Learning Nondeterministic Real-Time Automata

JIE AN, Max Planck Institute for Software Systems, Germany
BOHUA ZHAN and NAIJUN ZHAN, SKLCS and Science & Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
MIAOMIAO ZHANG, School of Software Engineering, Tongji University, China

We present an active learning algorithm named NRTALearning for nondeterministic real-time automata (NRTAs). Real-time automata (RTAs) are a subclass of timed automata with only one clock which resets at each transition. First, we prove the corresponding Myhill-Nerode theorem for real-time languages. Then we show that there exists a unique minimal deterministic real-time automaton (DRTA) recognizing a given real-time language, but the same does not hold for NRTAs. We thus define a special kind of NRTAs, named residual real-time automata (RRTAs), and prove that there exists a minimal RRTA to recognize any given real-time language. This transforms the learning problem of NRTAs to the learning problem of RRTAs. After describing the learning algorithm in detail, we prove its correctness and polynomial complexity. In addition, based on the corresponding Myhill-Nerode theorem, we extend the existing active learning algorithm NL^* for nondeterministic finite automata to learn RRTAs. We evaluate and compare the two algorithms on two benchmarks consisting of randomly generated NRTAs and rational regular expressions. The results show that NRTALearning generally performs fewer membership queries and more equivalence queries than the extended NL^* algorithm, and the learnt NRTAs have much fewer locations than the corresponding minimal DRTAs. We also conduct a case study using a model of scheduling of final testing of integrated circuits.

CCS Concepts: • **Computer systems organization** → **Real-time languages**; • **Theory of computation** → *Regular languages.*

Additional Key Words and Phrases: active learning, model learning, nondeterministic real-time automata, real-time languages

ACM Reference Format:
Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2021. Learning Nondeterministic Real-Time Automata. *ACM Trans. Embedd. Comput. Syst.* xx, x, Article 7 (October 2021), 26 pages. https://doi.org/10.1145/3477030

1 INTRODUCTION

In recent decades, model learning has attracted increasing attentions in many communities, especially formal methods and artificial intelligence, since it has wide applications in model checking [3], analysis of protocols [17], grammatical inference [11], interpretation of neural networks [34, 35] and so on. For model learning technique, a seminal work is the minimally adequate teacher (MAT) framework [7] proposed by Angluin to learn regular languages in 1987. In the MAT framework, a learner actively learns a regular language from a teacher using membership and equivalence queries. For a membership query, the learner asks whether a word belongs to the target language.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.
Authors' addresses: Jie An, jiean@mpi-sws.org, Max Planck Institute for Software Systems, Paul-Ehrlich Str. G 26, Kaiserslautern, Germany, 67663; Bohua Zhan, bzhan@ios.ac.cn; Naijun Zhan, znj@ios.ac.cn, SKLCS and Science & Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China, 100190 and University of Chinese Academy of Sciences, Beijing, China, 100049; Miaomiao Zhang, miaomiao@tongji.edu.cn, School of Software Engineering, Tongji University, Shanghai, China, 201804.

For an equivalence query, the learner submits a deterministic finite automaton to the teacher as a hypothesis for the target language. The teacher can answer yes or no for the queries and in the latter case, provide a counterexample as evidence for the differences between the current hypothesis and the target. Following this approach, many efficient algorithms have been proposed for active learning of different kinds of automata. We refer to the survey [27] for a comprehensive introduction.

For real-time and embedded systems, timing constraints play a key role in the correctness and safety of the system. Classical finite automata is unable to describe the infinite number of timed actions. Instead, *Timed automata* [2], a kind of finite automata extended with a finite number of real-valued clocks, are widely used to model real-time and embedded systems. In this paper, we consider the active learning problem of *real-time automata* (RTAs), a subclass of timed automata with a single clock that resets at every transition, with the goal to learn nondeterministic RTAs for real-time languages in the MAT framework. RTAs yield simple models while preserving adequate expressiveness, and therefore have been widely used in practical real-time systems, e.g. scheduling of real-time tasks [25] and key-distribution protocols [14]. In [15], Dima introduced real-time automata and revealed some of its important properties, including the Kleene theorem for RTAs, which shows that the expressiveness of rational regular expressions, deterministic real-time automata (DRTAs) and nondeterministic real-time automata (NRTAs) are all equivalent. On the other hand, the Kleene theorem also tells us that a DRTA can be exponentially bigger than an equivalent NRTA in terms of the number of locations, which is similar to the situation of DFAs and NFAs. Hence, compared to learning DRTAs, learning NRTAs may lead to more succinct models, which will be more useful, especially for applications in verification.

The cornerstone of learning regular languages in the MAT framework is the Myhill-Nerode theorem which shows that a language L is regular if and only if the right-congruence relation R_L has a finite number of equivalence classes. Thus, we can map each equivalence class to a location (state) in a finite automaton, and this implies that there is a unique minimal DFA which recognizes L. However, there is no unique minimal NFA for the language, which means it is not clear which target automaton should be learned in the MAT framework. Hence, Bollig et al. proposed an algorithm named NL^* [8] to learn a subclass of NFAs named residual finite state automata (RFSAs) [12], which have the property that there is a unique minimal RFSA recognizing a given regular language. Inspired by their work, we first prove a corresponding Myhill-Nerode theorem for real-time languages which shows that there exists a unique minimal DRTA for a given real-time language. However, there is no unique minimal NRTA, so we proceed to define the notions of residual real-time languages and residual real-time automata (RRTAs). By proving that there exists a unique minimal RRTA for a given real-time language, we transform the learning problem of NRTAs to the learning problem of RRTAs. Compared to NL^*, the challenge is to handle nondeterministic behaviours caused by timing information in timed words instead of untimed actions. By carefully designing the readiness conditions of the observation table and the process for handling counterexamples, we present an active learning algorithm named NRTALearning. Based on the corresponding Myhill-Nerode theorem, we can also directly extend NL^* to learn RRTAs. We prove the correctness and termination of both algorithms, and show the polynomial complexity of both algorithms in terms of the number of queries. The two algorithms have been implemented and evaluated on two benchmarks consisting of randomly generated NRTAs and rational regular expressions. The results show that the algorithm NRTALearning generally performs fewer membership queries and more equivalence queries than the extended NL^* algorithm. Additionally, the learnt NRTAs have much fewer locations than the corresponding minimal DRTAs. Finally, we show a case study using a model of scheduling of final testing of integrated circuits.

In summary, our main contributions are as follows.

- A version of Myhill-Nerode theorem for real-time languages.
- Definition of residual real-time automata, and a proof that there is a unique minimal RRTA recognizing a given real-time language.
- Two efficient active learning algorithms for NRTAs. One is in the standard MAT framework, and the other needs an assumption.
- Implementation and experimental evaluation on two benchmarks for learning NRTAs and rational regular expressions[1].

Related work. There are several works on learning timed models. We first introduce works in the active learning paradigm. In [18], Grinchtein et al. proposed a learning algorithm for deterministic event-recording automata (ERAs) which are a kind of timed automata that, for every untimed action a, a clock is used to record the time length from the last occurrence of a to now. In [19], Henry et al. considered learning a kind of deterministic ERAs, named reset-free ERAs. However, Dima pointed out that RTAs are incomparable to ERAs since RTA may accept languages consisting of two actions separated by an interval with integer length while ERAs may not [15]. An et al. proposed an active learning algorithm for deterministic one-clock timed automata in [4]. However, guessing reset information of the clock leads to a combinatorial explosion in the number of candidate tables, and thus an exponential complexity. Hence, they considered learning DRTAs with a similar technique [5]. In this paper, we consider learning NRTAs directly. Another kind of simple timed model is called Mealy machine with timers. The value of each timer decreases and a timeout is triggered when the value becomes 0. Caldwell et al. proposed an algorithm on learning such a model from programmable logic controllers [10]. In [28], Vaandrager et al. presented an efficient learning algorithm for such models with one timer. Passive learning for timed models has also attracted much interest. Passive learning aims at identifying a model from a given data set and the learnt model is only required to be consistent with the data set. Based on the classic identification method for DFAs, named evidence-driven state-merging (EDSM), Verwer et al. proposed the RTI algorithm for identifying DRTAs in the limit [32, 33]. After that, they presented a passive learning algorithm for deterministic one-clock timed automata [29–31]. Additionally, the passive learning methods cited above concern only discrete-time semantics of the timed models, i.e., the clock valuations are non-negative integers. There are also some works incorporating techniques from machine learning, e.g., learning deterministic timed automata via genetic programming (GP) [26] and learning probabilistic real-time automata via clustering techniques [23]. Recently, Aichernig et al. extended their GP-based learning method in an active manner using conformance testing [1] and successfully learned models with large size. The conformance relation requires that the learnt deterministic model and the system under test agree on a finite set of sampled traces. Thus it cannot guarantee the correctness of the learnt model. Even such techniques have been applied to learn hybrid automata [20, 24]. To the best of our knowledge, our paper is the first work on active learning of a kind of nondeterministic timed automata.

Organization. In the following, Section 2 recalls important preliminary definitions. The corresponding Myhill-Nerode Theorem for real-time languages is presented and proved in Section 3. We then define residual real-time languages and residual real-time automata in Section 4. Two active learning algorithms for NRTAs are proposed in Section 5 and evaluated in Section 6. Finally, we conclude the paper in Section 7.

2 PRELIMINARIES

In this section, we recall some notions including timed words, timed automata, real-time automata and their recognized timed languages. Let $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ be the set of non-negative real numbers and

[1]The prototype and experiments are available at https://github.com/Leslieaj/NRTALearning.

natural numbers, respectively, and $\mathbb{B}$ the Boolean set. We use $\top$ to stand for true and $\bot$ for false. Let Σ be a set of actions considered in this paper.

A *(delay) timed word* over $\Sigma \times \mathbb{R}_{\geq 0}$ is a finite sequence $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, where $\sigma_i \in \Sigma$ and $\tau_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq n$, $|\omega| = n$ is the length of ω, and each τ_i represents the delay time between two consecutive actions. We use ϵ to represent the empty word with $|\epsilon| = 0$. A timed word ω is called a *timed action* if $|\omega| = 1$. A *timed language* $\mathcal{L}$ can be viewed as a set of timed words.

Timed automata [2], an extension of finite automata with a finite number of real-valued clocks, are widely used to model real-time systems. Let C be the set of clock variables, denote by Φ_C the set of *clock constraints* of the form $\phi ::= \top \mid c \bowtie m \mid \phi \wedge \phi$, where $c \in C$, $m \in \mathbb{N}$ and $\bowtie \in \{=, <, >, \leq, \geq\}$. A *clock valuation* is a function $\nu \colon C \to \mathbb{R}_{\geq 0}$ that assigns a non-negative real number to the clocks. For $t \in \mathbb{R}_{\geq 0}$, let $\nu + t$ be the clock valuation with $(\nu + t)(c) = \nu(c) + t$ for all $c \in C$.

In this paper, we consider a subclass of timed automata with a single clock which resets at every transition, termed *real-time automata* [15]. According to the definitions of clock constraint and clock valuation, a transition *guard* in a real-time automaton can be represented by an interval with endpoints in $\mathbb{N} \cup \{\infty\}$. For example, let c be the unique clock, $\phi_1 \colon c < 5 \wedge c \geq 3$ is represented as $[3, 5)$, $\phi_2 \colon c = 6$ as $[6, 6]$, and $\phi_3 \colon \top$ as $[0, \infty)$. We omit the single clock and give a more succinct definition as follows.

Definition 2.1 (Nondeterministic real-time automata). A (nondeterministic) real-time automaton is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ where

- Q is a finite set of locations;
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma \times 2^{\mathbb{R}_{\geq 0}} \times Q$ is a transition relation with $|\Delta| < \infty$, where $2^{\mathbb{R}_{\geq 0}}$ represents the set of intervals whose endpoints are in $\mathbb{N} \cup \{\infty\}$;
- $Q_0 \subseteq Q$ is a finite set of initial locations;
- $F \subseteq Q$ is a finite set of accepting locations.

