx_K}$; $q_0^{\mathrm{r}} \ni q_0$; $Q_F^{\mathrm{r}} = \{q^{\mathrm{r}} \in Q^{\mathrm{r}} \mid q^{\mathrm{r}} \subseteq Q_F\}$; $E = \{(q^{\mathrm{r}}, a, q'^{\mathrm{r}}) \mid \exists q \in q^{\mathrm{r}}, q' \in q'^{\mathrm{r}}.\, q \xrightarrow{a} q'\} \cup \{(q^{\mathrm{r}}, \varepsilon, q'^{\mathrm{r}}) \mid \exists q \in q^{\mathrm{r}}, q' \in q'^{\mathrm{r}}, \tau \in \mathbb{R}_{>0}.\, q \xrightarrow{\tau} q'$ or $q \xrightarrow{\varepsilon, \tau} q'\}$.*

For a timed word w and concrete states $q, q' \in Q$, we write $q \xrightarrow{w} q'$ if there is a run from q to q' associated with w.

The proof of Lemma 33 is as follows.

Lemma 33 (recalled). *For any recognizable timed language $\mathcal{L} \subseteq \mathcal{T}(\Sigma)$, the quotient set $\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}$ is finite.*

Proof (sketch). Let $\mathcal{A} = (\Sigma, L, l_0, C, I, \Delta, F)$ be a DTA recognizing $\mathcal{L}$, let $\mathcal{S} = (Q, q_0, Q_F, \to)$ be the TTS of $\mathcal{A}$, let $\mathcal{A}^{\mathrm{r}} = (\Sigma, Q^{\mathrm{r}}, q_0^{\mathrm{r}}, Q_F^{\mathrm{r}}, E)$ be the RA of $\mathcal{A}$, and for each $c \in C$, let K_c be the maximum constant compared with c in $\mathcal{A}$.

Without loss of generality, we can assume that $\mathcal{A}$ is complete, i. e., for any $q \in Q$ and $a \in \Sigma$, there is $q' \in Q$ satisfying $q \xrightarrow{a} q'$.

For a simple elementary language (u, Λ), we denote $q_0^{\mathrm{r}} \xrightarrow{(u,\Lambda)} q^{\mathrm{r}}$ for $q^{\mathrm{r}} \in Q^{\mathrm{r}}$ such that some $q \in q^{\mathrm{r}}$ is reachable by some $w \in (u, \Lambda)$. For any $q^{\mathrm{r}} \in Q^{\mathrm{r}}$ and for any simple elementary languages (u, Λ) and (u', Λ') satisfying $q_0^{\mathrm{r}} \xrightarrow{(u,\Lambda)} q^{\mathrm{r}}$ and $q_0^{\mathrm{r}} \xrightarrow{(u',\Lambda')} q^{\mathrm{r}}$, let R be the renaming equation that contains $\mathbb{T}_{i,|u|} = \mathbb{T}'_{j,|u'|}$ if there is a clock $c \in C$ whose value in q^{r} is equal to $\mathbb{T}_{i,|u|}$ and $\mathbb{T}'_{j,|u'|}$, and the value of c is smaller than or equal to K_c. For such a renaming equation R, we have $(u, \Lambda) \sim_{\mathcal{L}}^{\mathcal{E}(\Sigma),R} (u', \Lambda')$. Therefore, for any such (u, Λ) and (u', Λ'), $(u, \Lambda) \sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} (u', \Lambda')$ holds, and thus, for any $(u, \Lambda), (u', \Lambda')$, if they load to the same state of the RA, we have $(u, \Lambda) \sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} (u', \Lambda')$. Since the state space of the RA is finite, we have $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}| \leq |Q^{\mathrm{r}}|$.

A.3 Proof of Theorem 21

The proof of Theorem 21 is as follows.

Theorem 21 (recalled). *For any recognizable timed language $\mathcal{L}$, there is a finite set S of elementary languages satisfying $\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} = \sim_{\mathcal{L}}^{S}$.*

Proof. By the definition of $\sim_{\mathcal{L}}^{S}$, for any $S \subseteq S'$, $\sim_{\mathcal{L}}^{S'}$ is finer than $\sim_{\mathcal{L}}^{S}$, and we have $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S}}| \leq |\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S'}}|$. Let $S \subseteq \mathcal{E}(\Sigma)$ be such that $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S}}| < |\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}|$. Since $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S}}| < |\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}|$, there are $(u, \Lambda), (u', \Lambda') \in \mathcal{SE}(\Sigma)$ satisfying $(u, \Lambda) \sim_{\mathcal{L}}^{S} (u', \Lambda')$ and $(u, \Lambda) \not\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} (u', \Lambda')$. Let $\mathbb{T}$ and $\mathbb{T}'$ be the domain of Λ and Λ', respectively. By definition of $(u, \Lambda) \not\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} (u', \Lambda')$, for any renaming equation R over $\mathbb{T}$ and $\mathbb{T}'$ satisfying $\forall w \in (u, \Lambda). \exists w' \in (u', \Lambda'). \kappa(w), \kappa(w') \models R$, there is $s \in \mathcal{E}(\Sigma) \setminus S$ satisfying $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u, \Lambda) \cdot s) \wedge R \wedge \Lambda' \not\Leftrightarrow \mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u', \Lambda') \cdot s) \wedge R \wedge \Lambda$. Therefore, for each R, by adding such s to S, we obtain $S' \supsetneq S$ satisfying $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S}}| < |\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S'}}|$. Since there are only finitely many renaming equations R over $\mathbb{T}$ and $\mathbb{T}'$, such refinement increase S only finitely.

Let a sequence $S_0 \subsetneq S_1 \subsetneq \dots$ of elementary languages such that $S_0 = \emptyset$ and each S_i is obtained by the above refinement of S_{i-1}. By Lemma 33, $|\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}|$ is finite, and such a sequence is finite. Therefore, there is $n \in \mathbb{N}$ such that $\mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{S_n}} = \mathcal{SE}(\Sigma)/{\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)}}$. Moreover, by the construction of S_i, each S_i is finite, and thus, S_n is a finite set of elementary languages satisfying $\sim_{\mathcal{L}}^{\mathcal{E}(\Sigma)} = \sim_{\mathcal{L}}^{S_n}$.

A.4 Uniqueness of the fractional part

Here, we show the following property used in Section 4.1.

Proposition 36. *For any simple elementary language $(u, \Lambda) \in \mathcal{SE}(\Sigma)$ and for any timed words $w, w' \in (u, \Lambda)$ such that $w = \tau_0 a_1 \tau_1 a_2 \dots a_n \tau_n$ and $w' =$*

$\tau'_0 a_1 \tau'_1 a'_2 \dots a_n \tau'_n$, for any $i, j \in \{0, 1, \dots, n\}$, we have the following, where $\mathbb{T}_{i,n} = \sum_{k=i}^{n} \tau_k$ and $\mathbb{T}'_{i,n} = \sum_{k=i}^{n} \tau'_k$.

– $\mathrm{frac}(\mathbb{T}_{i,n}) = 0 \iff \mathrm{frac}(\mathbb{T}'_{i,n}) = 0$
– $\mathrm{frac}(\mathbb{T}_{i,n}) \leq \mathrm{frac}(\mathbb{T}_{j,n}) \iff \mathrm{frac}(\mathbb{T}'_{i,n}) \leq \mathrm{frac}(\mathbb{T}'_{j,n})$

Proof. If we have $\mathrm{frac}(\mathbb{T}_{i,n}) = 0$, since Λ is simple and canonical, Λ implies $\mathbb{T}_{i,n} = d$ for some $d \in \mathbb{N}$. Since $w' \in (u, \Lambda)$, we also have $\mathrm{frac}(\mathbb{T}'_{i,n}) = 0$.

Assume we have $\mathrm{frac}(\mathbb{T}_{i,n}) \leq \mathrm{frac}(\mathbb{T}_{j,n})$. If we have $\mathrm{frac}(\mathbb{T}_{i,n}) = 0$, by the first part of the property, we have $\mathrm{frac}(\mathbb{T}'_{i,n}) = 0$ and $\mathrm{frac}(\mathbb{T}'_{i,n}) \leq \mathrm{frac}(\mathbb{T}'_{j,n})$ holds. If we have $\mathrm{frac}(\mathbb{T}_{j,n}) = 0$, by $\mathrm{frac}(\mathbb{T}_{i,n}) \leq \mathrm{frac}(\mathbb{T}_{j,n})$, we have $\mathrm{frac}(\mathbb{T}_{i,n}) = 0$, and $\mathrm{frac}(\mathbb{T}'_{i,n}) \leq \mathrm{frac}(\mathbb{T}'_{j,n})$ holds. If we have $\mathrm{frac}(\mathbb{T}_{i,j}) = 0$, since Λ is simple and canonical, we also have $\mathrm{frac}(\mathbb{T}'_{i,j}) = 0$. Therefore, we have $\mathrm{frac}(\mathbb{T}_{i,n}) = \mathrm{frac}(\mathbb{T}_{j,n})$ and $\mathrm{frac}(\mathbb{T}'_{i,n}) = \mathrm{frac}(\mathbb{T}'_{j,n})$. If none of $\mathrm{frac}(\mathbb{T}_{i,n})$, $\mathrm{frac}(\mathbb{T}_{j,n})$, and, $\mathrm{frac}(\mathbb{T}_{i,j})$ is zero, let $a, b, c \in \mathbb{N}$ be such that $\mathbb{T}_{i,n} \in (a, a+1)$, $\mathbb{T}_{j,n} \in (b, b+1)$, and $\mathbb{T}_{i,j} \in (c, c+1)$. Clearly, we have either $c = a - b$ or $c = a - b - 1$. If we have $c = a - b$, we have $\mathrm{frac}(\mathbb{T}_{i,n}) < \mathrm{frac}(\mathbb{T}_{j,n})$ and $\mathrm{frac}(\mathbb{T}'_{i,n}) < \mathrm{frac}(\mathbb{T}'_{j,n})$. Otherwise, we have $\mathrm{frac}(\mathbb{T}_{i,n}) > \mathrm{frac}(\mathbb{T}_{j,n})$ and $\mathrm{frac}(\mathbb{T}'_{i,n}) > \mathrm{frac}(\mathbb{T}'_{j,n})$. Overall, we have $\mathrm{frac}(\mathbb{T}_{i,n}) \leq \mathrm{frac}(\mathbb{T}_{j,n}) \iff \mathrm{frac}(\mathbb{T}'_{i,n}) \leq \mathrm{frac}(\mathbb{T}'_{j,n})$. ◻

A.5 Proof of Theorem 27

First, we show the following lemma on successors.