A transition $(q, \sigma, I, q') \in \Delta$ allows a jump from the *source location* q to the *target location* q' by performing the action $\sigma \in \Sigma$ if the guard I is satisfied (i.e., $\nu(c) \in I$). Meanwhile, clock c is reset to zero. Since the unique clock c resets at every transition, the value of the logic clock c represents the delay time between two actions. Thus, Δ induces the *transition function* $\delta \colon Q \times \Sigma \times \mathbb{R}_{\geq 0} \to 2^Q$ such that $\delta(q, (\sigma, \tau)) = \{q' \in Q \mid (q, \sigma, I, q') \in \Delta, \sigma \in \Sigma, \nu(c) = \tau \in I\}$. We extend δ to $\delta \colon Q \times (\Sigma \times \mathbb{R}_{\geq 0})^* \to 2^Q$ by $\delta(q, \epsilon) = \{q\}$ and $\delta(q, (\sigma, \tau) \cdot \omega) = \bigcup_{q' \in \delta(q, (\sigma, \tau))} \delta(q', \omega)$, and subsequently to the set of locations $Q' \subseteq Q$ by $\delta(Q', \omega) = \bigcup_{q \in Q'} \delta(q, \omega)$.

A *run* of an RTA $\mathcal{A}$ is either a single initial state $\rho = q_0 \in Q_0$ or a finite sequence $\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n$, with $n > 0$, $(q_{i-1}, \sigma_i, I_i, q_i) \in \Delta$, and $\tau_i \in I_i$ for $1 \leq i \leq n$. When action σ_i is being performed, $\nu(c) = \tau_i$. After that, the clock c resets to 0.

The *trace* of a run ρ is a timed word defined as: $trace(q_0) = \epsilon$, and if $\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n$ then $trace(\rho) = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$. For an RTA $\mathcal{A}$, its *recognized timed language* can be defined on traces as $\mathcal{L}(\mathcal{A}) = \{trace(\rho) \mid \rho \text{ starts from } q_0 \in Q_0 \text{ and ends in } q_n \in F\}$. Given an RTA, $\mathcal{L}_q$ denotes the timed language starting from a location $q \in Q$, which is the set of timed words ω such that $\delta(q, \omega) \cap F \neq \emptyset$. In this paper, we consider the *real-time languages* defined as follows.

Definition 2.2 (Real-time languages). Given a timed language $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$, $\mathcal{L}$ is a *real-time language* if $\mathcal{L}$ can be recognized by an RTA $\mathcal{A}$, i.e. $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

An RTA is a *deterministic* real-time automaton (DRTA) if and only if there is at most one run for a given timed word ω, i.e. $|Q_0| = 1$ and $|\delta(q, (\sigma, \tau))| = 1$ for all $q \in Q$ and $(\sigma, \tau) \in \Sigma \times \mathbb{R}_{\geq 0}$.

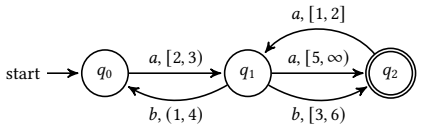


Fig. 1. An NRTA $\mathcal{A}$ with a set $Q_0 = \{q_0\}$ of initial locations and a set $F = \{q_2\}$ of accepting locations.

Otherwise, we call it *nondeterministic real-time automaton* (NRTA). Therefore, given a timed word ω, it is accepted by an NRTA $\mathcal{A}$ if and only if one of its corresponding runs ends in an accepting location $q \in F$ of $\mathcal{A}$. According to the Kleene Theorem for RTAs [15], DRTAs and NRTAs have the same expressiveness.

Example 2.3. Consider NRTA $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ in Fig. 1. The set of locations $Q = \{q_0, q_1, q_2\}$, the finite alphabet $\Sigma = \{a, b\}$, the set of initial locations $Q_0 = \{q_0\}$, the set of accepting locations $F = \{q_2\}$, and $\Delta = \{(q_0, a, [2, 3), q_1), (q_1, a, [5, \infty), q_2), (q_1, b, (1, 4), q_0), (q_1, b, [3, 6), q_2), (q_2, a, [1, 2], q_1)\}$. For the timed word $\omega = (a, 2.1), (b, 3)$, there are two runs in $\mathcal{A}$, i.e., $\rho_1 = q_0 \xrightarrow[2.1]{a} q_1 \xrightarrow[3]{b} q_0$ and $\rho_2 = q_0 \xrightarrow[2.1]{a} q_1 \xrightarrow[3]{b} q_2$, corresponding to it. Clearly, ω is accepted by NRTA $\mathcal{A}$ as ρ_2 ends in an accepting location $q_2 \in F$.

3 THE MYHILL-NERODE THEOREM FOR REAL-TIME LANGUAGES

In order to prove the Myhill-Nerode Theorem for real-time languages, we first recall the notion of the *region* [2]. Since there is only one clock c, given a clock valuation v, we define the region $[\![v]\!]$ containing v as $[\![v]\!] = [v, v]$ if $v \in \mathbb{N}$, and $[\![v]\!] = (\lfloor v \rfloor, \lfloor v \rfloor + 1)$ otherwise, where $\lfloor v \rfloor$ is the integer part of v. As a convention, $[\![v]\!] = (\kappa, \infty)$ if v is greater than the maximum constant κ appearing in the RTA. Hence, there exist $2\kappa + 2$ such regions, including $[n, n]$ with $0 \le n \le \kappa$, $(n, n+1)$ with $0 \le n < \kappa$, and (κ, ∞). We further define the region words as follows.

Definition 3.1 (region words). Given a timed word $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, a word $\gamma = (\sigma_1, [\![\tau_1]\!])(\sigma_1, [\![\tau_2]\!]) \cdots (\sigma_n, [\![\tau_n]\!])$ is the *region word* of ω, denoted as $\gamma = [\![\omega]\!]$.

The key concept in the classic Myhill-Nerode Theorem is that of indistinguishable words (the right-congruence relation). Hence, we introduce two definitions about indistinguishable timed words as follows.

Definition 3.2. Let $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\ge 0})^*$ be any timed language. Two timed words $\omega_1, \omega_2 \in (\Sigma \times \mathbb{R}_{\ge 0})^*$ are *indistinguishable* by $\mathcal{L}$, denoted $\omega_1 \sim_{\mathcal{L}} \omega_2$, if for every timed word $\omega' \in (\Sigma \times \mathbb{R}_{\ge 0})^*$, it holds that $\omega_1 \cdot \omega' \in \mathcal{L}$ if and only if $\omega_2 \cdot \omega' \in \mathcal{L}$.

Definition 3.3. Given a DRTA $\mathcal{A}$, two timed words $\omega_1, \omega_2 \in (\Sigma \times \mathbb{R}_{\ge 0})^*$ are *indistinguishable* by $\mathcal{A}$, denoted $\omega_1 \sim_{\mathcal{A}} \omega_2$, if $\delta(q_0, \omega_1) = \delta(q_0, \omega_2)$, i.e. the reachable location for ω_1 is the same as the reachable location for ω_2.

Lemma 3.4. *Given a DRTA $\mathcal{A}$, for all $\omega_1, \omega_2 \in (\Sigma \times \mathbb{R}_{\ge 0})^*$, $\omega_1 \sim_{\mathcal{A}} \omega_2$ if ω_1 and ω_2 belong to the same region word γ.*

Proof. Depending on the definition of DRTAs and the regions, we have $\delta(q_0, \omega_1) = \delta(q_0, \omega_2)$ if ω_1 and ω_2 belong to the same region word γ. It follows that $\omega_1 \sim_{\mathcal{A}} \omega_2$, according to Definition 3.3. □

Lemma 3.5. *If a timed language $\mathcal{L} = \mathcal{L}(\mathcal{A})$ for a DRTA $\mathcal{A}$, then for all $\omega_1, \omega_2 \in (\Sigma \times \mathbb{R}_{\ge 0})^*$, if $\omega_1 \sim_{\mathcal{A}} \omega_2$ then $\omega_1 \sim_{\mathcal{L}} \omega_2$.*

Proof. Suppose $\mathcal{L} = \mathcal{L}(\mathcal{A})$, therefore $\omega \in \mathcal{L} \Leftrightarrow \delta(q_0, \omega) \in F$, where $\omega \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, q_0 is the unique initial location in DRTA $\mathcal{A}$ and F is the finite set of accepting locations of $\mathcal{A}$. Then if $\omega_1 \sim_{\mathcal{A}} \omega_2$, we have $\delta(q_0, \omega_1) = \delta(q_0, \omega_2)$ according to Definition 3.3. That means the two runs ends in a same location in $\mathcal{A}$ after reading ω_1 and ω_2 respectively. Suppose $\omega' \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, clearly $\delta(q_0, \omega_1\omega') = \delta(q_0, \omega_2\omega')$. Therefore,

$$\omega_1\omega' \in \mathcal{L} \Leftrightarrow \omega_1\omega' \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \delta(q_0, \omega_1\omega') \in F \Leftrightarrow \delta(q_0, \omega_2\omega') \in F \Leftrightarrow \omega_2\omega' \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \omega_2\omega' \in \mathcal{L}.$$

Hence, it follows that $\omega_1 \sim_{\mathcal{L}} \omega_2$ according to Definition 3.2. □

Corollary 3.6. *If $\mathcal{L}$ is a real-time language, then $\sim_{\mathcal{L}}$ has a finite number of equivalence classes.*

Proof. Suppose $\mathcal{L}$ is a real-time language, then there exists a DRTA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$ according to Definition 2.2. Lemma 3.5 shows that $\sim_{\mathcal{L}}$ has at most as many equivalence classes as $\sim_{\mathcal{A}}$. According to Definition 3.3, each location in $\mathcal{A}$ represents an equivalence class which contains all timed words that can reach this location. Hence, the number of equivalence classes in $\sim_{\mathcal{L}}$ is at most the number of locations in $\mathcal{A}$, which is finite. Therefore, $\sim_{\mathcal{L}}$ has a finite number of equivalence classes and furthermore the number of equivalence classes is at most the number of locations in DRTA $\mathcal{A}$. □

Based on the above corollary and lemmas, we claim that there is a corresponding Myhill-Nerode theorem for real-time languages as follows.

Theorem 3.7 (Myhill-Nerode theorem for real-time languages). *$\mathcal{L}$ is a real-time language if and only if $\sim_{\mathcal{L}}$ has a finite number of equivalence classes which satisfy the following two conditions:*

1. *For all $\omega \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, $\sigma \in \Sigma$ and $\tau, \tau' \in \mathbb{R}_{\geq 0}$, if $[\![\tau']\!] = [\![\tau]\!]$, then $\omega \cdot (\sigma, \tau') \sim_{\mathcal{L}} \omega \cdot (\sigma, \tau)$;*
2. *There exists $\kappa \in \mathbb{N}$, such that for all $\sigma \in \Sigma$ and $\tau, \tau' \in \mathbb{R}_{\geq 0}$, if $\tau > \kappa$ and $\tau' > \kappa$ then $\omega \cdot (\sigma, \tau) \sim_{\mathcal{L}} \omega \cdot (\sigma, \tau')$.*

Furthermore there is a unique minimal (w.r.t. the number of locations) DRTA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$.

Proof. Corollary 3.6 shows that if $\mathcal{L}$ is a real-time language, then $\sim_{\mathcal{L}}$ has finitely many equivalence classes. We further prove that $\sim_{\mathcal{L}}$ satisfies the above two conditions. Suppose that a DRTA $\mathcal{A}$ recognizes the real-time language $\mathcal{L}$. For the first condition, we assume that $\mathcal{A}$ ends in a location q after reading $\omega \cdot (\sigma, \tau)$. By Lemma 3.4, for all $\tau' \in [\![\tau]\!]$, we have $\omega \cdot (\sigma, \tau') \sim_{\mathcal{A}} \omega \cdot (\sigma, \tau)$. Then by Lemma 3.5, we have $\omega \cdot (\sigma, \tau') \sim_{\mathcal{L}} \omega \cdot (\sigma, \tau)$. For the second condition, we let κ be the maximum constant appearing in the timed constraints of $\mathcal{A}$. By the definitions of DRTAs and regions, if $\tau > \kappa$ and $\tau' > \kappa$, then $[\![\tau]\!] = [\![\tau']\!] = (\kappa, \infty)$. By Lemma 3.4, $\omega \cdot (\sigma, \tau) \sim_{\mathcal{A}} \omega \cdot (\sigma, \tau')$. Then by Lemma 3.5, $\omega \cdot (\sigma, \tau) \sim_{\mathcal{L}} \omega \cdot (\sigma, \tau')$.