Lemma 37. *Let (P, S, T) be a closed, consistent, and exterior-consistent timed observation table. For any $p \in \mathrm{succ}(P)$, there is $p' \in P$ such that $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$ and for any $w \in p$, there is $w' \in p'$ satisfying $(w, w') \in [\![\Phi]\!]$, where Φ is the chronometric relational morphism constructed in Algorithm 2.*

Proof. If $p \in P$, for any $w \in p$, we have $(w, w) \in [\![\Phi]\!]$. Assume $p \notin P$. Let $\tilde{p} \in P$ satisfying $p \in \mathrm{succ}(\tilde{p})$. Since the timed observation table is exterior-consistent, there is $(u^{\mathrm{ext}}, \Lambda^{\mathrm{ext}}) \in \mathrm{ext}(\tilde{p})$ satisfying $p \subseteq (u^{\mathrm{ext}}, \Lambda^{\mathrm{ext}})$. By closedness of the timed observation table and by the construction of Φ, there are $p' = (u', \Lambda') \in P$ and a renaming equation R satisfying $p \sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} p'$, and $(u^{\mathrm{ext}}, \Lambda^{\mathrm{ext}}, u', \Lambda', R) \in \Phi$. By $p \sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} p'$, for any $w \in p$, there is $w' \in (u', \Lambda')$ satisfying $\kappa(w), \kappa(w') \models R$. Therefore, we have $(w, w') \in [\![\Phi]\!]$.

The proof of Theorem 27 is as follows.

Theorem 27 (recalled). *For any cohesive timed observation table (P, S, T), for any $p \in P \cup \mathrm{succ}(P)$, $\mathcal{L}_{\mathrm{tgt}} \cap p = \mathcal{L}(\mathtt{MakeDTA}(P, S, T)) \cap p$ holds.*

Proof. Let (P, F, Φ) be the recognizable timed language we construct in Algorithm 2. Since p is a simple elementary language, for any recognizable timed language $\mathcal{L}$, we have either $p \subseteq \mathcal{L}$ or $p \cap \mathcal{L} = \emptyset$.

If $p \in P$, by the construction of F in Algorithm 2, $p \subseteq F$ holds if and only if we have $p \subseteq \mathcal{L}_{\mathrm{tgt}}$. By definition of recognizable timed languages, we have $p \subseteq \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ if and only if $p \subseteq \mathcal{L}_{\mathrm{tgt}}$. Therefore, we have $\mathcal{L}_{\mathrm{tgt}} \cap p = \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \cap p$.

If $p \in \mathrm{succ}(P) \setminus P$, we let $p' \in P$ be such that $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$ and for any $w \in p$, there is $w' \in p'$ satisfying $(w, w') \in [\![\Phi]\!]$. The existence of such p' is guaranteed by Lemma 37. By the construction of $\mathcal{A}_{\mathrm{hyp}}$, for such $w \in p$ and $w' \in p'$, we have $w \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \iff w' \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. Since $p \sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} p'$, we have $p \cap \mathcal{L}_{\mathrm{tgt}} = \emptyset \iff p' \cap \mathcal{L}_{\mathrm{tgt}} = \emptyset$. Since p and p' are simple elementary languages, we have $\mathcal{L}_{\mathrm{tgt}} \cap p = \mathcal{L}(\mathcal{A}_{\mathrm{hyp}}) \cap p$.

A.6 Proof of Theorem 28

The proof of Theorem 28 is as follows.

Theorem 28 (recalled). *For any cohesive timed observation table (P, S, T), $\sim^{S}_{\mathcal{L}_{\mathrm{tgt}}} = \sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$ implies $\mathcal{L}_{\mathrm{tgt}} = \mathcal{L}(\texttt{MakeDTA}(P, S, T))$.*

Proof. Let $w \in \mathcal{T}(\Sigma)$ be a timed word. Let $\mathcal{A}_{\mathrm{hyp}} = \texttt{MakeDTA}(P,\ S,\ T)$. Let $w_0, w_1, \dots, w_n, w'_1, w'_2, \dots, w'_n, w''_1, w''_2, \dots, w''_n, \overline{w}_1, \overline{w}_2, \dots, \overline{w}_n$, be timed words, let $p_1, p_2, \dots, p_n \in P$, and $p'_1, p'_2, \dots, p'_n \in \mathrm{succ}(P)$ be simple elementary languages, and let $R_1, R_2, \dots, R_n$ be renaming equations satisfying the following properties. We note that the above exists for some $n > 0$ because the timed observation table is closed.

- $w_0 = w$
- $w''_n = \varepsilon$
- For each $i \in \{1, 2, \dots, n\}$, we have $w'_i \in p'_i$, $\overline{w}_i \in p_i$, $w_{i-1} = w'_i \cdot w''_i$, $w_i = \overline{w}_i \cdot w''_i$, $p'_i \sim^{S,R_i}_{\mathcal{L}_{\mathrm{tgt}}} p_i$, and $\kappa(w'_i), \kappa(\overline{w}_i) \models R_i$.

By the construction of the hypothesis DTA $\mathcal{A}_{\mathrm{hyp}}$, we have $w \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ if and only if $w_n \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. By Lemma 32, for each $i \in \{1, 2, \dots, n\}$, we have $w_{i-1} \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ if and only if $w_{i-1} \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. By Theorem 27, we have $w_n \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$ if and only if $w_n \in \mathcal{L}_{\mathrm{tgt}}$. Overall, we have $w \in \mathcal{L}_{\mathrm{tgt}}$ if and only if $w \in \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$. Therefore, we have $\mathcal{L}_{\mathrm{tgt}} = \mathcal{L}(\mathcal{A}_{\mathrm{hyp}})$.

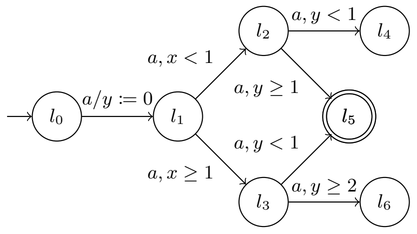
B Detail of the algorithm

B.1 Detail of symbolic membership

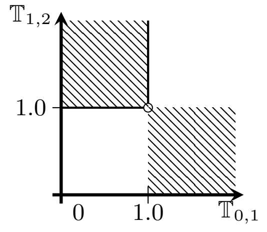
We show the detail of our algorithm to obtain the symbolic membership using finite membership queries. Algorithm 5 shows an outline of our construction. In Algorithm 5, we enumerate all the simple and canonical timed conditions $\Lambda' \subseteq \Lambda$. Such enumeration can be done, e. g., by a depth-first-search: each depth corresponds to (i, j) satisfying $0 \leq i \leq j \leq |u|$ and at each branch, we pick $\mathbb{T}_{i,j} \in (k, k+1)$ or $\mathbb{T}_{i,j} = k$ for some k. Such an exhaustive trial (instead of, e. g., a binary search) is necessary because $\texttt{mem}^{\texttt{sym}}_{\mathcal{L}}((u, \Lambda))$ is potentially non-convex.

Algorithm 5: Outline of the construction of symbolic membership

```
Input  : An elementary language $(u, \Lambda)$ and the function $\texttt{mem}_{\mathcal{L}}$ mapping a
         timed word $w$ to its membership to a timed language $\mathcal{L}$
Output : The symbolic membership query $g$ of $(u, \Lambda)$ in $\mathcal{L}$
1 Function $\texttt{mem}^{\texttt{sym}}_{\mathcal{L}}((u, \Lambda))$:
2     $g \leftarrow \bot$
3     foreach simple and canonical $\Lambda' \subseteq \Lambda$ do
          // The result is independent of the choice of $w$
4         pick $w \in (u, \Lambda')$
5         if $\texttt{mem}_{\mathcal{L}}(w) = \top$ then
6             $g \leftarrow g \vee \Lambda'$
7     return $g$
```



(a) DTA recognizing $\mathcal{L}$



(b) $\texttt{mem}^{\texttt{sym}}_{\mathcal{L}}((u, \Lambda))$ projected to $\mathbb{T}_{0,1}$ and $\mathbb{T}_{1,2}$

Fig. 4: Example of non-convex symbolic membership $\texttt{mem}^{\texttt{sym}}_{\mathcal{L}}((u, \Lambda))$, where $u = aaa$ and $\Lambda = \bigwedge_{0<i\leq j\leq 2,(i,j)\notin\{(0,1),(1,2)\}} \mathbb{T}_{i,j} \in (0,1) \wedge \mathbb{T}_{0,1} \in (0,2) \wedge \mathbb{T}_{1,2} \in (0,2)$

Example 38. Let $u = aaa$ and $\Lambda = \bigwedge_{0<i\leq j\leq 2,(i,j)\notin\{(0,1),(1,2)\}} \mathbb{T}_{i,j} \in (0,1) \wedge \mathbb{T}_{0,1} \in (0,2) \wedge \mathbb{T}_{1,2} \in (0,2)$. Let $\mathcal{L}$ be the timed language recognized by the DTA in Fig. 4a. The accepting location l_5 is reachable by following $l_0 \to l_1 \to l_2 \to l_5$ or $l_0 \to l_1 \to l_3 \to l_5$, and the union of the corresponding timing constraints along these paths are non-convex. Therefore, the symbolic membership is non-convex, as shown in Fig. 4.

The number of membership queries used in Algorithm 5 is the same as the number of the simple and canonical timed conditions $\Lambda' \subseteq \Lambda$, which exponentially blows up to the number of variables in Λ. In our DTA learning algorithm, Algorithm 5 is used only for elementary languages of the form $(u, \Lambda) \cdot (u', \Lambda')$, where (u, Λ) and (u', Λ') are simple. The following example shows that even with such restriction, the number of the necessary membership queries may blow up exponentially.