Then we prove the other direction. If $\sim_{\mathcal{L}}$ has finitely many equivalence classes which satisfy the two conditions, we can build a DRTA $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ which recognizes $\mathcal{L}$ as follows. Let $\mathbb{L}_0, \mathbb{L}_1, \cdots, \mathbb{L}_n$ be the disjoint equivalence classes of $\sim_{\mathcal{L}}$, such that $\epsilon \in \mathbb{L}_0$. Note that the union of the equivalence classes is $(\Sigma \times \mathbb{R}_{\geq 0})^*$. First, we build the set of locations. We set $Q = \{q_0, q_1, \cdots, q_n\}$, where each q_i represents the equivalence class $\mathbb{L}_i$. Then for building a transition $(q_i, \sigma, [\![\tau]\!], q_j)$, where $\sigma \in \Sigma$, $\tau \in \mathbb{R}_{\geq 0}$ and $0 \leq i, j \leq n$, we select a timed word ω from $\mathbb{L}_i$ and then find q_j such that $\omega \cdot (\sigma, \tau) \in \mathbb{L}_j$. Since $\sim_{\mathcal{L}}$ satisfies the two conditions, the number of regions $[\![\tau]\!]$ is finite. Therefore, Δ is a finite set. Finally, the initial location is q_0 and the set of accepting locations is $F = \{q_i \mid \mathbb{L}_i \subseteq \mathcal{L}\}$. By induction, we can prove $\delta(q_0, \omega) = q_i \in F \Leftrightarrow \omega \in \mathbb{L}_i \subseteq \mathcal{L}$, i.e. $\mathcal{L}(\mathcal{A}) = \mathcal{L}$.

Combining the two directions, $\mathcal{L}$ is a real-time language if and only if $\sim_{\mathcal{L}}$ has finitely many equivalence classes. Furthermore, we can build a unique minimal DRTA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. □

4 RESIDUAL REAL-TIME AUTOMATA

From Theorem 3.7, we know that there exists a unique minimal DRTA which can recognize a given real-time language. However, the same does not hold for NRTAs, which means that there is no unique minimal NRTA as the learning target for a given real-time language. Therefore, inspired by Denis et al.'s work on residual finite state automata (RFSAs) [12], we introduce a special kind of NRTAs, named residual real-time automata (RRTAs), which have nice properties for active learning, that is for a real-time language there is a unique minimal RRTA recognizing it. Then we can transform the problem of actively learning NRTAs to the problem of actively learning RRTAs, which will be the focus of Section 5.

Definition 4.1 (Residual real-time language). Let $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$ be a real-time language and ω be a timed word. The residual real-time language of $\mathcal{L}$ with regard to ω is defined by $\omega^{-1}\mathcal{L} = \{\omega' \in (\Sigma \times \mathbb{R}_{\geq 0})^* \mid \omega\omega' \in \mathcal{L}\}$. Let $Res(\mathcal{L})$ be the set of residual real-time languages of $\mathcal{L}$.

If $\mathcal{L}$ is recognized by an NRTA $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$, then $q \in \delta(Q_0, \omega) \Rightarrow \mathcal{L}_q \subseteq \omega^{-1}\mathcal{L}$. Based on Theorem 3.7 and the definition of residual real-time language, if RTA $\mathcal{A}' = (Q', \Sigma, \Delta', Q_0', F')$ is the unique minimal DRTA recognizing $\mathcal{L}$, then (1) there exists a unique $q' \in Q'$ such that $\mathcal{L}_{q'} = \omega^{-1}\mathcal{L}$ for each $\omega^{-1}\mathcal{L} \neq \emptyset$; (2) there exists a unique residual language $\omega^{-1}\mathcal{L}$ such that $\omega^{-1}\mathcal{L} = \mathcal{L}_{q'}$ for each $q' \in Q'$. In other words, for the minimal DRTA $\mathcal{A}'$, there is a bijection between the set of locations Q' and the set of residual real-time languages $Res(\mathcal{L}(\mathcal{A}')) = Res(\mathcal{L})$.

Definition 4.2 (Residual real-time automata). A *residual real-time automaton* is an NRTA $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ such that $\forall q \in Q, \exists \omega \in (\Sigma \times \mathbb{R}_{\geq 0})^* : \mathcal{L}_q = \omega^{-1}\mathcal{L}(\mathcal{A})$. Additionally, ω is called a *characterizing* timed word for q.

Immediately, we can see that a location q in an RRTA $\mathcal{A}$ represents a residual real-time language of $\mathcal{L}(\mathcal{A})$, but not every residual real-time language of $\mathcal{L}(\mathcal{A})$ corresponds to a single location. Suppose $\mathcal{A}'$ is the unique minimal DRTA for $\mathcal{L}(\mathcal{A})$, i.e. $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$, since there is a bijection between the set of locations of $\mathcal{A}'$ and the set of residual real-time languages of $\mathcal{L}(\mathcal{A}')$, the locations of RRTA $\mathcal{A}$ are a subset of the locations of the unique minimal DRTA $\mathcal{A}'$.

Definition 4.3 (Prime and composed residual real-time languages). Let $\mathcal{L}$ be a real-time language. A residual real-time language $\omega^{-1}\mathcal{L}$ is called *prime* if

$$\bigcup \{\omega'^{-1}\mathcal{L} \mid \omega'^{-1}\mathcal{L} \subsetneq \omega^{-1}\mathcal{L}\} \subsetneq \omega^{-1}\mathcal{L},$$

otherwise, $\omega^{-1}\mathcal{L}$ is called *composed*.

In other words, $\omega^{-1}\mathcal{L}$ is composed if there exist $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n \in Res(\mathcal{L}) \backslash \{\omega^{-1}\mathcal{L}\}$ such that $\omega^{-1}\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \cdots \cup \mathcal{L}_n$. Otherwise, it is prime. Additionally, the set of prime residual real-time languages of $\mathcal{L}$ is finite.

The following lemma shows that given an RRTA $\mathcal{A}$, each prime residual language $\omega^{-1}\mathcal{L}(\mathcal{A})$ corresponds to a location of $\mathcal{A}$. In other words, RRTA $\mathcal{A}$ has at least as many locations as the number of the prime residual real-time languages of $\mathcal{L}(\mathcal{A})$.

Lemma 4.4. *If $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ is an RRTA, then there exists a location $q \in Q$ such that $\mathcal{L}_q = \omega^{-1}\mathcal{L}(\mathcal{A})$ for each prime residual real-time language $\omega^{-1}\mathcal{L}(\mathcal{A})$.*

Proof. (Sketch) Given a prime residual $\omega^{-1}\mathcal{L}(\mathcal{A})$, suppose that $\delta(Q_0, \omega) = \{q_1, q_2, \cdots, q_m\}$ and let $\omega_1, \omega_2, \ldots, \omega_m$ be the characterizing timed words such that $\mathcal{L}_{q_i} = {\omega_i}^{-1}\mathcal{L}(\mathcal{A})$, where $1 \leq i \leq m$. Depending on Definition 4.1, $\omega^{-1}\mathcal{L}(\mathcal{A}) = \bigcup_{i=1}^{m} {\omega_i}^{-1}\mathcal{L}(\mathcal{A})$. As $\omega^{-1}\mathcal{L}(\mathcal{A})$ is a prime residual real-time language, according to Definition 4.3, there should exist a $\omega_i \in \{\omega_1, \omega_2, \ldots, \omega_m\}$ such that ${\omega_i}^{-1}\mathcal{L}(\mathcal{A}) = \omega^{-1}\mathcal{L}(\mathcal{A})$. Then we find a location $q = q_i \in Q$ such that $\mathcal{L}_{q_i} = \omega^{-1}\mathcal{L}(\mathcal{A})$. □

Definition 4.5 (Canonical residual real-time automata). Let $\mathcal{L}$ is a real-time language. An RRTA $\mathcal{A} = (Q, \Sigma, \Delta, Q_0, F)$ is the *canonical residual real-time automaton* (CRRTA) of $\mathcal{L}$, where

- $Q = \{\omega^{-1}\mathcal{L} \mid \omega^{-1}\mathcal{L} \text{ is prime}\}$ is the finite set of locations;
- Σ is the alphabet;
- $\Delta = \{({\omega_1}^{-1}\mathcal{L}, \sigma, [\![\tau]\!], {\omega_2}^{-1}\mathcal{L}) \mid \sigma \in \Sigma \wedge \tau \in \mathbb{R}_{\geq 0} \wedge {\omega_1}^{-1}\mathcal{L}, {\omega_2}^{-1}\mathcal{L} \in Q \wedge {\omega_2}^{-1}\mathcal{L} \subseteq (\omega_1 \cdot (\sigma, \tau))^{-1}\mathcal{L}\}$ is the transition relation;
- $Q_0 = \{\omega^{-1}\mathcal{L} \mid \omega^{-1}\mathcal{L} \subseteq \mathcal{L} \wedge \omega^{-1}\mathcal{L} \in Q\}$ is the set of initial locations;
- $F = \{\omega^{-1}\mathcal{L} \mid \epsilon \in \omega^{-1}\mathcal{L} \wedge \omega^{-1}\mathcal{L} \in Q\}$ is the set of accepting locations.

According to Theorem 3.7, there exists $\kappa \in \mathbb{N}$ as the maximum value appearing in the timed constraints. Therefore Δ is finite and the CRRTA of $\mathcal{L}$ is well-defined. The CRRTA is the RRTA with the minimum number of locations (reduced) and maximum number of transitions (saturated).

Theorem 4.6. *The canonical residual real-time automaton $\mathcal{A}$ of a real-time language $\mathcal{L}$ is the minimal (w.r.t. the number of locations) RRTA which recognizes $\mathcal{L}$.*

Proof. (sketch) It's not hard to prove that the CRRTA $\mathcal{A}$ is an RRTA. Then Lemma 4.4 shows that it has a minimal number of locations. □

Note that the CRRTA $\mathcal{A}$ has a lot of transitions, according to Definition 4.5. One way to reduce the number of transitions is to merge those with the same source location, target location and action, and whose region guards are adjacent. For example, if $(q, \sigma, (1, 2), q')$ and $(q, \sigma, [2, 2], q')$ are two transitions in Δ, then we can merge them into a new transition $(q, \sigma, (1, 2], q')$. Such operations do not change the number of locations and the recognized real-time language of $\mathcal{A}$. If $\mathcal{A}'$ is the automaton transformed from a CRRTA $\mathcal{A}$ through the above operation, $\mathcal{A}'$ is seen as the CRRTA for the same real-time language hereafter.

5 LEARNING RESIDUAL REAL-TIME AUTOMATA

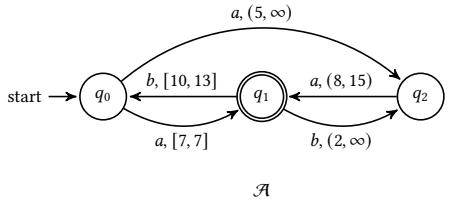
Based on the canonical property of RRTAs, we can transform the problem of actively learning NRTAs to the problem of actively learning minimal RRTAs. In this section, we describe the learning algorithm, analyse its complexity and prove its correctness.

We first describe the settings for active learning of real-time languages in general. Following Angluin's MAT framework, there exists a teacher who knows the target real-time language $\mathcal{L}$ and answers two kinds of queries from a learner. For a membership query, the learner asks whether a timed word ω is in the language $\mathcal{L}$ or not. The teacher can answer yes or no. The learner collects the results of membership queries in an observation table. For an equivalence query, the teacher receives an NRTA $\mathcal{A}$ from the learner as a hypothesis for $\mathcal{L}$. The teacher answers whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. If not, the teacher also returns a timed word as a counterexample which distinguishes $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}$. In what follows, we present the details of the learning algorithm.

5.1 Membership query and real-time observation table

In order to gather enough information to construct a hypothesis, the learner makes membership queries like "Is the timed word ω in $\mathcal{L}$?". In practice, a membership query is often conducted by testing. In theory, we assume that the teacher has an oracle to answer membership queries, i.e., $\mathsf{MQ} : (\Sigma \times \mathbb{R}_{\geq 0})^* \to \{+, -\}$. Given an RTA recognizing $\mathcal{L}$, the teacher gives a positive answer if there is a run ρ ending in an accepting location after reading ω. The results of membership queries are collected in a real-time observation table $\mathcal{T}$ as follows.

Definition 5.1 (Real-time observation table). A real-time observation table is a 6-tuple $\mathcal{T} = (\Sigma, \Xi, S, R, E, f)$, where Σ is the alphabet, $\Xi = \Sigma \times \mathbb{R}_{\geq 0}$ is the set of all timed actions, $S, R, E \subseteq \Xi^*$, S



$\mathcal{A}$

$\mathcal{T}$	ϵ	$(a, 8.1)$	$(a, 8.1)(a, 8.1)$	$(a, 15)$	
ϵ	−	−	+	−	✓
$(a, 5.1)$	−	+	−	−	✓
$(a, 0)$	−	−	−	−	✓
$(a, 5.1)(a, 8.1)$	+	−	−	−	✓
$(b, 0)$	−	−	−	−	✓
$(a, 5.1)(a, 0)$	−	−	−	−	✓
$(a, 5.1)(b, 0)$	−	−	−	−	✓
⋯			⋯		⋯
$(a, 7)$	+	+	−	−	
⋯			⋯		⋯

Fig. 2. Left: an example of an NRTA. Right: an example of real-time observation table in which the prime prefixes are labelled by ✓. Some rows of the table are omitted.

is called the set of prefixes, R the set of extended prefixes and E the set of suffixes, respectively. Specifically,

- S and R are disjoint, i.e. $S \cap R = \emptyset$;
- $S \cup R$ is prefix-closed and E is suffix-closed;
- The empty word ϵ is both a prefix and a suffix, i.e. $\epsilon \in S \cup R$ and $\epsilon \in E$;
- $f : (S \cup R) \cdot E \to \{+, -\}$ is a mapping function such that for every timed word (prefix) $\omega \in S \cup R$ and every timed word (suffix) $e \in E$, $f(\omega \cdot e) = +$ if the timed word $\omega \cdot e \in \mathcal{L}$, i.e. $\mathrm{MQ}(\omega e) = +$ and $f(\omega \cdot e) = -$ otherwise, i.e. $\mathrm{MQ}(\omega e) = -$.

Given a table $\mathcal{T}$, we define a function $val : S \cup R \to (E \to \{+, -\})$ mapping every prefix $\omega \in S \cup R$ to a value vector indexed by the suffix $e \in E$, in which each value is defined by $f(\omega \cdot e)$. Intuitively, the vectors denote different locations in a hypothesis automaton, and the suffixes are used to distinguish the locations. Here we define a function $row : S \cup R \to 2^E$ by $row(\omega) = \{e \in E \mid f(\omega \cdot e) = +\}$ for each $\omega \in S \cup R$. Hence, considering the definitions of residual real-time languages and RRTAs in Section 4, we find that $row(\omega)$ represents a subset of the residual real-time language $\omega^{-1}\mathcal{L}$ indicated by the prefix $\omega \in S \cup R$.

Therefore, in order to indicate the prime and composed residual real-time languages of $\mathcal{L}$ using the prefixes in $S \cup R$, we define prime and composed prefixes as follows.

Definition 5.2 (Prime and composed prefixes). Given a table $\mathcal{T} = (\Sigma, \Xi, S, R, E, f)$, a prefix $\omega \in S \cup R$ is *prime* if

$$\bigcup \{row(\omega') \mid row(\omega') \subsetneq row(\omega)\} \subsetneq row(\omega)$$

with $\omega' \in S \cup R$, otherwise *composed*.

In other words, a prefix $\omega \in S \cup R$ is composed if there exist $\omega_1, \omega_2, \ldots, \omega_n \in (S \cup R) \backslash \{\omega\}$ such that $row(\omega) = row(\omega_1) \cup row(\omega_2) \cup \cdots \cup row(\omega_n)$. Otherwise, the prefix ω is prime. *It's worth noting that given a prime prefix $\omega \in S \cup R$ there may exist prime prefixes $s \in S$ such that $row(s) \subsetneq row(\omega)$.*

Example 5.3. Fig. 2 shows another NRTA $\mathcal{A}$ and an instance of the real-time observation table $\mathcal{T}$ during the learning process for $\mathcal{A}$. We use the example to illustrate the definitions about real-time observation tables. The prefix set S is $\{\epsilon, (a, 5.1), (a, 0), (a, 5.1)(a, 8.1)\}$, the extended prefix set R is $\{(b, 0), (a, 5.1)(a, 0), (a, 5.1)(b, 0), (a, 7), \cdots\}$ and the suffix set E is $\{\epsilon, (a, 8.1), (a, 8.1)(a, 8.1), (a, 15)\}$. Given a timed word $\omega \cdot e = (a, 5.1) \cdot (a, 8.1)$, we have $f(\omega \cdot e) = +$, since it is accepted by $\mathcal{A}$. For the functions val and row, we have $val((a, 5.1)) = \{-, +, -, -\}$ and $row = \{(a, 8.1)\}$. The prefix $(a, 7)$ is a composed prefix since $row((a, 7)) = \{\epsilon, (a, 8.1)\} = row((a, 5.1)(a, 8.1)) \cup row((a, 5.1))$. As an illustration of the note in the previous paragraph, although $(a, 5.1)$ is a prime prefix, there exists a prime prefix $(a, 0) \in S$ such that $row((a, 0)) \subsetneq row((a, 5.1))$.

The basic idea of the learning algorithm is to find all prime prefixes. Based on Definition 4.3, there exists a set of timed words to distinguish prime residual real-time languages from each other. Thus we try to find the suffix set E containing distinguishing words. Before constructing a hypothesis from a table, the learner has to ensure that the table satisfies the following readiness conditions:

- **Reduced**: $\forall s \in S : s$ is prime, and $\forall s, s' \in S : s \neq s' \Rightarrow val(s) \neq val(s')$.
- **Closed**: $\forall r \in R : row(r) = \bigcup \{row(s) \mid s \in S \wedge row(s) \subseteq row(r)\}$.
- **Consistent**: $\forall \omega, \omega' \in S \cup R : row(\omega) \subseteq row(\omega') \Rightarrow row(\omega \cdot \boldsymbol{\sigma}) \subseteq row(\omega' \cdot \boldsymbol{\sigma})$ if $\omega \cdot \boldsymbol{\sigma}, \omega' \cdot \boldsymbol{\sigma} \in S \cup R$, where $\boldsymbol{\sigma} \in \Xi = \Sigma \times \mathbb{R}_{\geq 0}$.
- **Evidence-closed**: $\forall s \in S, e \in E : s \cdot e \in S \cup R$.
- **Distinctness**: $\forall \omega \in S \cup R, \boldsymbol{\sigma} \in \Xi = \Sigma \times \mathbb{R}_{\geq 0}$: $\omega \cdot \boldsymbol{\sigma} \in S \cup R \Rightarrow s_i \cdot \boldsymbol{\sigma} \in S \cup R$, where $s_i \in \{s \in S \mid row(s) \subseteq row(\omega)\}$.

Since S is the set of current prime prefixes, a reduced table helps to set a bijection between S and prime residuals, which corresponds to Definition 4.5. A closed table ensures that for every current composed residuals indicated by $r \in R$ is composed by an union of prime residuals indicated by current prime prefixes, which corresponds to Definition 4.3 and 5.2. Given a table, if $row(\omega) \subseteq row(\omega')$, it shows a relation that the residual language indicated by ω is a subset of the residual language indicated by ω'. Hence if the table is consistent, after performing the same timed action $\boldsymbol{\sigma}$, the relation is still maintained for the timed words $\omega \cdot \boldsymbol{\sigma}$ and $\omega' \cdot \boldsymbol{\sigma}$. The above two readiness conditions are direct extensions of those in [8]. The evidence-closed condition is added and serves the same function as in [4, 5, 16]. The last condition, distinctness, is unique for this paper, and is needed to deal with nondeterministic behavior caused by timing information. Given a prefix ω, it represents a virtual location which combines the locations indicated by prime prefixes $s_i \in \{s \in S \mid row(s) \subseteq row(\omega)\}$. If a table contains the prefixes $s_i \cdot \boldsymbol{\sigma}$, it will make the residual and location indicated by s_i distinct. Such condition prevents the hypothesis receiving a repeating counterexample and therefore ensures the termination of our learning algorithm. A real-time observation table is *prepared* if it satisfies the above conditions. To achieve it, we introduce the following operations.

Making $\mathcal{T}$ reduced. If $\mathcal{T}$ is not reduced, there are two cases. If there is an $s \in S$ which is not prime, there exists a set $\{s_1, s_2, \ldots, s_n\} \in S \backslash \{s\}$ such that $row(s) = row(s_1) \cup row(s_2) \cup \cdots \cup row(s_n)$. In this case we fix the table by moving s to R. The second part of the condition, that there are no two rows in S which are the same, is guaranteed by the other table operations and the process of handling counterexamples in Section 5.3.

Making $\mathcal{T}$ closed. If $\mathcal{T}$ is not closed, we find a row $r \in R$ such that $\bigcup \{row(s) | row(s) \subseteq row(r)\} \neq row(r)$, where $s \in S$, then move r from R to S. It means that we find a new location. Additionally, for each action $\sigma \in \Sigma$, we add a new row $r \cdot (\sigma, 0)$ into R and fill the table by performing membership queries $\mathrm{MQ}(r \cdot (\sigma, 0) \cdot e)$ for every suffix $e \in E$. Such an operation is important since it guarantees that at every location all actions in Σ are enabled. Comparing with the L^* and NL^* algorithms, we do not add timed words $r \cdot (\sigma, \tau)$ for all $\tau \in \mathbb{R}_{\geq 0}$ to the table since the number of such timed words is infinite. The partition function in Section 5.2 will handle all clock valuations in $\mathbb{R}_{\geq 0}$.

Making $\mathcal{T}$ consistent. If $\mathcal{T}$ is not consistent, it implies that there exist at least two rows $\omega, \omega' \in S \cup R$, such that $\omega \cdot \boldsymbol{\sigma}, \omega' \cdot \boldsymbol{\sigma} \in S \cup R$ for some $\boldsymbol{\sigma} \in \Sigma \times \mathbb{R}_{\geq 0}$, with $row(\omega) \subseteq row(\omega')$, but $row(\omega \cdot \boldsymbol{\sigma}) \nsubseteq row(\omega' \cdot \boldsymbol{\sigma})$. Then we find a suffix $e \in E$ such that $f(\omega \cdot \boldsymbol{\sigma} \cdot e) = +$ and $f(\omega' \cdot \boldsymbol{\sigma} \cdot e) = -$. The suffix e can also be found using $row(\omega) \backslash (row(\omega) \cap row(\omega'))$. One inconsistency can be fixed by adding a new suffix $\boldsymbol{\sigma} \cdot e$ to E. Afterwards, the table is filled via membership queries.

$\mathcal{T}_2$	ϵ	$(a,8.1)$
ϵ	−	−
$(a,0)$	−	−
$(b,0)$	−	−
$(a,5.1)$	−	+
$(a,5.1)(a,8.1)$	+	−

$\mathcal{T}_3$	ϵ	$(a,8.1)$
ϵ	−	−
$(a,5.1)$	−	+
$(a,0)$	−	−
$(b,0)$	−	−
$(a,5.1)(a,8.1)$	+	−
$(a,5.1)(a,0)$	−	−
$(a,5.1)(b,0)$	−	−

$\mathcal{T}_4$	ϵ	$(a,8.1)$
ϵ	−	−
$(a,5.1)$	−	+
$(a,0)$	−	−
$(b,0)$	−	−
$(a,5.1)(a,8.1)$	+	−
$(a,5.1)(a,0)$	−	−
$(a,5.1)(b,0)$	−	−
$(a,8.1)$	−	+

$\mathcal{T}_5$	ϵ	$(a,8.1)$	$(a,8.1)(a,8.1)$
ϵ	−	−	+
$(a,5.1)$	−	+	−
$(a,0)$	−	−	−
$(b,0)$	−	−	−
$(a,5.1)(a,8.1)$	+	−	−
$(a,5.1)(a,0)$	−	−	−
$(a,5.1)(b,0)$	−	−	−
$(a,8.1)$	−	+	−

Fig. 3. The table instances used in Example 5.4 to illustrate the readiness conditions.