Example 39. Let Λ, and Λ' be $\Lambda = \bigwedge_{0\leq i\leq j\leq n} \mathbb{T}_{i,j} \in (0,1)$ and $\Lambda' = \bigwedge_{0\leq i\leq j\leq n'} \mathbb{T}'_{i,j} \in (0,1)$. Let $\Lambda\cdot\Lambda'$ be the timed condition such that $\Lambda\cdot\Lambda' = \{\nu\cdot\nu' \mid \nu \in \Lambda, \nu' \in \Lambda'\}$. For any simple and canonical timed condition $\Lambda'' \subseteq \Lambda\cdot\Lambda'$ and for each $0 \leq i \leq j \leq n+n'-1$, we have $\mathbb{T}''_{i,j} \in (0,1)$, $\mathbb{T}''_{i,j} = 1$, or $\mathbb{T}''_{i,j} \in (1,2)$. Moreover, if we have $\mathbb{T}''_{i,j} = 1$ or $\mathbb{T}''_{i,j} \in (1,2)$, for any i' and j' satisfying $0 \leq i' \leq i$, $j \leq j' \leq n+n'-1$, and $(i,j) \neq (i',j')$, we have $\mathbb{T}''_{i',j'} \in (1,2)$. Therefore, we can underapproximate the number of the simple and canonical timed conditions $\Lambda'' \subseteq \Lambda\cdot\Lambda'$ by counting the possible combination of such boundaries, i. e., the pair (i,j) satisfying $\mathbb{T}''_{i,j} = 1$ or $\mathbb{T}''_{i,j} \in (1,2)$. If there are no such boundaries, we have $\mathbb{T}_{i,j} \in (0,1)$ for any pair (i,j), and the number of the corresponding Λ'' is one. Let $m > 0$ be the number of the boundaries. For indices i,j corresponding to one of such boundaries satisfying $0 \leq i \leq n \leq j \leq n+n'-1$, for any indices i',j' corresponding to any other boundaries, we have $0 \leq i < i' \leq n \leq j < j' \leq n+n'-1$ or $0 \leq i' < i \leq n \leq j' < j \leq n+n'-1$. Therefore, the correspondence between i and j (and i' and j') is uniquely determined by order of the indices (i. e., $i < i'$ or $i' < i$), and it suffices to count the combination of the indices independently. Overall, the number of the simple and canonical timed conditions $\Lambda'' \subseteq \Lambda\cdot\Lambda'$ is greater than the following, and we indeed have an exponential blowup, where $\binom{n}{m}$ is the binomial coefficient.

$$1 + \sum_{m=0}^{\min\{n,n'\}} \binom{n}{m} \times \binom{n'}{m} > \sum_{m=0}^{\min\{n,n'\}} \binom{\min\{n,n'\}}{m} = 2^{\min\{n,n'\}}$$

B.2 Checking if $(u,\Lambda) \sim^{S}_{\mathcal{L}} (u',\Lambda')$

First, we show how to check if $(u,\Lambda) \sim_{\mathcal{L}}^{(u'',\Lambda''),R} (u',\Lambda')$ for a timed language $\mathcal{L}$, simple elementary languages (u,Λ), (u',Λ'), an elementary language (u'',Λ''), and a renaming equation R. We have $(u,\Lambda) \sim_{\mathcal{L}}^{(u'',\Lambda''),R} (u',\Lambda')$ if and only if the following three conditions hold.

1. For any $w \in (u,\Lambda)$, there is $w' \in (u',\Lambda')$ satisfying $\kappa(w), \kappa(w') \models R$.

Algorithm 6: Outline of the checking of $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$

```
Input  : A timed language $\mathcal{L}$, elementary languages $(u, \Lambda), (u', \Lambda')$, a set of
         elementary languages $S$, and a renaming equation $R$
Output : If $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$ or not
1 Function CheckRowEquivalenceSingle:
2     if $\Lambda \wedge \Lambda' \wedge R$ is unsatisfiable then
3         return $\bot$
4     for $s \in S$ do
5         if $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u, \Lambda) \cdot s) \wedge R \wedge \Lambda' \neq \mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u', \Lambda') \cdot s) \wedge R \wedge \Lambda$ then
6             return $\bot$
7     return $\top$
```

2. For any $w' \in (u', \Lambda')$, there is $w \in (u, \Lambda)$ satisfying $\kappa(w), \kappa(w') \models R$.
3. $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge R \wedge \Lambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathtt{sym}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge R \wedge \Lambda$ holds.

Without loss of generality, we assume that Λ, Λ', and Λ'' are simple and canonical. Since R is a finite conjunction of equations of the form $\mathbb{T}_{i,|u|} = \mathbb{T}'_{j,|u'|}$, Item 1 holds if and only if for any $w \in (u, \Lambda)$, there is $w' \in (u', \Lambda')$ such that for any $\mathbb{T}_{i,|u|} = \mathbb{T}'_{j,|u'|}$ in R, $\kappa(w)(\mathbb{T}_{i,|u|}) = \kappa(w')(\mathbb{T}'_{j,|u'|})$. This holds if and only if $\Lambda \Rightarrow (R \wedge \Lambda'){\downarrow_{\mathbb{T}}}$ is a tautology, where $(R \wedge \Lambda'){\downarrow_{\mathbb{T}}}$ is the constraint $R \wedge \Lambda'$ (over $\mathbb{T} \cup \mathbb{T}'$) restricted to $\mathbb{T}$. If $\Lambda \Rightarrow (R \wedge \Lambda'){\downarrow_{\mathbb{T}}}$ is a tautology, we have $\Lambda \wedge (R \wedge \Lambda'){\downarrow_{\mathbb{T}}} = \Lambda$. Since Λ and $(R \wedge \Lambda'){\downarrow_{\mathbb{T}}}$ are simple and canonical, $\Lambda \wedge (R \wedge \Lambda'){\downarrow_{\mathbb{T}}} = \Lambda$ holds if and only if $\Lambda \wedge (R \wedge \Lambda'){\downarrow_{\mathbb{T}}}$ is satisfiable. Moreover, this is equivalent to the satisfiability of $\Lambda \wedge \Lambda' \wedge R$. Therefore, Item 1 can be checked by examining if $\Lambda \wedge \Lambda' \wedge R$ is satisfiable. Similarly, Item 2 can be checked by examining if $\Lambda \wedge \Lambda' \wedge R$ is satisfiable.

One can check if $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$ by checking $(u, \Lambda) \sim_{\mathcal{L}}^{(u'', \Lambda''),R} (u', \Lambda')$ for each $(u'', \Lambda'') \in S$. Algorithm 6 shows an outline of it.

To check if $(u, \Lambda) \sim_{\mathcal{L}}^{S} (u', \Lambda')$, we generate candidate renaming equations R and check if $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$. Each R can be seen as a bipartite graph (V_1, V_2, E) such that $V_1 = \{\mathbb{T}_{i,n} \mid 0 \leq i \leq n\}$, $V_2 = \{\mathbb{T}'_{i',n'} \mid 0 \leq i' \leq n'\}$, and $E = \{(\mathbb{T}_{i,n}, \mathbb{T}_{i',n'}) \mid R \text{ contains } \mathbb{T}_{i,n} = \mathbb{T}_{i',n'}\}$. Therefore, constructing a candidate renaming equation R is equivalent to constructing a set of edges E.

Before showing our construction, we prove the following properties.

Proposition 40. *For any (u, Λ), (u', Λ'), S, $\mathcal{L}$, and R satisfying $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$, if Λ and Λ' are simple and canonical, for each equation $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ in R, the range of $\mathbb{T}_{i,|u|}$ in Λ and the range of $\mathbb{T}_{i',|u'|}$ in Λ' are the same.*

Proof. For $i \in \{0, 1, \dots, |u|\}$ (resp. $i' \in \{0, 1, \dots, |u'|\}$), let I_i (resp. $I'_{i'}$) be such that Λ (resp. Λ') contains $\mathbb{T}_{i,|u|} = I_i$ (resp. $\mathbb{T}'_{i',|u'|} = I'_{i'}$). Since Λ and Λ' are simple and canonical, $I_i \neq I'_{i'}$ implies $I_i \cap I'_{i'} = \emptyset$. Since we have $(u, \Lambda) \sim_{\mathcal{L}}^{S,R} (u', \Lambda')$, $\Lambda \wedge \Lambda' \wedge R$ must be satisfiable. Therefore, for each equation $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ in R, we have $I_i = I'_{i'}$.

Definition 41. *Let $\mathcal{L}$ be a timed language and (u, Λ) and (u', Λ') be elementary languages over $\mathbb{T}$ and $\mathbb{T}'$, respectively. For $i \in \{0, 1, \dots, n\}$ and $i' \in \{0, 1, \dots, n'\}$, we call $\mathbb{T}_{i,n} + \mathbb{T}'_{0,i'}$ is* non-trivial *in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u', \Lambda'))$ if the range of $\mathbb{T}_{i,n} + \mathbb{T}'_{0,i'}$ is different between $\Lambda \wedge \Lambda'$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u', \Lambda'))$.*

Proposition 42. *For any (u, Λ), (u', Λ'), S, $\mathcal{L}$, and R satisfying $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$, if Λ and Λ' are simple and canonical, for each equation $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ in R, for each $(u'', \Lambda'') \in S$, and for each $i'' \in \{0, 1, \dots, |u''|\}$, $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is non-trivial in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$ if and only if $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is non-trivial in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$.*

Proof. Assume $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is non-trivial in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$, i. e., the range of $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$. Since $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$ implies $\Lambda \wedge \Lambda''$, the range of $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ remains different by strengthening $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$. Therefore, the range of $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge \Lambda' \wedge R$. By $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$, we have $\Lambda = \Lambda \wedge (\Lambda' \wedge R){\downarrow_{\mathbb{T}}}$. Therefore, the range of $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda \wedge (\Lambda' \wedge R){\downarrow_{\mathbb{T}}} \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge \Lambda' \wedge R$. Since $\Lambda' \wedge R$ is a constraint over $\mathbb{T} \cup \mathbb{T}'$, the range of $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda \wedge \Lambda' \wedge \Lambda'' \wedge R$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge \Lambda' \wedge R$. Since R contains $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is also different between $\Lambda \wedge \Lambda' \wedge \Lambda'' \wedge R$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge \Lambda' \wedge R$. Since we have $(u, \Lambda) \sim^{(u'', \Lambda''), R}_{\mathcal{L}} (u', \Lambda')$, $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda'')) \wedge R \wedge \Lambda'$ holds if and only if $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge R \wedge \Lambda$ holds. Therefore, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda \wedge \Lambda' \wedge \Lambda'' \wedge R$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda \wedge R$. Since $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda \wedge R$ implies $\Lambda' \wedge \Lambda'' \wedge \Lambda \wedge R$, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ remains different by relaxing $\Lambda' \wedge \Lambda'' \wedge \Lambda \wedge R$. Therefore, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda' \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda \wedge R$. Since $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$ implies Λ', the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda' \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda \wedge \Lambda' \wedge R$. Since $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ does not contain any variables in $\mathbb{T}$, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda' \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda' \wedge (\Lambda \wedge R){\downarrow_{\mathbb{T}'}}$. By $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$, we have $\Lambda' = \Lambda' \wedge (\Lambda \wedge R){\downarrow_{\mathbb{T}'}}$. Therefore, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda' \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda'$. Since $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$ implies Λ', we have $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) = \mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda'')) \wedge \Lambda'$, and thus, the range of $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$ is different between $\Lambda' \wedge \Lambda''$ and $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$. Therefore, $\mathbb{T}_{i',|u'|} + \mathbb{T}''_{0,i''}$ is non-trivial in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u', \Lambda') \cdot (u'', \Lambda''))$. The proof of the opposite direction is similar.

Proposition 43. *For any (u, Λ), (u', Λ'), S, and $\mathcal{L}$ satisfying $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$, if Λ and Λ' are simple and canonical, there is a renaming equation R such that for any equation $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ in R, there are $(u'', \Lambda'') \in S$ and $i'' \in \{0, 1, \dots, |u''|\}$ such that $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is non-trivial in $\mathtt{mem}^{\mathsf{sym}}_{\mathcal{L}}((u, \Lambda) \cdot (u'', \Lambda''))$.*

Proof. Let $R' = R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|})$ be a renaming equation satisfying $(u, \varLambda) \sim_{\mathcal{L}}^{S,R'} (u', \varLambda')$, and for any $(u'', \varLambda'') \in S$ and for any $i'' \in \{0, 1, \dots, |u''|\}$, $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is trivial in $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda''))$. We show that $(u, \varLambda) \sim_{\mathcal{L}}^{S,R} (u', \varLambda')$ also holds. By definition of $(u, \varLambda) \sim_{\mathcal{L}}^{S,R'} (u', \varLambda')$, we have the following three conditions.