Making $\mathcal{T}$ evidence-closed. If $\mathcal{T}$ is not evidence-closed, then we can find $s \in S$ and $e \in E$ with $s \cdot e \notin S \cup R$ and put all prefixes of $s \cdot e$ in R except for those already in $S \cup R$. Similarly, we need to fill the table through membership queries.

Making $\mathcal{T}$ distinct. If $\mathcal{T}$ is not distinct, then we can find a row $\omega \in S \cup R$ with $\omega \cdot \boldsymbol{\sigma} \in S \cup R$ for some $\boldsymbol{\sigma} \in \Sigma \times \mathbb{R}_{\geq 0}$ but not all $s_i \cdot \boldsymbol{\sigma}$ in $S \cup R$, where $s_i \in \{s \in S \mid row(s) \subseteq row(\omega)\}$. Then the table can be fixed by adding all such $s_i \cdot \boldsymbol{\sigma}$ to R. Similarly, the table needs to be filled via membership queries. Note that we need to perform this operation no matter if ω is prime or composed.

A table may need several rounds of the above operations before being prepared (cf. Algorithm 1). The following example illustrates the readiness conditions and the corresponding operations.

Example 5.4. As shown in Fig. 3, there are four table instances $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$ and $\mathcal{T}_5$.

$\mathcal{T}_2$ is not closed since there exists a row $(a,5.1) \in R$ such that $\bigcup \{row(\epsilon)\} = \emptyset \neq row((a,5.1))$. In order to repair it, we move the row $(a,5.1)$ from R to S and then add $(a,5.1) \cdot (a,0)$ and $(a,5.1) \cdot (b,0)$ to R. After making membership queries, we get $\mathcal{T}_3$.

$\mathcal{T}_3$ is not distinct. It can be shown as follows. Let $\omega \cdot \boldsymbol{\sigma} = (a,5.1) \cdot (a,8.1)$. Then we can build the set $\{\epsilon, (a,5.1)\}$ since $row(\epsilon) \subseteq row((a,5.1))$ and $row((a,5.1)) \subseteq row((a,5.1))$. However, $\epsilon \cdot (a,8.1)$ is not in $S \cup R$. Hence, we add $\epsilon \cdot (a,8.1)$ into R to repair it. This results in table $\mathcal{T}_4$. Actually, since $\mathcal{T}_3$ is also not evidence-closed, $\epsilon \cdot (a,8.1)$ can also be added into R in the process of making $\mathcal{T}_3$ evidence-closed. It depends on the checking order of the conditions.

$\mathcal{T}_4$ is not consistent, since $row(\epsilon) \subseteq row((a,5.1))$ but $row(\epsilon \cdot (a,8.1)) \not\subseteq row((a,5.1) \cdot (a,8.1))$. To repair it, we find the suffix $(a,8.1) \in E$ with $f(\epsilon \cdot (a,5.1) \cdot (a,8.1)) = +$ and $f((a,5.1) \cdot (a,8.1) \cdot (a,5.1)) = -$, then add $(a,8.1) \cdot (a,8.1)$ to E as a new suffix. This results in table $\mathcal{T}_5$. It is not hard to find that $\mathcal{T}_5$ is not closed and we can move $(a,5.1)(a,8.1)$ from R to S to repair it.

5.2 Constructing a hypothesis from a prepared table $\mathcal{T}$

Once the learner obtains a prepared table $\mathcal{T}$, a hypothesis NRTA can be constructed in two steps. The learner first constructs an NFA M from $\mathcal{T}$, and then transforms it to an NRTA $\mathcal{H}$ as the current hypothesis. We describe each of the two steps in turn.

Construction of NFA M. Given a prepared real-time observation table $\mathcal{T} = (\Sigma, \Xi, S, R, E, f)$, the learner builds an NFA $M = (Q_M, \Sigma_M, \Delta_M, Q_M^0, F_M)$ as follows:

- the finite set of locations $Q_M = \{q_{val(s)} \mid s \in S\}$;
- the abstract alphabet $\Sigma_M = \{\boldsymbol{\sigma} \in \Xi \mid \omega \cdot \boldsymbol{\sigma} \in S \cup R \wedge row(\omega) \in \{row(s) \mid s \in S\}\}$;
- the transition relation $\Delta_M = \{(q_{val(\omega)}, \boldsymbol{\sigma}, q_{val(s')}) \mid \omega \cdot \boldsymbol{\sigma} \in S \cup R \wedge row(\omega) \in \{row(s) \mid s \in S\} \wedge row(s') \subseteq row(\omega \cdot \boldsymbol{\sigma}) \wedge s' \in S\}$; (The corresponding transition function is denoted as $\hat{\delta}$);
- the set of initial locations $Q_M^0 = \{q_{val(s)} \mid row(s) \subseteq row(\epsilon)$ for $s \in S \wedge \epsilon \in S \cup R\}$;
- the set of accepting locations $F_M = \{q_{val(s)} \mid f(s \cdot \epsilon) = +$ for $s \in S \wedge \epsilon \in E\}$.

If $\mathcal{T}$ is a prepared table, the NFA M is well-defined. For the set of locations, we let each current prime prefix in S to represent a location. The transitions are built as follows. For each timed

word $\omega \cdot \boldsymbol{\sigma} \in S \cup R$, since $\mathcal{T}$ is prefix-closed, we know ω is also in $S \cup R$. If ω is prime, then $row(\omega) \in \{row(s) \mid s \in S\}$, and we add a transition $(q_{val(\omega)}, \boldsymbol{\sigma}, q_{val(s')})$ for each $s' \in S$ with $row(s') \subseteq row(\omega \cdot \boldsymbol{\sigma})$. If ω is composed, we have $row(\omega) \notin \{row(s) \mid s \in S\}$, so no transition comes directly from ω. However, since $\mathcal{T}$ is distinct, if $row(\omega)$ decomposes as $row(s_1) \cup row(s_2) \cup \cdots \cup row(s_n)$, where each $s_i \in S$, then we have added all $s_i \cdot \boldsymbol{\sigma}$ for $1 \leq i \leq n$ into R, and it suffices to build transitions using $row(s_i)$. The set Σ_M collects all timed actions $\boldsymbol{\sigma}$ which can trigger transitions, viewing a timed action $\boldsymbol{\sigma}$ as an abstract action in NFA M. For initial locations, the formula $Q_M^0 = \{q_{val(s)} \mid row(s) \subseteq row(\epsilon) \text{ for } s \in S \wedge \epsilon \in S \cup R\}$ means that the virtual initial location (reached by ϵ) is composed from several actual locations. Hence, the prefix ϵ is not in S if and only if it is a composed prefix. The set of accepting locations collects all locations indicated by the prime prefixes which are in $\mathcal{L}$.

Definition 5.5 (Compatibility). Let $\mathcal{T} = (\Sigma, \Xi, S, R, E, f)$ be a prepared table, and $M = (Q_M, \Sigma_M, \Delta_M, Q_M^0, F_M)$ be an NFA. We say M is compatible with $\mathcal{T}$ if for every timed word $\omega \cdot e \in (S \cup R) \cdot E$, M accepts $\omega \cdot e$ if and only if $f(\omega \cdot e) = +$.

According to the learning algorithms [4, 5, 16] for deterministic automata, we need to prove that NFA M is compatible with table $\mathcal{T}$. However, as pointed out in [8], this property does not necessarily hold for the constructed NFA in every round. Instead, it is shown in [8] that M satisfies several weaker properties, which is still sufficient for proving correctness of the algorithm. Further it is shown that if M is compatible with $\mathcal{T}$, then M is a canonical RFSA. Compared to [8], we defined several extra readiness conditions and modified the process for handling counterexamples, so the statement and proofs of the lemmas are slightly modified. The lemmas are as follows.

Lemma 5.6 ([8, Lemma 2]). *Let $\mathcal{T} = (\Sigma, \Xi, S, R, E, f)$ be a prepared table and $M = (Q_M, \Sigma_M, \Delta_M, Q_M^0, F_M)$ be the constructed NFA. For every timed word $s \cdot e \in S \cdot E$, $f(s \cdot e) = +$ iff $e \in L_{q_{val(s)}}$, where $L_{q_{val(s)}}$ is the language of location $q_{val(s)}$, i.e. $\hat{\delta}(q_{val(s)}, e) \cap F_M \neq \emptyset$. Moreover, $f(\epsilon \cdot e) = +$ iff $e \in L(M)$, where $L(M)$ is the recognized language of NFA M.*

Proof. Suppose $e = \epsilon$, we have $f(s \cdot e) = f(s \cdot \epsilon) = +$ iff $q_{val(s)} \in F_M$ by the definition of F_M. Hence, we have $f(s \cdot \epsilon) = +$ iff $\epsilon \in L_{q_{val(s)}}$.

Now suppose $e = \boldsymbol{\sigma} \cdot e'$. Since $\mathcal{T}$ is evidence-closed, we have $s \cdot \boldsymbol{\sigma} \in S \cup R$. Since E is suffix-closed, we have $e' \in E$. By the definition of Δ_M, there exist transitions with source location $q_{val(s)}$ and action $\boldsymbol{\sigma}$, so that $\boldsymbol{\sigma} \in \Sigma_M$. We prove the lemma by induction on the length of $e \in E$.

If $f(s \cdot e) = +$, then $f((s \cdot \boldsymbol{\sigma}) \cdot e') = f(s \cdot (\boldsymbol{\sigma} \cdot e')) = +$. We wish to find $s' \in S$ such that $f(s' \cdot e') = +$. If $s \cdot \boldsymbol{\sigma} \in S$, it suffices to set $s' = s \cdot \boldsymbol{\sigma}$. Otherwise, we have $s \cdot \boldsymbol{\sigma} \in R$. Since $f((s \cdot \boldsymbol{\sigma}) \cdot e') = +$, there exists at least one $s' \in S$ such that $f(s' \cdot e') = +$ and $row(s') \subseteq row(s \cdot \boldsymbol{\sigma})$, and it suffices to take that s'. By induction hypothesis, $e' \in L_{q_{val(s')}}$. By the definition of Δ_M, there is a transition from $q_{val(s)}$ to $q_{val(s')}$ with action $\boldsymbol{\sigma}$. Hence, the suffix $e = \boldsymbol{\sigma} \cdot e'$ is in $L_{q_{val(s)}}$. In particular, if $\epsilon \in S$ and $f(\epsilon \cdot e) = +$, we have shown that $e \in L_{q_{val}(\epsilon)}$. By the definition of Q_M^0, if $\epsilon \in S$, $L_{q_{val}(\epsilon)} \subseteq L(M)$ then $e \in L(M)$. If $\epsilon \in R$, by the definition of Q_M^0, every s such that $q_{val(s)} \in Q_M^0$ satisfies $row(s) \subseteq row(\epsilon)$. Additionally, since $\mathcal{T}$ is closed, then we can find at least one $s \in S$ such that $q_{val(s)} \in Q_M^0$ and $f(s \cdot e) = +$. It follows $e \in L(M)$.

For the inverse direction, suppose $e = \boldsymbol{\sigma} \cdot e'$ and $f(s \cdot e) = -$. We have $f((s \cdot \boldsymbol{\sigma}) \cdot e') = f(s \cdot (\boldsymbol{\sigma} \cdot e')) = -$. We wish to show $f(s' \cdot e') = -$ for every $s' \in S$ satisfying $row(s') \subseteq row(s \cdot \boldsymbol{\sigma})$. Since $\mathcal{T}$ is closed, we have $row(s \cdot \boldsymbol{\sigma}) = \bigcup \{row(s') \mid s' \in S \wedge row(s') \subseteq row(s \cdot \boldsymbol{\sigma})\}$. Since $f((s \cdot \boldsymbol{\sigma}) \cdot e') = -$ and $row(s') \subseteq row(s \cdot \boldsymbol{\sigma})$, then $f(s' \cdot e') = -$ for every such s'. By the induction hypothesis, $e' \notin L_{q_{val(s')}}$ for all $s' \in S$ satisfying $row(s') \subseteq row(s \cdot \boldsymbol{\sigma})$. By the definition of Δ_M, there exists no transition from $q_{val(s)}$ to $q_{val(s')}$ with action $\boldsymbol{\sigma}$. Hence, the suffix $e = \boldsymbol{\sigma} \cdot e'$ is not in $L_{q_{val(s)}}$. Moreover, we can prove if $f(\epsilon \cdot e) = -$ then $e \notin L(M)$.

Combining the two directions, we have $f(s\cdot e)=+$ iff $e\in L_{q_{val(s)}}$ and $f(\epsilon\cdot e)=+$ iff $e\in L(M)$. □

Lemma 5.7 ([8, Lemma 3]). *Let $\mathcal{T}=(\Sigma,\Xi,S,R,E,f)$ be a prepared table and $M=(Q_M,\Sigma_M,\Delta_M,Q^0_M,F_M)$ be the constructed NFA. For $s,s'\in S$, $row(s)\subseteq row(s')$ if and only if $L_{q_{val(s)}}\subseteq L_{q_{val(s')}}$.*

Proof. If $row(s)\subseteq row(s')$ and suppose $\omega\in L_{q_{val(s)}}$, we need to prove $\omega\in L_{q_{val(s')}}$. If $\omega=\epsilon$, we have $f(s\cdot\epsilon)=+$. Since $row(s)\subseteq row(s')$, $f(s'\cdot\epsilon)=+$. By the definition of F_M, $q_{val(s')}\in F_M$. Hence, $\omega=\epsilon\in L_{q_{val(s')}}$. If $\omega=\boldsymbol{\sigma}\cdot\omega'$, where $\boldsymbol{\sigma}\in\Sigma_M$ and $\omega\in L_{q_{val(s)}}$, then $\hat{\delta}(q_{val(s)},\boldsymbol{\sigma}\cdot\omega')\cap F_M\neq\emptyset$. Hence there is $s''\in S$ with $q_{val(s'')}\in\hat{\delta}(q_{val(s)},\boldsymbol{\sigma})$ and $\hat{\delta}(q_{val(s'')},\omega')\cap F_M\neq\emptyset$. Based on the definition of Δ_M, $row(s'')\subseteq row(s\cdot\boldsymbol{\sigma})$. Since $\mathcal{T}$ is consistent, we also have $row(s\cdot\boldsymbol{\sigma})\subseteq row(s'\cdot\boldsymbol{\sigma})$. Therefore, $row(s'')\subseteq row(s'\cdot\boldsymbol{\sigma})$. Thus, $s''\in\hat{\delta}(q_{val(s')},\boldsymbol{\sigma})$, which implies $\omega=\boldsymbol{\sigma}\cdot\omega'\in L_{q_{val(s')}}$.

For the inverse direction, suppose $row(s)\not\subseteq row(s')$, then there exists $e\in E$ such that $f(s\cdot e)=+$ while $f(s'\cdot e)=-$. By Lemma 5.6, $e\in L_{q_{val(s)}}$ and $e\notin L_{q_{val(s')}}$. Therefore, $L_{q_{val(s)}}\not\subseteq L_{q_{val(s')}}$. □

Lemma 5.8 ([8, Lemma 4]). *Let $\mathcal{T}=(\Sigma,\Xi,S,R,E,f)$ be a prepared table and suppose the constructed NFA $M=(Q_M,\Sigma_M,\Delta_M,Q^0_M,F_M)$ is compatible with $\mathcal{T}$. For every $\omega\in S\cup R$, if ω is prime, then $q_{val(\omega)}\in\hat{\delta}(Q^0_M,\omega)$.*

Proof. Suppose $q_{val(\omega)}\notin\hat{\delta}(Q^0_M,\omega)$. Since ω is prime, then there is a prefix $s\in S$ with $row(s)=row(\omega)$. Thus, $q_{val(s)}\notin\hat{\delta}(Q^0_M,s)$. For all $s'\in S$ satisfying $q_{val(s')}\in\hat{\delta}(Q^0_M,\omega)=\hat{\delta}(Q^0_M,s)$, we have $row(s')\subseteq row(s)$. By Lemma 5.7, $L_{q_{val(s')}}\subseteq L_{q_{val(s)}}$. As $q_{val(s)}\notin\hat{\delta}(Q^0_M,s)$, there exists $e\in E$ such that $f(s\cdot e)=+$ and for all $q_{val(s')}\in\hat{\delta}(Q^0_M,s)$, $f(s'\cdot e)=-$ which implies $e\notin L_{q_{val(s')}}$. Thus, $e\notin L_{q_{val(s)}}$ and then $s\cdot e\notin L(M)$ while $f(s\cdot e)=+$. This contradicts the assumption that M is compatible with $\mathcal{T}$. Hence, we have $q_{val(\omega)}\in\hat{\delta}(Q^0_M,\omega)$. □

After constructing an NFA M from a prepared table, the learner can transform it to a hypothesis NRTA $\mathcal{H}=(Q,\Sigma,\Delta,Q_0,F)$ by using the following partition function.

Definition 5.9 (Partition function [4]). Given a list of clock valuations $\ell=\tau_0,\tau_1,\cdots,\tau_n$ with $0=\tau_0<\tau_1\cdots<\tau_n$, and $\lfloor\tau_i\rfloor\neq\lfloor\tau_j\rfloor$ if $\tau_i,\tau_j\in\mathbb{R}_{\geq 0}\setminus\mathbb{N}$ and $i\neq j$ for all $1\leq i,j\leq n$, let $\tau_{n+1}=\infty$, then a partition function $P(\cdot)$ mapping ℓ to a set of intervals $\{I_0,I_1,\ldots,I_n\}$, which is a partition of $\mathbb{R}_{\geq 0}$, is defined as

$$I_i=\begin{cases}[\tau_i,\tau_{i+1}), & \text{if } \tau_i\in\mathbb{N}\wedge\tau_{i+1}\in\mathbb{N};\\ (\lfloor\tau_i\rfloor,\tau_{i+1}), & \text{if } \tau_i\in\mathbb{R}_{\geq 0}\setminus\mathbb{N}\wedge\tau_{i+1}\in\mathbb{N};\\ [\tau_i,\lfloor\tau_{i+1}\rfloor], & \text{if } \tau_i\in\mathbb{N}\wedge\tau_{i+1}\in\mathbb{R}_{\geq 0}\setminus\mathbb{N};\\ (\lfloor\tau_i\rfloor,\lfloor\tau_{i+1}\rfloor], & \text{if } \tau_i\in\mathbb{R}_{\geq 0}\setminus\mathbb{N}\wedge\tau_{i+1}\in\mathbb{R}_{\geq 0}\setminus\mathbb{N}.\end{cases}$$

Construction of hypothesis $\mathcal{H}$. The hypothesis NRTA $\mathcal{H}=(Q,\Sigma,\Delta,Q_0,F)$ can be constructed by the following steps. First, let $Q=Q_M$, $Q_0=Q^0_M$, $F=F_M$ and let Σ be the given alphabet as in $\mathcal{T}$. For every location $q\in Q_M$ and every action $\sigma\in\Sigma$, we find a set of clock valuations $\Psi_{q,\sigma}=\{\tau\mid(q,(\sigma,\tau),q')\in\Delta_M\}$. Then we apply the partition function P to the result of sorting the elements in $\Psi_{q,\sigma}$ to get k intervals, written as $I_1,\cdots,I_k$, where $k=|\Psi_{q,\sigma}|$. These intervals satisfy $\tau_i\in I_i$ for any $1\leq i\leq k$. Consequently, for every $(q,(\sigma,\tau_i),q')\in\Delta_M$, a fresh transition (q,σ,I_i,q') is added to Δ. The construction build a bijection between Δ_M and Δ.

Since NFA M is not always compatible with $\mathcal{T}$, the constructed NRTA $\mathcal{H}$ is not always compatible with $\mathcal{T}$ either, i.e., it does not hold that for all $\omega\cdot e\in(S\cup R)\cdot E$, $\mathcal{H}$ accepts $\omega\cdot e$ if and only if $f(\omega\cdot e)=+$. However, the following lemma shows that the two automata $\mathcal{H}$ and M are compatible with each other w.r.t. all $\omega\in S\cup R$ which can be generated by M.

	$\mathcal{T}$	ϵ	$\cdots$
q_0	ϵ	–	$\cdots$
q_1	$(b,3)$	–	$\cdots$
q_2	$(a,1)$	–	$\cdots$
q_3	$(b,3)(a,1)$	–	$\cdots$
q_4	$\cdots$	–	$\cdots$
q_5	$\cdots$	+	$\cdots$
$\cdots$	$\cdots$		$\cdots$
$\cdots$	$\cdots$		$\cdots$
q_4	$(a,1)(b,7)$	–	$\cdots$
q_5	$(b,3)(a,1)(b,4)$	+	$\cdots$
$\cdots$	$\cdots$		$\cdots$

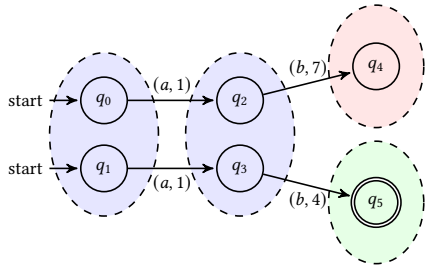


Fig. 4. An example without the distinctness condition.

Lemma 5.10. *Given a NFA $M = (Q_M, \Sigma_M, \Delta_M, Q_M^0, F_M)$ generated from a prepared table $\mathcal{T}$, and the hypothesis NRTA $\mathcal{H} = (Q, \Sigma, \Delta, Q_0, F)$ transformed from M. For all $\omega \in S \cup R$ which can be generated by M, we have $\mathcal{H}$ and M share the same set of runs by reading ω.*

Proof. Suppose $\omega = \epsilon$, then $\mathcal{H}$ and M share the same set of runs since $Q_0 = Q_M^0$. Suppose $\omega = \omega' \cdot (\sigma_k, \tau_k)$ is a timed word which can be generated by M. Since $\omega \in S \cup R$, then $\omega' \in S \cup R$. By induction, we have M and $\mathcal{H}$ share the same set of runs by reading ω'. Let $\{q_1, q_2, \ldots, q_m\} = \hat{\delta}(Q_M^0, \omega')$ in M, then $\{q_1, q_2, \ldots, q_m\} = \delta(Q_0, \omega')$ in $\mathcal{H}$ by the inductive hypothesis. Let *Source* be a subset of $\{q_1, q_2, \ldots, q_m\}$ such that for each $q \in$ *Source* there exists $s \in S$ with $q = q_{val(s)}$ and $s \cdot (\sigma_k, \tau_k) \in S \cup R$. For ω, there are also prime rows $s \in S$ with $row(s) \subseteq row(\omega)$. Suppose $\{q_1', q_2', \ldots, q_n'\} = \hat{\delta}(Q_M^0, \omega)$, denoted as *Target*. Let *Seg* be the set of segments going from *Source* to *Target* which can occur as the last segment of a run in M by reading ω. Let *Seg'* be its counterpart in $\mathcal{H}$. If $\mathcal{H}$ and M do not share the same set of runs by reading ω, there are two possibilities:

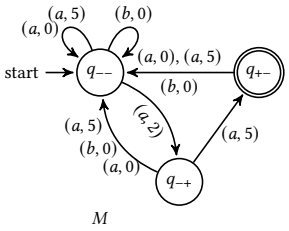
1. Suppose there is a segment $q \xrightarrow[\tau_k]{\sigma_k} q'$ in *Seg'* but not in *Seg*. If $q \notin \{q_1, q_2, \ldots, q_m\}$, then q can be reached by a run reading ω' in $\mathcal{H}$ but not in M. This is a contradiction to the induction hypothesis. If $q' \notin \{q_1', q_2', \ldots, q_n'\}$, then q' can be reached by a run reading ω in $\mathcal{H}$ but not in M. This contradicts the bijection between Δ_M and Δ (and each transition in Δ is obtained by extending the corresponding transition in Δ_M to an interval).
   If $q \in \{q_1, q_2, \ldots, q_m\}$ and $q \notin$ *Source*, then there exists $s \in S$ such that $q = q_{val(s)}$, but $s \cdot (\sigma_k, \tau_k) \notin S \cup R$ and $s \cdot (\sigma_k, \tau_k') \in S \cup R$, where $[\![\tau_k']\!] \neq [\![\tau_k]\!]$ and $\tau_k' < \tau_k$. As it leads to a transition $(q_i, \sigma_k, I_k', q_j')$, where $I_k' = [\lfloor \tau_k' \rfloor, ?)$ or $I_k' = (\lfloor \tau_k' \rfloor, ?)$, τ_k satisfies the guard I_k' if the upper-bound of I_k' is greater than τ_k. However, $s \cdot (\sigma_k, \tau_k) \notin S \cup R$ contradicts the distinctness condition. Since $\omega = \omega' \cdot (\sigma_k, \tau_k) \in S \cup R$, then $s \cdot (\sigma_k, \tau_k) \in S \cup R$ for all $row(s) \subseteq row(\omega')$. Then the upper-bound of I_k' will be less than τ_k, as the new transition is $(q_i, \sigma_k, I_k', q_j')$, where $I_k' = [\lfloor \tau_k' \rfloor, \lfloor \tau_k \rfloor)$ or $I_k' = (\lfloor \tau_k' \rfloor, \lfloor \tau_k \rfloor)$.
2. Suppose there is a segment $q \xrightarrow[\tau_k]{\sigma_k} q'$ in *Seg* but not in *Seg'*. This is again a contradiction to the bijection between Δ_M and Δ.