- For any $w \in (u, \varLambda)$, there is $w' \in (u', \varLambda')$ satisfying $\kappa(w), \kappa(w') \models R'$.
- For any $w' \in (u', \varLambda')$, there is $w \in (u, \varLambda)$ satisfying $\kappa(w), \kappa(w') \models R'$.
- For any $(u'', \varLambda'') \in S$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R' \land \varLambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda'')) \land R' \land \varLambda$ holds.

Since R' implies R, we immediately have the following two conditions.

- For any $w \in (u, \varLambda)$, there is $w' \in (u', \varLambda')$ satisfying $\kappa(w), \kappa(w') \models R$.
- For any $w' \in (u', \varLambda')$, there is $w \in (u, \varLambda)$ satisfying $\kappa(w), \kappa(w') \models R$.

By definition of R', for any $(u'', \varLambda'') \in S$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}) \land \varLambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda'')) \land R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}) \land \varLambda$ holds. Since $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda''))$ implies $\varLambda \land \varLambda''$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}) \land \varLambda'$ is equivalent to $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}) \land \varLambda \land \varLambda' \land \varLambda''$. Since for any $i'' \in \{0, 1, \dots, |u''|\}$, $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ is trivial in $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda''))$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land (\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}) \land \varLambda \land \varLambda' \land \varLambda''$ is equivalent to $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land \varLambda \land \varLambda' \land \varLambda''$, which is also equivalent to $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land \varLambda'$. Therefore, for any $(u'', \varLambda'') \in S$ $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R' \land \varLambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land \varLambda'$ holds. By a similar discussion, for any $(u'', \varLambda'') \in S$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda'')) \land R' \land \varLambda$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda'')) \land R \land \varLambda$ holds. Overall, for any $(u'', \varLambda'') \in S$, $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda'')) \land R \land \varLambda'$ holds if and only if $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda'')) \land R \land \varLambda$ holds, and thus, we have $(u, \varLambda) \sim_{\mathcal{L}}^{S,R} (u', \varLambda')$.

Algorithm 7 outlines our construction of the candidate renaming equations. In line 1, we construct the bipartite graph $(\tilde{V_1}, \tilde{V_2}, \tilde{E})$ such that:

- $\tilde{V_1} \subseteq \mathbb{T}$ and $\tilde{V_2} \subseteq \mathbb{T}'$ are such that $\tau_i \in \tilde{V_1}$ (resp. $\tau'_{i'} \in \tilde{V_2}$) if and only if there are $(u'', \varLambda'') \in S$ and $i'' \in \{0, 1, \dots, |u''|\}$ such that $\mathbb{T}_{i,|u|} + \mathbb{T}''_{0,i''}$ (resp. $\mathbb{T}'_{i',|u'|} + \mathbb{T}''_{0,i''}$) is non-trivial in $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u, \varLambda) \cdot (u'', \varLambda''))$ (resp. $\mathtt{mem}_{\mathcal{L}}^{\mathsf{sym}}((u', \varLambda') \cdot (u'', \varLambda''))$);
- $\tilde{E}$ is such that $(\tau_i, \tau'_{i'}) \in \tilde{E}$ if and only if the range of $\mathbb{T}_{i,|u|}$ in $\varLambda$ and the range of $\mathbb{T}'_{i',|u'|}$ in $\varLambda'$ are the same.

We encode a renaming equation R by a set $\tilde{E}_R \subseteq \tilde{E}$ of edges such that R contains $\mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ if and only if $(\tau_i, \tau'_{i'}) \in \tilde{E}_R$. By Proposition 43, each candidate renaming equation R does not have to contain the variables not in $\tilde{V_1} \cup \tilde{V_2}$. The edges $\tilde{E}$ consists of the candidate equations satisfying the condition in Proposition 40. Therefore, we can check if $(u, \varLambda) \sim_{\mathcal{L}}^{S} (u', \varLambda')$ by enumerating $\tilde{E}_R \subseteq \tilde{E}$ and checking if we have $(u, \varLambda) \sim_{\mathcal{L}}^{S,R} (u', \varLambda')$. We note that each disjoint subgraph of this bipartite graph is complete.

Algorithm 7: Outline of an algorithm to check if $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$

```
Input  : A timed language $\mathcal{L}$, elementary languages $(u, \Lambda), (u', \Lambda')$, and a set
         of elementary languages $S$
Output : If $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$ or not
1  construct $(\tilde{V_1}, \tilde{V_2}, \tilde{E})$
2  $\overline{R} \leftarrow \{\top\}$
   // Generate candidate renaming equations
3  foreach disjoint subgraph $(\tilde{V_1'}, \tilde{V_2'}, \tilde{E'})$ of $(\tilde{V_1}, \tilde{V_2}, \tilde{E})$ do
       // Pick one edge for each disjoint subgraph
4      $\overline{R} \leftarrow \{R \land \mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|} \mid R \in \overline{R}, (\tau_i, \tau'_{i'}) \in \tilde{E'}\}$
5  while $\forall R \in \overline{R}. \neg$CheckRowEquivalenceSingle($\mathcal{L}, (u, \Lambda), (u', \Lambda'), S, R$) do
6      $\overline{R_{\mathrm{new}}} \leftarrow \emptyset$
       // Try to add another equation
7      foreach disjoint subgraph $(\tilde{V_1'}, \tilde{V_2'}, \tilde{E'})$ of $(\tilde{V_1}, \tilde{V_2}, \tilde{E})$ do
8          foreach $R \in \overline{R}$ and $(\tau_i, \tau'_{i'}) \in \tilde{E'}$ do
9              if $\Lambda \land \Lambda' \land R \land \mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ is satisfiable then
10                 add $R \land \mathbb{T}_{i,|u|} = \mathbb{T}'_{i',|u'|}$ to $\overline{R_{\mathrm{new}}}$
11     if $\overline{R_{\mathrm{new}}} = \overline{R} \lor \overline{R_{\mathrm{new}}} = \emptyset$ then
12         return $\bot$
13     $\overline{R} \leftarrow \overline{R_{\mathrm{new}}}$
14 return $\top$
```

Since each variable in $\tilde{V_1} \cup \tilde{V_2}$ must be constrained by R, R has to contain at least one equation for each disjoint subgraph of the bipartite $(\tilde{V_1}, \tilde{V_2}, \tilde{E})$. For each candidate renaming equation R, we check if $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$. If $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$ is satisfied by some R, we conclude $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$.

We remark that, overall, the checking of $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$ only uses the information in the timed observation table, and does not affect the overall complexity of the learning algorithm in terms of the number of queries.

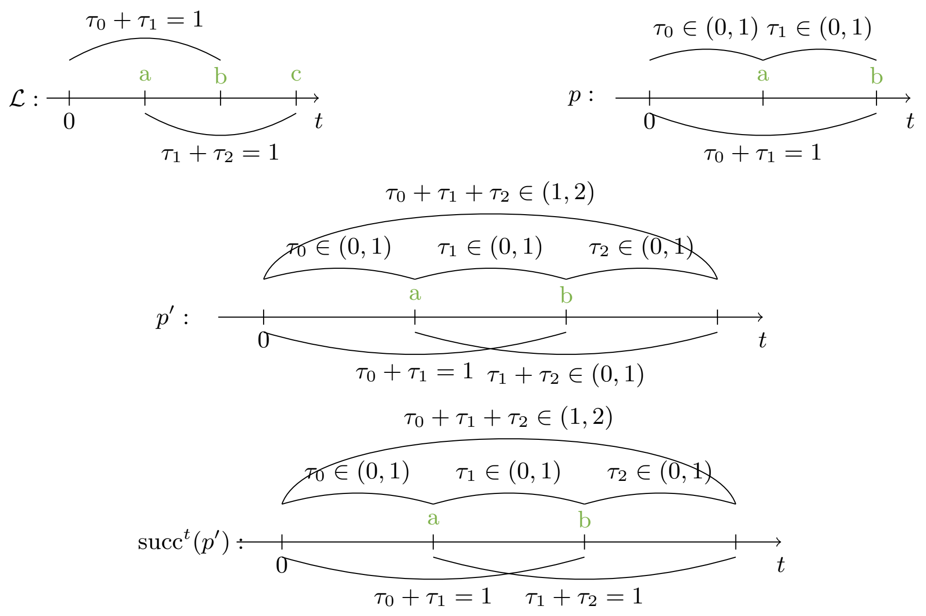
C On the consistency in L*-style algorithms

The following example shows that we cannot refine S by a "continuous consistency" defined by the continuous successors.

Example 44. Let $\mathcal{L} = \{\tau_0 a \tau_1 b \tau_2 c \tau_3 \mid \tau_0 + \tau_1 = \tau_1 + \tau_2 = 1\}$ and let $p = (u, \Lambda)$ and $p' = (u', \Lambda')$ be such that $u = u' = ab$, $\Lambda = \{\tau_0 \in (0, 1) \land \tau_1 \in (0, 1) \land \tau_2 = 0 \land \tau_0 + \tau_1 = 1 \land \tau_1 + \tau_2 \in (0, 1) \land \tau_0 + \tau_1 + \tau_2 = 1\}$, $\Lambda' = \{\tau_0 \in (0, 1) \land \tau_1 \in (0, 1) \land \tau_2 \in (0, 1) \land \tau_0 + \tau_1 = 1 \land \tau_1 + \tau_2 \in (0, 1) \land \tau_0 + \tau_1 + \tau_2 \in (1, 2)\}$. See Fig. 5 for an illustration. The orders on the fractional parts are as follows.

- Θ_p: $0 = \mathrm{frac}(\mathbb{T}_{2,2}) < \mathrm{frac}(\mathbb{T}_{0,2}) < \mathrm{frac}(\mathbb{T}_{1,2})$
- $\Theta_{p'}$: $0 < \mathrm{frac}(\mathbb{T}_{2,2}) < \mathrm{frac}(\mathbb{T}_{0,2}) < \mathrm{frac}(\mathbb{T}_{1,2})$

Let $s = (u_s, \Lambda_s)$ be an elementary language. $(p \cdot s) \cap \mathcal{L}$ is nonempty if and only if we have $u_s = c$ and the range of τ^s_0 in Λ_s contains $(0, 1)$. $(p' \cdot s) \cap \mathcal{L}$ is also



$\tau_0 + \tau_1 = 1$
a
b
c
$\mathcal{L}$:
0
t
$\tau_1 + \tau_2 = 1$
$\tau_0 \in (0,1)$ $\tau_1 \in (0,1)$
a
b
p :
0
t
$\tau_0 + \tau_1 = 1$
$\tau_0 + \tau_1 + \tau_2 \in (1,2)$
$\tau_0 \in (0,1)$ $\tau_1 \in (0,1)$ $\tau_2 \in (0,1)$
a
b
p' :
0
t
$\tau_0 + \tau_1 = 1$ $\tau_1 + \tau_2 \in (0,1)$
$\tau_0 + \tau_1 + \tau_2 \in (1,2)$
$\tau_0 \in (0,1)$ $\tau_1 \in (0,1)$ $\tau_2 \in (0,1)$
a
b
$\mathrm{succ}^t(p')$:
0
t
$\tau_0 + \tau_1 = 1$ $\tau_1 + \tau_2 = 1$

Fig. 5: Illustration of $\mathcal{L}$, p, p', and $\mathrm{succ}^t(p')$ in Example 44

nonempty if and only if we have the same condition. Moreover, for any such s, we have $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}(p \cdot s) = \Lambda \wedge \Lambda_s \wedge (\tau_1 + \tau^s_0 = 1)$ and $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}}(p' \cdot s) = \Lambda' \wedge \Lambda_s \wedge (\tau'_1 + \tau'_2 + \tau^s_0 = 1)$. Therefore, we have $p \sim^{\mathcal{E}(\Sigma),R}_{\mathcal{L}} p'$, with $R = \{\tau_1 = \tau'_1 + \tau'_2\}$.

In contrast, for $\mathrm{succ}^t(p) = p'$ and $\mathrm{succ}^t(p')$, for $s' = (u_{s'}, \Lambda_{s'})$ such that $u_{s'} = c$ and $\Lambda_{s'} = \{\tau^{s'}_0 = \tau^{s'}_1 = 0\}$, we have $\mathrm{succ}^t(p) \cdot s' \cap \mathcal{L} = \emptyset$ and $\mathrm{succ}^t(p') \cdot s' \cap \mathcal{L} \neq \emptyset$. Therefore, there is no renaming equation R satisfying $\mathrm{succ}^t(p) \not\sim^{\mathcal{E}(\Sigma),R}_{\mathcal{L}} \mathrm{succ}^t(p')$. Overall, in general, the "continuous consistency" does not hold even for $S = \mathcal{E}(\Sigma)$, and thus, we cannot refine S by enforcing "consistency" for continuous successors.

In contrast, any timed observation table becomes *discretely* consistent in the limit.

Proposition 45. *Let $O = (P, S, T)$ be a timed observation table. If $\sim^S_{\mathcal{L}_{\mathrm{tgt}}}$ is equal to $\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}}$, O is consistent.*

Proof. We prove the contraposition, i. e., $\mathrm{succ}^a(p) \not\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$ implies $p \not\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}} p'$. When $\mathrm{succ}^a(p) \not\sim^{\mathcal{E}(\Sigma)}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$ holds, since $\mathcal{L}_{\mathrm{tgt}}$ is a recognizable timed language, by Theorem 21, there is a finite set $S \subseteq \mathcal{E}(\Sigma)$ satisfying $\mathrm{succ}^a(p) \not\sim^S_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$. Moreover, by definition of $\sim^S_{\mathcal{L}_{\mathrm{tgt}}}$, for any renaming equation R, we have $\mathrm{succ}^a(p) \not\sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$. For such S and R, let $S' = \{a \cdot s \mid s \in S\}$ and let R' be the renaming equation such that $\tau_{|u|+1}$ and $\tau'_{|u'|+1}$ in R are replaced with $\tau_{|u|}$ and $\tau'_{|u'|}$, respectively, where $a \cdot s = \{a \cdot w \mid w \in s\}$, and u and u' are such that $p = (u, \Lambda)$ and $p' = (u', \Lambda')$.

Let $(u_a, \Lambda_a) = \mathrm{succ}^a(p)$ and $(u'_a, \Lambda'_a) = \mathrm{succ}^a(p')$. When we have $\mathrm{succ}^a(p) \not\sim^{S,R}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^a(p')$ at least one of the following holds.

1. There is $w_a \in (u_a, \Lambda_a)$ such that for any $w'_a \in (u'_a, \Lambda'_a)$, we have $\kappa(w_a), \kappa(w'_a) \not\models R$.
2. There is $w'_a \in (u'_a, \Lambda'_a)$ such that for any $w_a \in (u_a, \Lambda_a)$, we have $\kappa(w_a), \kappa(w'_a) \not\models R$.
3. There is $(u'', \Lambda'') \in S$ such that $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u_a, \Lambda_a) \cdot (u'', \Lambda'')) \land R \land \Lambda'_a$ is strictly stronger than $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u'_a, \Lambda'_a) \cdot (u'', \Lambda'')) \land R \land \Lambda_a$.
4. There is $(u'', \Lambda'') \in S$ such that $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u'_a, \Lambda'_a) \cdot (u'', \Lambda'')) \land R \land \Lambda_a$ is strictly stronger than $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u_a, \Lambda_a) \cdot (u'', \Lambda'')) \land R \land \Lambda'_a$.

When Item 1 holds, by definition of (u_a, Λ_a) and (u'_a, Λ'_a), there is $w \in (u, \Lambda)$ such that $w \cdot a = w_a$ and for any $w' \in (u', \Lambda')$, we have $\kappa(w \cdot a), \kappa(w' \cdot a) \not\models R$. For such w and w', we have $\tau_{|u_a|-1} = \tau_{|u_a|}$ and $\tau'_{|u'_a|-1} = \tau_{|u'_a|}$ in $\kappa(w \cdot a)$ and $\kappa(w' \cdot a)$. Therefore, when we have Item 1, there is $w \in (u, \Lambda)$ such that for any $w' \in (u', \Lambda')$, we have $\kappa(w), \kappa(w') \models R'$, and we have $p \not\sim^{S',R'}_{\mathcal{L}_{\mathrm{tgt}}} p'$. Similarly, Item 2 also implies $p \not\sim^{S',R'}_{\mathcal{L}_{\mathrm{tgt}}} p'$.

Since $(u_a, \Lambda_a) \cdot (u'', \Lambda'') = (u, \Lambda) \cdot (a \cdot (u'', \Lambda''))$ holds, $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u_a, \Lambda_a) \cdot (u'', \Lambda'')) \land R \land \Lambda'_a$ is equivalent to $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u, \Lambda) \cdot (a \cdot (u'', \Lambda''))) \land R' \land \Lambda'$ with a variable renaming. Similarly, $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u', \Lambda') \cdot (a \cdot (u'', \Lambda''))) \land R' \land \Lambda$ is equivalent to $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u'_a, \Lambda'_a) \cdot (u'', \Lambda'')) \land R \land \Lambda_a$ with a variable renaming. Therefore, when we have Item 3, $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u, \Lambda) \cdot (a \cdot (u'', \Lambda''))) \land R \land \Lambda'$ is also strictly stronger than $\mathtt{mem}^{\mathtt{sym}}_{\mathcal{L}_{\mathrm{tgt}}}((u', \Lambda') \cdot (a \cdot (u'', \Lambda''))) \land R \land \Lambda$, and we have $p \not\sim^{S',R'}_{\mathcal{L}_{\mathrm{tgt}}} p'$. Similarly, Item 4 also implies $p \not\sim^{S',R'}_{\mathcal{L}_{\mathrm{tgt}}} p'$.

D Details of the implementation

Here, we describe some details of our prototype library LearnTA. LearnTA is implemented in C++17 using Boost (as a general-purpose library) and Eigen. We implemented LearnTA focusing on DTAs without unobservable transitions, i. e., each edge is labeled with an event $a \in \Sigma$. To construct a DTA without unobservable transitions, we require the following condition to a timed observation table in addition to the cohesion. This can make P larger but does not harm the termination of the learning algorithm intuitively because of the finiteness of the region graph.

Definition 46 (time-saturation). *A timed observation table* (P, S, T) *is* time-saturated *if for each* $p \in P$ $\mathrm{succ}^t(p) \notin P$ *implies* $p \sim^{S,\top}_{\mathcal{L}_{\mathrm{tgt}}} \mathrm{succ}^t(p)$.

On timed conditions We use *difference bounded matrices (DBM)* [18] to represent timed conditions. We use Eigen to implement DBMs. A DBM represents a finite conjunction of inequalities constraints of the form $x - x' \prec d$, $x \prec d$, or $-x \prec d$, where x and x' are variables, d is a constant, and $\prec \in \{<, \leq\}$. We use a DBM over $\{\mathbb{T}_{0,n}, \mathbb{T}_{1,n}, \dots, \mathbb{T}_{n,n}\}$ to encode a timed condition over $\mathbb{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$. By this encoding, we can represent any timed condition because $\mathbb{T}_{i,j} = \mathbb{T}_{i,n} - \mathbb{T}_{j+1,n}$, where $0 \leq i \leq j < n$.

On the generation of candidate renaming equations As we show in Appendix B.2, when we check if $(u, \Lambda) \sim^{S}_{\mathcal{L}} (u', \Lambda')$, we generate candidate renaming equations R and check if $(u, \Lambda) \sim^{S,R}_{\mathcal{L}} (u', \Lambda')$. As we have already mentioned, such a candidate renaming equation is generated from a bipartite graph constructed from Λ and Λ'. In our implementation, to make the DTA construction simpler, we focus on a reaming equation such that the morphism defined by $(u, \Lambda, u', \Lambda', R)$ is a function. More concretely, we generate renaming equations such that $R = \top$ (i. e., R does not constrain the value after the morphism) or R constrains $\mathbb{T}'_{i,n}$ if the value of $\mathbb{T}'_{i,n}$ is not uniquely defined in Λ'. Such a restriction does not affect the termination of the learning algorithm.

On "continuous consistency" As we observe in Example 44, we cannot enforce "continuous consistency" to timed observation tables. Nevertheless, we can often refine S by adding some dwell time before $s \in S$. In our implementation, to reduce the number of equivalence queries, when the timed observation table is continuously inconsistent, we try to add some dwell time before $s \in S$ and refine S if it indeed refines S.

On DTA construction To simplify the DTA construction, we use clock renaming (i. e., $c := c'$ for $c, c' \in C$) and constant assignments (i. e., $c := d$ for $c \in C$ and $d \in \mathbb{N}$). It is known that these extension does not change the expressive power [15] but can reduce the number of states. We also simplify the candidate DTAs by merging transitions with the same source/target locations, the same resets, and juxtaposed guards, i. e., the union of two guards is still a guard. Moreover, we remove the "dead" locations, i. e., locations unreachable to any accepting locations, by a zone-based analysis. Such simplification makes the constructed DTAs more interpretable by a human.

As we observe, e. g., in Fig. 2c, our DTA learning algorithm generates a DTA with redundant clock variables due to the DTA construction in [29]. It is a future work to apply clock reduction techniques [34,23] to generate a DTA with fewer clock variables.