Hence, by induction, $\mathcal{H}$ and M have the same set of runs by reading ω. □

Thanks to the distinctness condition and the bijection between Δ_M and Δ, we can guarantee the compatibility between $\mathcal{H}$ and M. The following example illustrates the situation without the distinctness condition.

Example 5.11. In Fig. 4, the dashed ellipses represent all ending locations after reading a timed word. There are two initial locations q_0 and q_1 of M. M can reach q_2 and q_3 after reading timed action $(a, 1)$. q_2 and q_3 are indicated by $(a, 1)$ and $(b, 3)(a, 1)$ respectively. M reaches a non-accepting

$\mathcal{T}$	ϵ	$(a,5)$
ϵ	−	−
$(a,2)$	−	+
$(a,2)(a,5)$	+	−
$(a,0)$	−	−
$(b,0)$	−	−
$(a,2)(a,0)$	−	−
$(a,2)(b,0)$	−	−
$(a,5)$	−	−
$(a,2)(a,5)(a,0)$	−	−
$(a,2)(a,5)(b,0)$	−	−
$(a,2)(a,5)(a,5)$	−	−



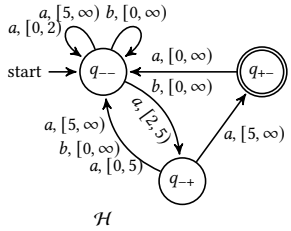


Fig. 5. A prepared table $\mathcal{T}$, the corresponding NFA M and the hypothesis NRTA $\mathcal{H}$.

location q_4 from q_2 after reading $(b,7)$ and reaches an accepting location q_5 from q_3 after reading $(b,4)$. As $row((b,3)(a,1)) \subseteq row((a,1))$ and $(b,3)(a,1)\cdot(b,7) \notin S\cup R \wedge (b,3)(a,1)\cdot(b,4) \in S\cup R$, where $[\![4]\!] \neq [\![7]\!] \wedge 4 < 7$, it follows that $\mathcal{H}$ and M do not share the same set of runs for $(a,1)(b,7)$ by Lemma 5.10. Thus, in the transformed $\mathcal{H}$, there exist two transitions $(q_3, b, [4,\infty), q_5)$ and $(q_2, b, [7,\infty), q_4)$. Obviously, $(a,1)(b,7)$ is accepted by $\mathcal{H}$ as there exists a $\rho = q_1 \xrightarrow[1]{a} q_3 \xrightarrow[7]{b} q_5$ with $q_5 \in F$. Thus, $\mathcal{H}$ is not compatible with M. What's worse, if $(a,1)(b,7)$ is added by a negative counterexample, $(a,1)(b,7)$ will be a repeating counterexample since $\mathcal{H}$ is not compatible with M.

However, $row((b,3)(a,1)) \subseteq row((a,1))$ since M can reach q_3 by reading $(a,1)$. With the distinctness condition, $(b,3)(a,1)\cdot(a,7)$ should be added in R. Thus, there must exist a transition $(q_3,(b,7),q_?)$ in M for every $q_? \in \hat{\delta}(q_3,(b,7))$. What's more, $\hat{\delta}(q_3,(b,7)) \cap F_M = \emptyset$ as $(a,1)(a,7)$ is not accepted by M. After the hypothesis construction, there exist new transitions $(q_3, b, [4,7), q_5)$ (corresponding to $(q_3,(b,4),q_5)$) and $(q_3, b, [7,+), q_?)$ (corresponding to $(q_3,(b,7),q_?)$) instead of transition $(q_3, b, [4,\infty), q_5)$. Therefore, the distinctness condition can guarantee the avoidance of repeating counterexamples.

Theorem 5.12. *If the NFA M is compatible with $\mathcal{T}$, then $\mathcal{H}$ is a canonical RRTA.*

Proof. (sketch) Similar to [8], M will be a canonical RFSA if M is compatible with $\mathcal{T}$, since M satisfies Lemma 5.6, 5.7 and 5.8. Due to Lemma 5.10, M and $\mathcal{H}$ share the same set of runs. Based on the definition of CRRTA and the construction of $\mathcal{H}$ from M, we have $\mathcal{H}$ is a canonical RRTA if M is a canonical RFSA. Hence, $\mathcal{H}$ is a canonical RRTA if M is compatible with $\mathcal{T}$. □

Example 5.13. As shown in Fig. 5, we illustrate the two steps construction from $\mathcal{T}$ to M and M to $\mathcal{H}$. Note that we have combined some transitions and intervals. Since there are three prefixes in S, NFA M has three locations q_{--}, q_{-+} and q_{+-}. We show an example on building transitions according to the prefixes $(a,2)$ and $(a,2)(a,5)$. The source location is determined by $q_{val((a,2))} = q_{-+}$. Taking the action $(a,5)$, the automaton jumps to either q_{--} or q_{+-}, since $row(\epsilon) \subseteq row((a,2)(a,5))$ and $row((a,2)(a,5)) \subseteq row((a,2)(a,5))$. From M to $\mathcal{H}$, we take an example on recovering the guards on transitions with the source location q_{--} and the action a. We first build the set $\Psi_{q_{--},a} = \{0,2,5\}$ and then get the intervals $[0,2)$,$[2,5)$ and $[5,\infty)$ after applying the partition function.

5.3 Equivalence query and counterexample processing

After constructing a hypothesis NRTA $\mathcal{H}$, the learner conducts an equivalence query to the teacher, e.g. whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}$. We denote the equivalence query oracle as EQ. In theory, suppose that the teacher holds an RTA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. The teacher can answer the query be checking whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{A})$. The complexity of the equivalence problem of NRTA is PSPACE-complete and can be decided by first converting NRTA to DRTA and then using classic methods to decide the inclusion problem of two DRTAs. However, it is not scalable. In [9], Bonchi and Pous introduced an algorithm

named HKC to decide the equivalence problem of NFAs based on technique of bisimulation up to congruence. We adapt it to decide the equivalence problem of NRTAs. In practice, the equivalence query is usually done by conformance testing.

If the teacher gives a negative answer for an equivalence query, the learner can also receive a timed word ω as a counterexample for $\mathcal{L}(\mathcal{H}) = \mathcal{L}$, i.e. either $\omega \in \mathcal{L}(\mathcal{H}) \wedge \omega \notin \mathcal{L}$ or $\omega \notin \mathcal{L}(\mathcal{H}) \wedge \omega \in \mathcal{L}$. We call ω a negative counterexample and a positive counterexample in the two cases, respectively.

Processing counterexample: Step 1. Suppose a counterexample $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$. First, the learner apply a normalization function *norm* on ω and get $\omega' = (\sigma_1, \tau_1')(\sigma_2, \tau_2') \cdots (\sigma_n, \tau_n')$ as a new counterexample.

Definition 5.14 (Normalization function [4]). A normalization function *norm* maps a timed word $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$ to another timed word by resetting any clock valuation to its integer part plus a constant fractional part, i.e., $norm(\omega) = (\sigma_1, \tau_1')(\sigma_2, \tau_2') \cdots (\sigma_n, \tau_n')$, where $\tau_i' = \tau_i$ if $\tau_i \in \mathbb{N}$, and otherwise $\tau_i' = \lfloor \tau_i \rfloor + \theta$ for some fixed constant $\theta \in (0, 1)$.

The following theorem guarantees that ω' is still a correct counterexample. In this paper, we set $\theta = 0.1$. Clearly our approach works for any other value in interval $(0, 1)$.

Theorem 5.15 ([4]). *If a timed word $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n) \in \mathcal{L}(\mathcal{H}) \ominus \mathcal{L}$, its normalization $\omega' = norm(\omega) \in \mathcal{L}(\mathcal{H}) \ominus \mathcal{L}$, where $\ominus$ denotes the symmetric difference of two sets.*

Proof. (sketch) It can be proved according to the definitions of region and NRTA. □

Processing counterexample: Step 2. After the normalization, add all prefixes of ω' to R and add all suffixes of ω' to E, except for those already in $S \cup R$ and E respectively. The learner then fills the current table $\mathcal{T}$ by membership queries.

Our counterexample process is different from L^*'s and NL^*'s. All prefixes of ω' are added in R, as we want to query other timed actions (σ, τ), where $\sigma \in \Sigma$ and $\tau \neq 0$, since we only add $s \cdot (\sigma, 0)$ to R when making table closed. The query results will help to modify the guards in transitions. Likewise, we want to collect suffixes which can distinguish the prime residuals indicated by the prefixes $s \in S$, so all suffixes of a counterexample are added to E. In [21] and [8], it is only needed to add all suffixes of a counterexample to E, since $s \cdot \sigma$ for every $\sigma \in \Sigma$ is added to the table when making table closed. However, we cannot obtain this as the number of timed actions (σ, τ) is infinite.

This procedure can be viewed as a nondeterministic version combining the counterexample process in [4] and that in [21]. The important thing is to consider all $s \in S$ with $row(s) \subseteq row(\epsilon)$ which indicate multiple initial locations instead of just considering ϵ. Thanks to the distinctness condition, we just need to add all prefixes of ω' to R rather than all elements in $\{s \cdot u \mid s \in S \wedge row(s) \subseteq row(\epsilon) \wedge u \in prefixes(\omega')\}$ to R in this step, where $prefixes(\omega')$ is the prefix set of ω'. Since u can be viewed as $\epsilon \cdot u$, if the table is distinct, then $\epsilon \cdot u \in S \cup R \Rightarrow s_i \cdot u \in S \cup R$, where $s_i \in \{s \mid row(s) \subseteq row(\epsilon)\}$. Removing the condition will lead to repeating counterexamples as ϵ may be composed and thus not in S. As a result, adding all prefixes of counterexamples may not exclude any error in the current hypothesis. Therefore, the learning process will not terminate.