E Detail of the benchmarks

Here, we present the detail of the benchmarks.

E.1 Random

Random is a benchmark taken from [7]. Random consists of five classes: 3_2_10, 4_2_10, 4_4_20, 5_2_10, and 6_2_10, where each value of $|L|\_|\Sigma|\_M$ is the number

Table 4: Summary of the complexity of each DTA in Random. $|L|$ is the number of locations, $|\Sigma|$ is the alphabet size, and K_C is the maximum constant in the guards.

	$\|L\|$	$\|\Sigma\|$	K_C
3_2_10/3_2_10-1	3	2	7
3_2_10/3_2_10-10	3	2	9
3_2_10/3_2_10-2	3	2	8
3_2_10/3_2_10-3	3	2	10
3_2_10/3_2_10-4	3	2	7
3_2_10/3_2_10-5	3	2	5
3_2_10/3_2_10-6	3	2	10
3_2_10/3_2_10-7	3	2	6
3_2_10/3_2_10-8	3	2	10
3_2_10/3_2_10-9	3	2	9
4_2_10/4_2_10-1	4	2	9
4_2_10/4_2_10-10	4	2	10
4_2_10/4_2_10-2	4	2	9
4_2_10/4_2_10-3	4	2	10
4_2_10/4_2_10-4	4	2	9
4_2_10/4_2_10-5	4	2	10
4_2_10/4_2_10-6	4	2	8
4_2_10/4_2_10-7	4	2	7
4_2_10/4_2_10-8	4	2	6
4_2_10/4_2_10-9	4	2	10
4_4_20/4_4_20-1	4	4	20
4_4_20/4_4_20-10	4	4	20
4_4_20/4_4_20-2	4	4	20
4_4_20/4_4_20-3	4	4	20
4_4_20/4_4_20-4	4	4	20
4_4_20/4_4_20-5	4	4	18
4_4_20/4_4_20-6	4	4	19
4_4_20/4_4_20-7	4	4	20
4_4_20/4_4_20-8	4	4	20
4_4_20/4_4_20-9	4	4	19
5_2_10/5_2_10-1	5	2	9
5_2_10/5_2_10-10	5	2	10
5_2_10/5_2_10-2	5	2	10
5_2_10/5_2_10-3	5	2	10
5_2_10/5_2_10-4	5	2	10
5_2_10/5_2_10-5	5	2	10
5_2_10/5_2_10-6	5	2	9
5_2_10/5_2_10-7	5	2	8
5_2_10/5_2_10-8	5	2	10
5_2_10/5_2_10-9	5	2	9
6_2_10/6_2_10-1	5	2	8
6_2_10/6_2_10-10	5	2	10
6_2_10/6_2_10-2	5	2	7
6_2_10/6_2_10-3	5	2	10
6_2_10/6_2_10-4	5	2	10
6_2_10/6_2_10-5	5	2	10
6_2_10/6_2_10-6	5	2	10
6_2_10/6_2_10-7	5	2	9
6_2_10/6_2_10-8	5	2	10
6_2_10/6_2_10-9	5	2	10

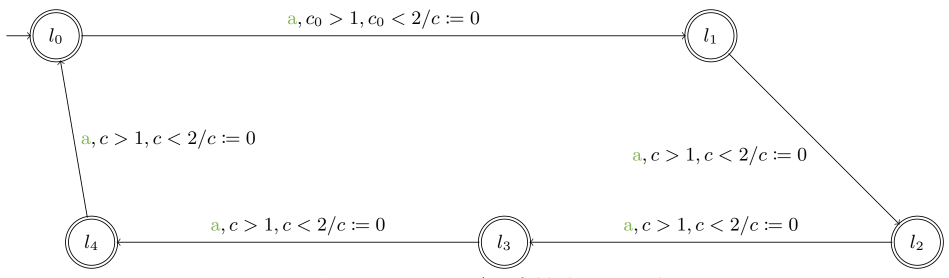


Fig. 6: The target DTA of Unbalanced:1

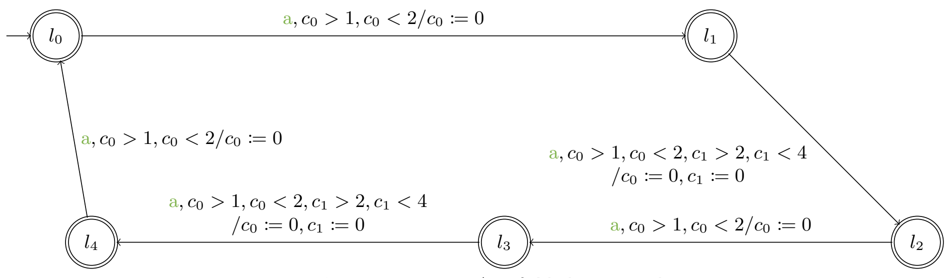

l_0
l_1
l_2
l_3
l_4
a, $c_0 > 1, c_0 < 2/c_0 := 0$
a, $c_0 > 1, c_0 < 2/c_0 := 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$
$/c_0 := 0, c_1 := 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$
$/c_0 := 0, c_1 := 0$
a, $c_0 > 1, c_0 < 2/c_0 := 0$

Fig. 7: The target DTA of Unbalanced:2

of locations, the alphabet size, and the upper bound of the maximum constant in the guards in the DTAs, respectively. Each class consists of 10 randomly generated DTAs. Table 4 shows the summary of the complexity of each DTA. The maximum constant K_C in the guards can be smaller than its upper bound due to its random construction.

E.2 Unbalanced

Unbalanced is our original benchmark consisting of five DTAs. Figs. 6 to 10 illustrate the target DTAs. As shown in Figs. 6 to 10, the target DTAs have the same structure with a different number of clock variables and timing constraints.

E.3 AKM

AKM is a benchmark on an Authentication and Key management service of the WiFi. AKM is initially used in [37]. We took the model from [1].

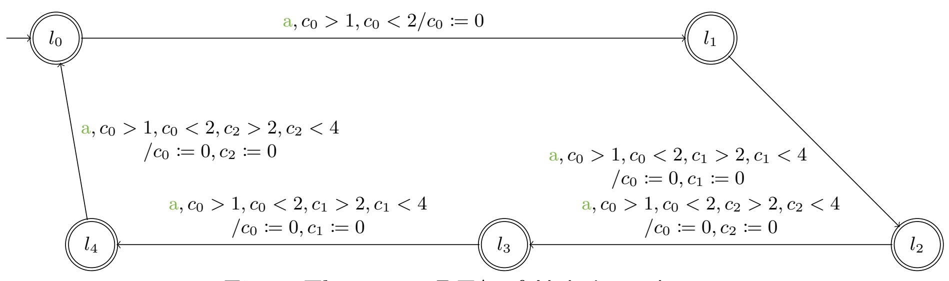

l_0
l_1
l_2
l_3
l_4
a, $c_0 > 1, c_0 < 2/c_0 := 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4$
$/c_0 := 0, c_2 := 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$
$/c_0 := 0, c_1 := 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$
$/c_0 := 0, c_1 := 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4$
$/c_0 := 0, c_2 := 0$

Fig. 8: The target DTA of Unbalanced:3

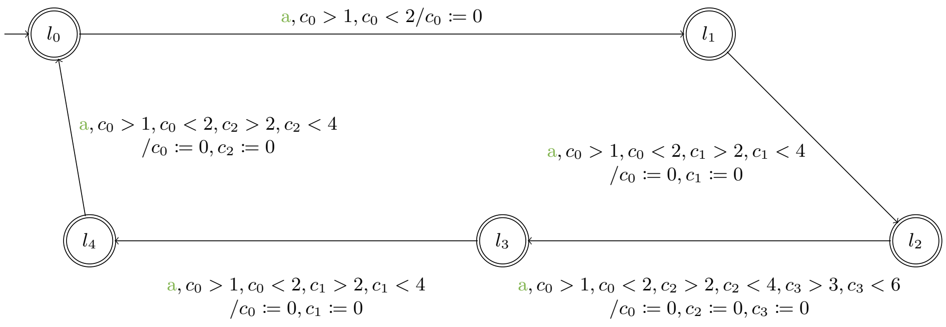

l_0
l_1
l_2
l_3
l_4
a, $c_0 > 1, c_0 < 2 / c_0 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$ $/ c_0 \coloneqq 0, c_1 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4, c_3 > 3, c_3 < 6$ $/ c_0 \coloneqq 0, c_2 \coloneqq 0, c_3 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$ $/ c_0 \coloneqq 0, c_1 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4$ $/ c_0 \coloneqq 0, c_2 \coloneqq 0$

Fig. 9: The target DTA of Unbalanced:4

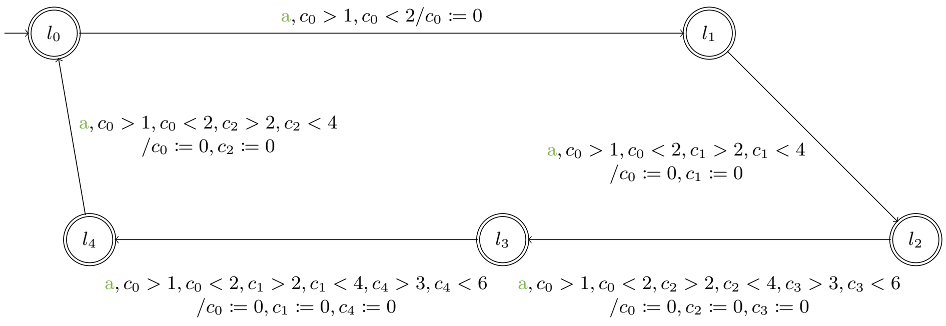

l_0
l_1
l_2
l_3
l_4
a, $c_0 > 1, c_0 < 2 / c_0 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4$ $/ c_0 \coloneqq 0, c_1 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4, c_3 > 3, c_3 < 6$ $/ c_0 \coloneqq 0, c_2 \coloneqq 0, c_3 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_1 > 2, c_1 < 4, c_4 > 3, c_4 < 6$ $/ c_0 \coloneqq 0, c_1 \coloneqq 0, c_4 \coloneqq 0$
a, $c_0 > 1, c_0 < 2, c_2 > 2, c_2 < 4$ $/ c_0 \coloneqq 0, c_2 \coloneqq 0$

Fig. 10: The target DTA of Unbalanced:5

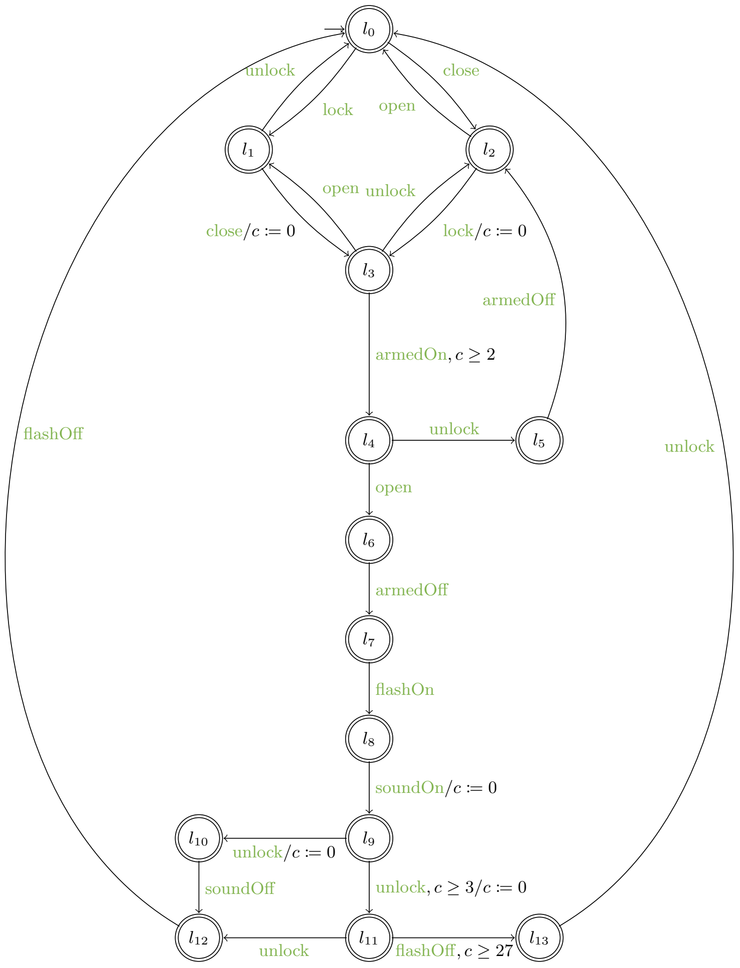


Fig. 11: The target DTA in CAS

E.4 CAS

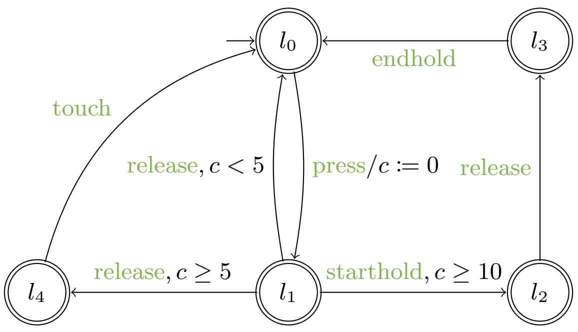
CAS is a benchmark on a car alarm system. Fig. 11 illustrates the target DTA. CAS is initially used in [4]. We took the model from [1].

E.5 Light

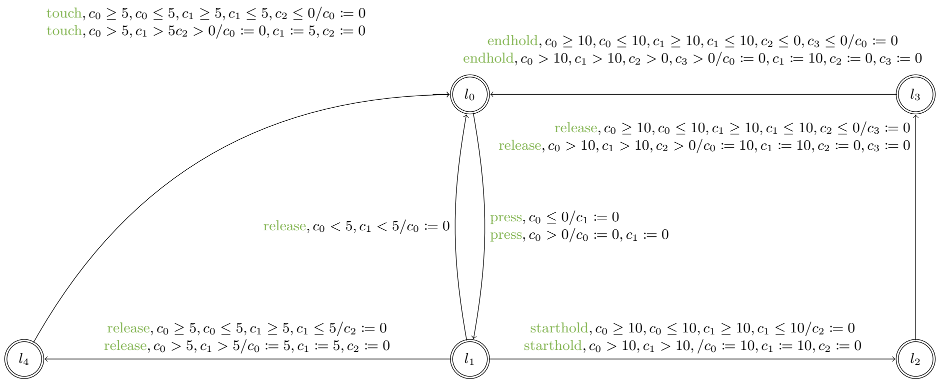
Light is a benchmark on a smart light switch. Light is initially used in [5] inspired by [25]. Fig. 12a illustrates the target DTA. We took the model from [1].

E.6 PC

PC is a benchmark on a particle counter. PC is initially used in [3]. We took the model from [1].

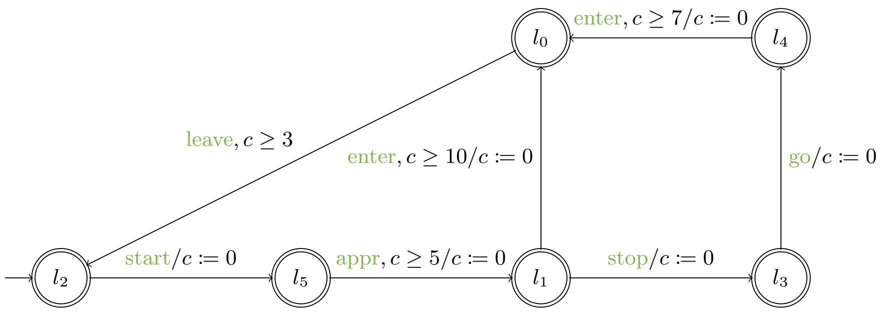


(a) The target DTA

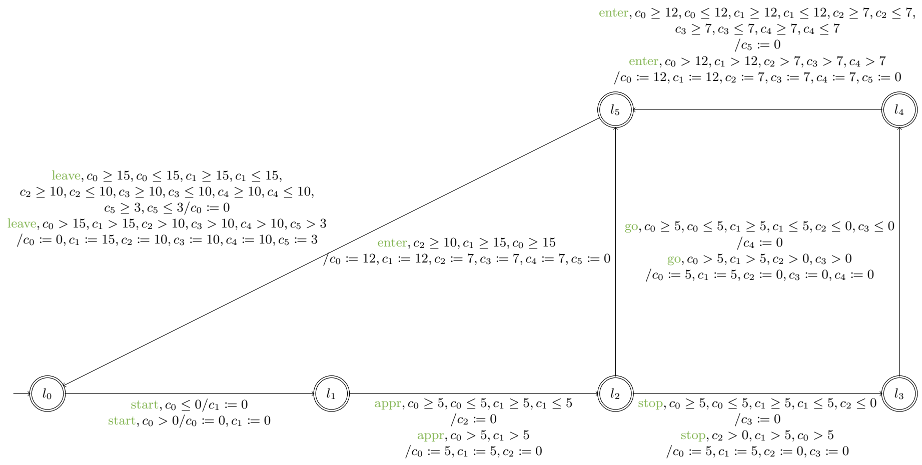

touch, $c_0 \geq 5, c_0 \leq 5, c_1 \geq 5, c_1 \leq 5, c_2 \leq 0/c_0 := 0$
touch, $c_0 > 5, c_1 > 5c_2 > 0/c_0 := 0, c_1 := 5, c_2 := 0$
endhold, $c_0 \geq 10, c_0 \leq 10, c_1 \geq 10, c_1 \leq 10, c_2 \leq 0, c_3 \leq 0/c_0 := 0$
endhold, $c_0 > 10, c_1 > 10, c_2 > 0, c_3 > 0/c_0 := 0, c_1 := 10, c_2 := 0, c_3 := 0$
l_0
l_3
release, $c_0 \geq 10, c_0 \leq 10, c_1 \geq 10, c_1 \leq 10, c_2 \leq 0/c_3 := 0$
release, $c_0 > 10, c_1 > 10, c_2 > 0/c_0 := 10, c_1 := 10, c_2 := 0, c_3 := 0$
release, $c_0 < 5, c_1 < 5/c_0 := 0$
press, $c_0 \leq 0/c_1 := 0$
press, $c_0 > 0/c_0 := 0, c_1 := 0$
release, $c_0 \geq 5, c_0 \leq 5, c_1 \geq 5, c_1 \leq 5/c_2 := 0$
release, $c_0 > 5, c_1 > 5/c_0 := 5, c_1 := 5, c_2 := 0$
starthold, $c_0 \geq 10, c_0 \leq 10, c_1 \geq 10, c_1 \leq 10/c_2 := 0$
starthold, $c_0 > 10, c_1 > 10, /c_0 := 10, c_1 := 10, c_2 := 0$
l_4
l_1
l_2

(b) The learned DTA after the simplification in Appendix D, e. g., removing the "dead" locations. For simplicity, we displace the edges with the same source and target locations on the same edge.

Fig. 12: Light



(a) The target DTA



enter, $c_0 \geq 12, c_0 \leq 12, c_1 \geq 12, c_1 \leq 12, c_2 \geq 7, c_2 \leq 7,$
$c_3 \geq 7, c_3 \leq 7, c_4 \geq 7, c_4 \leq 7$
$/ c_5 \coloneqq 0$
enter, $c_0 > 12, c_1 > 12, c_2 > 7, c_3 > 7, c_4 > 7$
$/ c_0 \coloneqq 12, c_1 \coloneqq 12, c_2 \coloneqq 7, c_3 \coloneqq 7, c_4 \coloneqq 7, c_5 \coloneqq 0$
l_5
l_4
leave, $c_0 \geq 15, c_0 \leq 15, c_1 \geq 15, c_1 \leq 15,$
$c_2 \geq 10, c_2 \leq 10, c_3 \geq 10, c_3 \leq 10, c_4 \geq 10, c_4 \leq 10,$
$c_5 \geq 3, c_5 \leq 3 / c_0 \coloneqq 0$
leave, $c_0 > 15, c_1 > 15, c_2 > 10, c_3 > 10, c_4 > 10, c_5 > 3$
$/ c_0 \coloneqq 0, c_1 \coloneqq 15, c_2 \coloneqq 10, c_3 \coloneqq 10, c_4 \coloneqq 10, c_5 \coloneqq 3$
enter, $c_2 \geq 10, c_1 \geq 15, c_0 \geq 15$
$/ c_0 \coloneqq 12, c_1 \coloneqq 12, c_2 \coloneqq 7, c_3 \coloneqq 7, c_4 \coloneqq 7, c_5 \coloneqq 0$
go, $c_0 \geq 5, c_0 \leq 5, c_1 \geq 5, c_1 \leq 5, c_2 \leq 0, c_3 \leq 0$
$/ c_4 \coloneqq 0$
go, $c_0 > 5, c_1 > 5, c_2 > 0, c_3 > 0$
$/ c_0 \coloneqq 5, c_1 \coloneqq 5, c_2 \coloneqq 0, c_3 \coloneqq 0, c_4 \coloneqq 0$
l_0
l_1
l_2
l_3
start, $c_0 \leq 0 / c_1 \coloneqq 0$
start, $c_0 > 0 / c_0 \coloneqq 0, c_1 \coloneqq 0$
appr, $c_0 \geq 5, c_0 \leq 5, c_1 \geq 5, c_1 \leq 5$
$/ c_2 \coloneqq 0$
appr, $c_0 > 5, c_1 > 5$
$/ c_0 \coloneqq 5, c_1 \coloneqq 5, c_2 \coloneqq 0$
stop, $c_0 \geq 5, c_0 \leq 5, c_1 \geq 5, c_1 \leq 5, c_2 \leq 0$
$/ c_3 \coloneqq 0$
stop, $c_2 > 0, c_1 > 5, c_0 > 5$
$/ c_0 \coloneqq 5, c_1 \coloneqq 5, c_2 \coloneqq 0, c_3 \coloneqq 0$

(b) The learned DTA after the simplification in Appendix D, e. g., removing the "dead" locations. For simplicity, we displace the edges with the same source and target locations on the same edge.

Fig. 13: Train

E.7 TCP

TCP is a benchmark on a functional specification of TCP protocol. TCP is initially used in [7]. We took the model from [1].

E.8 Train

Train is an abstract model of a train. Fig. 13a illustrates the target DTA. Train is initially used in [36]. We took the model from [1].

E.9 FDDI

FDDI is a benchmark of an abstract model of the FDDI protocol. The model is initially used in [17]. We took a model from TChecker [2] with two processes. The TChecker model of FDDI can be obtained by executing `fddi.sh 2 5 1 0`, where `fddi.sh` is distributed in https://github.com/ticktac-project/tchecker/blob/master/examples/fddi.sh.

F Detailed experiment results

Tables 5 and 6 show the detailed experiment results, where the columns $|L|$ and $|C|$ show the number of locations and the clock variables of the learned DTA.

Table 5: Detailed experiment results

		# of Mem. queries	# of Sym. Mem. queries	# of Eq. queries	Exec. time [sec.]	$\lvert L\rvert$	$\lvert C\rvert$
3_2_10/3_2_10-1	LearnTA	3897	2144	5	1.01e-01	4	3
	OneSMT	59	N/A	6	8.23e-01	3	1
3_2_10/3_2_10-10	LearnTA	22402	5259	6	6.27e-01	3	3
	OneSMT	468	N/A	11	9.58e-01	3	1
3_2_10/3_2_10-2	LearnTA	4192	3196	7	7.70e-02	4	3
	OneSMT	216	N/A	8	1.18e-01	3	1
3_2_10/3_2_10-3	LearnTA	35268	15341	11	2.32e+00	16	4
	OneSMT	263	N/A	12	2.39e-01	3	1
3_2_10/3_2_10-4	LearnTA	4148	2516	5	6.90e-02	6	3
	OneSMT	171	N/A	8	9.71e-02	3	1
3_2_10/3_2_10-5	LearnTA	2830	1532	5	9.20e-02	7	3
	OneSMT	32	N/A	5	6.58e-02	3	1
3_2_10/3_2_10-6	LearnTA	28643	9315	9	1.25e+00	4	4
	OneSMT	175	N/A	9	1.15e-01	3	1
3_2_10/3_2_10-7	LearnTA	31236	12704	9	1.93e+00	11	4
	OneSMT	387	N/A	13	2.80e-01	3	1
3_2_10/3_2_10-8	LearnTA	6914	3181	5	1.71e-01	3	2
	OneSMT	188	N/A	9	1.03e-01	3	1
3_2_10/3_2_10-9	LearnTA	2887	1850	4	4.50e-02	3	2
	OneSMT	91	N/A	7	9.23e-02	3	1
4_2_10/4_2_10-1	LearnTA	47914	23601	9	7.71e+00	13	3
	OneSMT	271	N/A	14	1.50e-01	4	1
4_2_10/4_2_10-10	LearnTA	10619	6186	4	4.88e-01	12	4
	OneSMT	396	N/A	10	3.53e-01	4	1
4_2_10/4_2_10-2	LearnTA	14258	9310	9	2.17e+00	14	3
	OneSMT	257	N/A	7	1.27e-01	4	1
4_2_10/4_2_10-3	LearnTA	29445	8279	8	9.55e-01	9	3
	OneSMT	262	N/A	13	1.44e-01	4	1
4_2_10/4_2_10-4	LearnTA	194442	69285	14	2.48e+01	26	4
	OneSMT	776	N/A	14	2.45e-01	4	1
4_2_10/4_2_10-5	LearnTA	22111	6815	7	1.06e+00	5	4
	OneSMT	366	N/A	14	1.77e-01	4	1
4_2_10/4_2_10-6	LearnTA	77864	30636	8	2.65e+01	14	4
	OneSMT	985	N/A	10	3.03e-01	4	1
4_2_10/4_2_10-7	LearnTA	98232	26569	6	9.18e+00	15	4
	OneSMT	272	N/A	13	1.65e-01	4	1
4_2_10/4_2_10-8	LearnTA	28427	10235	8	1.78e+00	14	3
	OneSMT	255	N/A	10	1.50e-01	4	1
4_2_10/4_2_10-9	LearnTA	36656	16403	6	5.15e+00	14	3
	OneSMT	670	N/A	16	2.76e-01	4	1
4_4_20/4_4_20-1	LearnTA	613498	209071	21	5.90e+01	14	4
	OneSMT	5329	N/A	35	1.54e+00	4	1
4_4_20/4_4_20-10	LearnTA	670040	304827	15	2.59e+02	26	4
	OneSMT	4140	N/A	42	2.19e+00	4	1
4_4_20/4_4_20-2	LearnTA	T/O	T/O	T/O	T/O	T/O	T/O
	OneSMT	4239	N/A	37	1.65e+00	4	1
4_4_20/4_4_20-3	LearnTA	T/O	T/O	T/O	T/O	T/O	T/O
	OneSMT	3961	N/A	29	1.60e+00	4	1
4_4_20/4_4_20-4	LearnTA	933187	331561	20	2.42e+02	22	3
	OneSMT	1740	N/A	26	1.18e+00	4	1
4_4_20/4_4_20-5	LearnTA	1055910	373123	14	4.94e+02	35	3
	OneSMT	1966	N/A	26	8.27e-01	4	1
4_4_20/4_4_20-6	LearnTA	1048528	492591	18	6.70e+02	34	3
	OneSMT	3013	N/A	32	1.62e+00	4	1
4_4_20/4_4_20-7	LearnTA	248399	125653	10	3.23e+01	18	3
	OneSMT	2652	N/A	30	1.19e+00	4	1
4_4_20/4_4_20-8	LearnTA	618743	186476	14	1.15e+03	11	3
	OneSMT	3863	N/A	36	1.40e+00	4	1
4_4_20/4_4_20-9	LearnTA	1681769	520310	11	8.34e+03	34	5
	OneSMT	4074	N/A	35	1.01e+00	4	1

Table 6: Detailed experiment results

		# of Mem. queries	# of Sym. Mem. queries	# of Eq. queries	Exec. time [sec.]	$\|L\|$	$\|C\|$
5_2_10/5_2_10-1	LearnTA	25038	10299	7	7.85e-01	8	3
	OneSMT	1332	N/A	15	5.20e-01	5	1
5_2_10/5_2_10-10	LearnTA	21615	10292	10	7.84e-01	7	3
	OneSMT	1093	N/A	22	4.04e-01	5	1
5_2_10/5_2_10-2	LearnTA	32562	13085	6	2.41e+00	13	5
	OneSMT	359	N/A	12	2.58e-01	4	1
5_2_10/5_2_10-3	LearnTA	37725	18962	8	6.62e+00	12	6
	OneSMT	652	N/A	16	2.84e-01	5	1
5_2_10/5_2_10-4	LearnTA	8121	4597	5	1.96e-01	5	3
	OneSMT	778	N/A	12	4.30e-01	4	1
5_2_10/5_2_10-5	LearnTA	334000	63946	12	1.67e+02	23	4
	OneSMT	752	N/A	15	3.01e-01	5	1
5_2_10/5_2_10-6	LearnTA	627980	69007	19	4.36e+01	8	4
	OneSMT	1159	N/A	16	4.55e-01	5	1
5_2_10/5_2_10-7	LearnTA	33157	14270	8	2.57e+00	11	5
	OneSMT	639	N/A	16	3.31e-01	5	1
5_2_10/5_2_10-8	LearnTA	56059	15114	6	2.17e+00	8	5
	OneSMT	896	N/A	20	3.41e-01	5	1
5_2_10/5_2_10-9	LearnTA	22807	12003	8	1.97e+00	17	6
	OneSMT	1100	N/A	18	3.35e-01	5	1
6_2_10/6_2_10-1	LearnTA	2912	1976	7	4.40e-02	4	2
	OneSMT	104	N/A	11	1.73e-01	4	1
6_2_10/6_2_10-10	LearnTA	100795	23336	9	4.67e+00	8	5
	OneSMT	2122	N/A	16	8.46e-01	6	1
6_2_10/6_2_10-2	LearnTA	3859	3060	6	1.08e-01	5	5
	OneSMT	910	N/A	14	6.89e-01	5	1
6_2_10/6_2_10-3	LearnTA	36219	21419	8	3.62e+00	12	7
	OneSMT	1445	N/A	28	7.68e-01	6	1
6_2_10/6_2_10-4	LearnTA	26780	9860	11	9.83e-01	6	4
	OneSMT	3124	N/A	27	8.77e-01	6	1
6_2_10/6_2_10-5	LearnTA	555939	70427	12	2.44e+02	16	5
	OneSMT	2593	N/A	16	8.27e-01	6	1
6_2_10/6_2_10-6	LearnTA	73012	28542	12	1.13e+01	7	5
	OneSMT	900	N/A	19	5.05e-01	5	1
6_2_10/6_2_10-7	LearnTA	208647	26577	10	1.37e+01	11	5
	OneSMT	1748	N/A	14	6.21e-01	6	1
6_2_10/6_2_10-8	LearnTA	33971	11466	10	1.19e+00	6	4
	OneSMT	2073	N/A	28	9.83e-01	6	1
6_2_10/6_2_10-9	LearnTA	22651	12691	14	1.95e+00	6	3
	OneSMT	3929	N/A	35	1.72e+00	6	1
AKM	LearnTA	12263	10881	11	5.85e-01	12	7
	OneSMT	3453	N/A	49	7.97e+00	12	1
CAS	LearnTA	66067	42611	17	4.65e+00	14	10
	OneSMT	4769	N/A	18	9.58e+01	14	1
FDDI	LearnTA	9986271	2408549	43	3.00e+03	348	10
PC	LearnTA	245134	162997	23	6.49e+01	25	10
	OneSMT	10390	N/A	29	1.24e+02	25	1
TCP	LearnTA	11300	11337	15	3.82e-01	20	9
	OneSMT	4713	N/A	32	2.20e+01	20	1
Train	LearnTA	13487	8424	8	1.72e-01	6	6
	OneSMT	838	N/A	13	1.13e+00	6	1
Unbalanced:1	LearnTA	51	52	2	2.00e-03	1	1
Unbalanced:2	LearnTA	576142	14439	3	3.64e+01	25	7
Unbalanced:3	LearnTA	403336	13001	4	2.24e+01	25	7
Unbalanced:4	LearnTA	4142835	46204	5	2.40e+02	93	8
Unbalanced:5	LearnTA	10691400	86184	5	8.68e+02	140	9