Improving Step 2 by homing sequences decomposition. In [22], Rivest and Schapire introduced a decomposition on counterexamples. We adapt it to a nondeterministic version in our counterexample processing (Step 2) as an improvement. For a normalized counterexample ω', if $\bigvee_{q_{val(s)} \in Q_0} \mathrm{MQ}(s \cdot \omega') \neq \mathrm{MQ}(\omega')$, then there exists a decomposition $\omega' = u \cdot \boldsymbol{\sigma} \cdot v$ into a prefix u, a timed action $\boldsymbol{\sigma} \in \Xi$, and a suffix v such that $\bigvee_{q_{val(s)} \in \delta(Q_0, u)} \mathrm{MQ}(s \cdot \boldsymbol{\sigma} v) \neq \bigvee_{q_{val(s')} \in \delta(Q_0, u\boldsymbol{\sigma})} \mathrm{MQ}(s' \cdot v)$. Such a decomposition can be found by using binary search on ω'. Then we add all prefixes of ω' to R and all suffixes of v in E instead of adding all prefixes and suffixes of ω' to the table in Step 2. Such operation

Algorithm 1: NRTALearning

```
   input  : the real-time observation table T = (Σ, Ξ, S, R, E, f).
   output : a canonical RRTA H recognizing the target real-time language L.
1  S ← {ϵ}; R ← {ω | ω = ϵ · (σ, 0), ∀σ ∈ Σ}; E ← {ϵ} ;          // initialization
2  fill T by membership queries;
3  equivalent ← ⊥;
4  while ¬equivalent do
5      prepared ← is_prepared(T) ;                  // whether the table is prepared
6      while ¬prepared do
7          if T is not reduced then make_reduced(T) ;
8          if T is not closed then make_closed(T) ;
9          if T is not consistent then make_consistent(T) ;
10         if T is not evidence-closed then make_evidence_closed(T) ;
11         if T is not distinct then make_distinct(T) ;
12         prepared ← is_prepared(T);
13     M ← build_NFA(T) ;                          // constructing an NFA M from T
14     H ← build_hypothesis(M) ;                   // constructing an NRTA H from M
15     equivalent, ctx ← equivalence_query(H) ;    // ctx represents a counterexample
16     if equivalent = ⊥ then
17         ctx_processing(T, ctx) ;                          // counterexample processing
18 return H;
```

will reduce the size of R and E comparing to the original Step 2, and thus reduce the number of membership queries. Additionally, it still maintains $S \cup R$ prefix-closed and E suffix-closed.

5.4 NRTA learning algorithm, correctness and complexity analysis

Algorithm 1 combines all components described in Section 5.1,5.2 and 5.3. First, the learner makes an initial table $\mathcal{T}$ by adding ϵ to the prefix set S and the suffix set E by Definition 5.1 and adding $\epsilon \cdot (\sigma, 0)$ to the set of extended prefixes R (Line 1). After filling $\mathcal{T}$ by several membership queries (Line 2), the learner checks if $\mathcal{T}$ is prepared or not (Line 5). If not, repeatedly check each table condition in sequence to find and fix violated one by the operations described in Section 5.1 (Line 7-11) until the current table $\mathcal{T}$ is prepared. In case $\mathcal{T}$ is prepared, the learner can build an NFA M (Line 13) and a hypothesis NRTA $\mathcal{H}$ afterwards (Line 14) by the construction procedure presented in Section 5.2. Then an equivalence query is performed by submitting $\mathcal{H}$ to the teacher (Line 15). The teacher returns the answer and a counterexample *ctx* in addition if the answer is negative. After performing a counterexample process (Line 17) described in Section 5.3, the learner needs to check whether the current table $\mathcal{T}$ is prepared again. The whole procedure repeats until the teacher gives a positive answer to an equivalence query, and the algorithm returns the current hypothesis $\mathcal{H}$ as a canonical RRTA recognizing the target real-time language $\mathcal{L}$ (Line 18). The correctness and termination of Algorithm 1 is stated in the following theorem.

Theorem 5.16 (Correctness and termination). *Algorithm 1 terminates and returns the canonical RRTA $\mathcal{H}$ which recognizes the target real-time language $\mathcal{L}$.*

Proof. We first show the correctness of Algorithm 1. Suppose Algorithm 1 terminates and return an NRTA $\mathcal{H}$. Since $\mathcal{L}(\mathcal{H})$ passed the equivalence checking with $\mathcal{L}$, then $\mathcal{H}$ recognizes $\mathcal{L}$. Consequently, $\mathcal{H}$ can pass all membership queries on $\omega \cdot e \in (S \cup R) \cdot E$. By Theorem 5.12, the NRTA $\mathcal{H}$ is a CRRTA. Then we can conclude that $\mathcal{H}$ is the CRRTA recognizing $\mathcal{L}$. The termination is guaranteed by Theorem 5.17 proved in the following. □

Suppose that the unique minimal DRTA for the target real-time language $\mathcal{L}$ has n locations, the length of the longest returned counterexample is h and let $|\Sigma| = m$. As a convention in active learning, the learning complexity is measured in terms of the number of two kinds of queries rather than the time complexity.

Theorem 5.17 (Complexity). *Algorithm 1 takes $O((n+nh(n^2+m\kappa n))\cdot h(n^2+m\kappa n)))$ membership queries and $O(n^2+m\kappa n)$ equivalence queries to learn the CRRTA recognizing $\mathcal{L}$.*

Proof. The proof is similar to that given in [8] for nondeterministic finite automata. We give an outline of the proof below, mainly focusing on the difference from [8]'s.

The main idea is to consider a measure including four natural numbers: l_s for the number of distinct rows ever in S, l for the number of distinct rows in the entire table, p for the number of prime rows in the table, and i for the number of strict containing relations of pairs of distinct rows in the table, i.e. $row(\omega) \subset row(\omega')$ where $\omega, \omega' \in S \cup R$. Clearly, the value of l_s, l, and p is upper-bounded by n. The value of i can only increase when l increases. Whenever l is increased by k, the value i increases by at most $kl + k(k-1)/2$. So i can be increased up to $n(n-1)/2$.

Next, consider how to measure changes after each operation. If the table is not closed, then extending the table increases l_s by 1 and possibly increases l by $k > 0$. If the table is not consistent or evidence-closed, extending the table does not change l_s, and might increase l. If l is not increased, then i is decreased by at least 1. These two cases are exactly the same as for NFA.

Now consider the processing of a counterexample. The main argument in [8] is that if l stays the same, then either i decreases or p increases, for otherwise the automata $\mathcal{H}_{\mathcal{T}}$ and $\mathcal{H}_{\mathcal{T}'}$ constructed before and after the processing must be the same, contradicting the fact that $\mathcal{H}_{\mathcal{T}'}$ handles an additional counterexample. In our case, there is another possible modification to the automata: splitting an edge into multiple edges by refining the guard. For each distinct row, its guard can be refined by at most $2\kappa + 1$ times, so the total number of refinements is at most $O(m\kappa n)$.

Finally, the process for maintaining the distinctness condition can at most add one row for each distinct row and timed action (pair of an action and a value containing in a region), hence the number of rows added to maintain distinctness is at most $O(m\kappa n)$.

The above argument shows that Algorithm 1 always reaches an equivalence query and terminates after performing $O(n^2 + m\kappa n)$ equivalence queries.

Now we consider the number of membership queries. The number of rows in S is upper-bounded by n. The operation making $\mathcal{T}$ evidence-closed may add at most $O(n \cdot h(n^2 + m\kappa n))$ rows in R. The counterexample process and the operation for distinctness add at most $O(n \cdot h(n^2 + m\kappa n))$ rows in R together and $O(h(n^2 + m\kappa n))$ columns in E. Hence, the number of membership queries is at most $O((n + nh(n^2 + m\kappa n)) \cdot h(n^2 + m\kappa n)))$. □

5.5 Extended-NL^*

Suppose $\mathcal{L}$ is a real-time language, based on the corresponding Myhill-Nerode Theorem for real-time languages, there is a constant $\kappa \in \mathbb{N}$ which will be the maximum value appearing in the guards of the corresponding RTAs recognizing $\mathcal{L}$. According to the definition of CRRTAs, assuming that κ is known before learning, we can extend NL^* algorithm to learn real-time languages as follows.

First construct all regions $[0,0],(0,1),\ldots,[\kappa,\kappa],(\kappa,\infty)$ offline, denoted as $Reg_0, Reg_1, \ldots, Reg_{2\kappa+1}$. Then select a number τ_i' from each region. If $Reg_i = [\tau_i, \tau_i]$ is a point region, then $\tau_i' = \tau_i$. Otherwise, choose $\tau_i' = \lfloor \tau_i \rfloor + 0.1$ where $\tau_i \in Reg_i$. After that, an abstract alphabet $\Sigma_{M'} = \{(\sigma, \tau_i') | \sigma \in \Sigma \wedge \tau_i' \in Reg_i \wedge 0 \leq i \leq 2\kappa+1\}$ can be constructed before learning. $\Sigma_{M'}$ is a finite set with $|\Sigma_{M'}| = (2\kappa+2)\cdot|\Sigma|$.

The learning process also follows the two steps construction. At the first step, an NFA $M' = (Q_{M'}, \Sigma_{M'}, \Delta_{M'}, Q^0_{M'}, F_{M'})$ with the finite alphabet $\Sigma_{M'}$ can be constructed using the NL^* algorithm

directly. At the second step, NFA M' is transformed to an NRTA $\mathcal{H}' = (Q', \Sigma, \Delta', Q'_0, F')$ by reusing the process presented in Section 5.2. From the result in [8], we have the following theorem.

Theorem 5.18 (Extended-NL^*). *The Extended-NL^* algorithm terminates and returns the CRRTA which recognizes the target real-time language $\mathcal{L}$ after performing $O(h\kappa mn^3)$ membership queries and $O(n^2)$ equivalence queries.*

Extended-NL^* is a conservative method since it makes membership queries for all regions at all locations. Note that instead of assuming κ is known before learning, we can also guess an initial value of κ, then increase it whenever a counterexample involves a larger time value. However, this may result in restart of the learning process frequently.

5.6 Running example

In this section, we present the learning process for NRTA $\mathcal{A}$ in Fig. 2. To simplify the presentation, we omit the evidence-closed condition which has no effect on this example. In Fig. 6, $\mathcal{T}_1$ is the initial instance of the table. Since $\mathcal{T}_1$ is prepared, the learner builds an NFA M_1 and a hypothesis NRTA $\mathcal{H}_1$ shown in Fig. 7. After making an equivalence query to the teacher, the learner receives a positive counterexample $ctx_1 = (a, 5.1)(a, 8.1)$. After processing the counterexample, $\mathcal{T}_2$ is generated. The procedure from $\mathcal{T}_2$ to $\mathcal{T}_5$ can be found in Example 5.4. As $\mathcal{T}_5$ is not closed, the learner moves $(a, 0)$ from R to S. After making the table distinct and closed, the learner obtains another prepared table $\mathcal{T}_8$ and builds an NFA M_2 and hypothesis $\mathcal{H}_2$. The learner receives a negative counterexample $ctx_2 = (a, 5.1)(a, 15)$ and generates $\mathcal{T}_9$. Then after 11 more rounds including processing 5 counterexamples, the learner gets a prepared table $\mathcal{T}_{20}$ and builds the final hypothesis $\mathcal{H}_8$ which recognizes the target language $\mathcal{L}(\mathcal{A})$. Obviously, if we delete the "sink" location q_{-----} and related transitions, then combine the intervals on the transition from q_{--+-+} to q_{-+---} and the intervals on the transition from q_{+----} to q_{-+---} respectively, the resulting automaton will be the same as $\mathcal{A}$ in Fig. 2.

6 EXPERIMENTS

We implemented the two algorithms NRTALearning and Extended-NL^* in Python and evaluated both algorithms on two benchmarks which include a set of randomly generated NRTAs and a set of rational regular expressions. We also demonstrate the algorithm NRTALearning on a scheduling example. All experiments have been conducted on an Intel Core i5-9600 @ 3.1GHz processor with 16GB RAM running 64-bit Debian GNU/Linux 10 equipped with Python 3.7.3.

6.1 Randomly generated NRTAs

In the first experiment, 100 randomly generated NRTAs are divided in 10 groups, with each group having different numbers of locations $|Q|$ and size of alphabet $|\Sigma|$. $|Q|$ ranges from 10 to 20 and $|\Sigma|$ ranges from 2 to 10. We set the maximum constant appearing in guards $\kappa = 20$. We abbreviate the algorithms NRTALearning and Extended-NL^* as NRTA-L and E-NL^* respectively. DRTA-L denotes the learning algorithm for DRTAs in [5]. The experimental results are shown in Table 1.

The three algorithms are all successful to learn out all 100 models. For all groups, NRTA-L takes fewer membership queries than E-NL^* and DRTA-L takes the most number of membership queries to learn the corresponding minimal DRTAs. NRTA-L is more radical for guessing the bounds of guards by using the partition function while E-NL^* is more conservative. However, the drawback of the radical operation is that we need more equivalence queries to correct the guards, which explains why NRTA-L takes more equivalence queries. Since the language equivalence problem of NRTAs is PSPACE-complete, more equivalence queries means more average running time for NRTA-L. Note that the complexity is measured in terms of the number of queries in the automaton learning theory as we mentioned before. The other observation is that the learnt CRRTAs are much

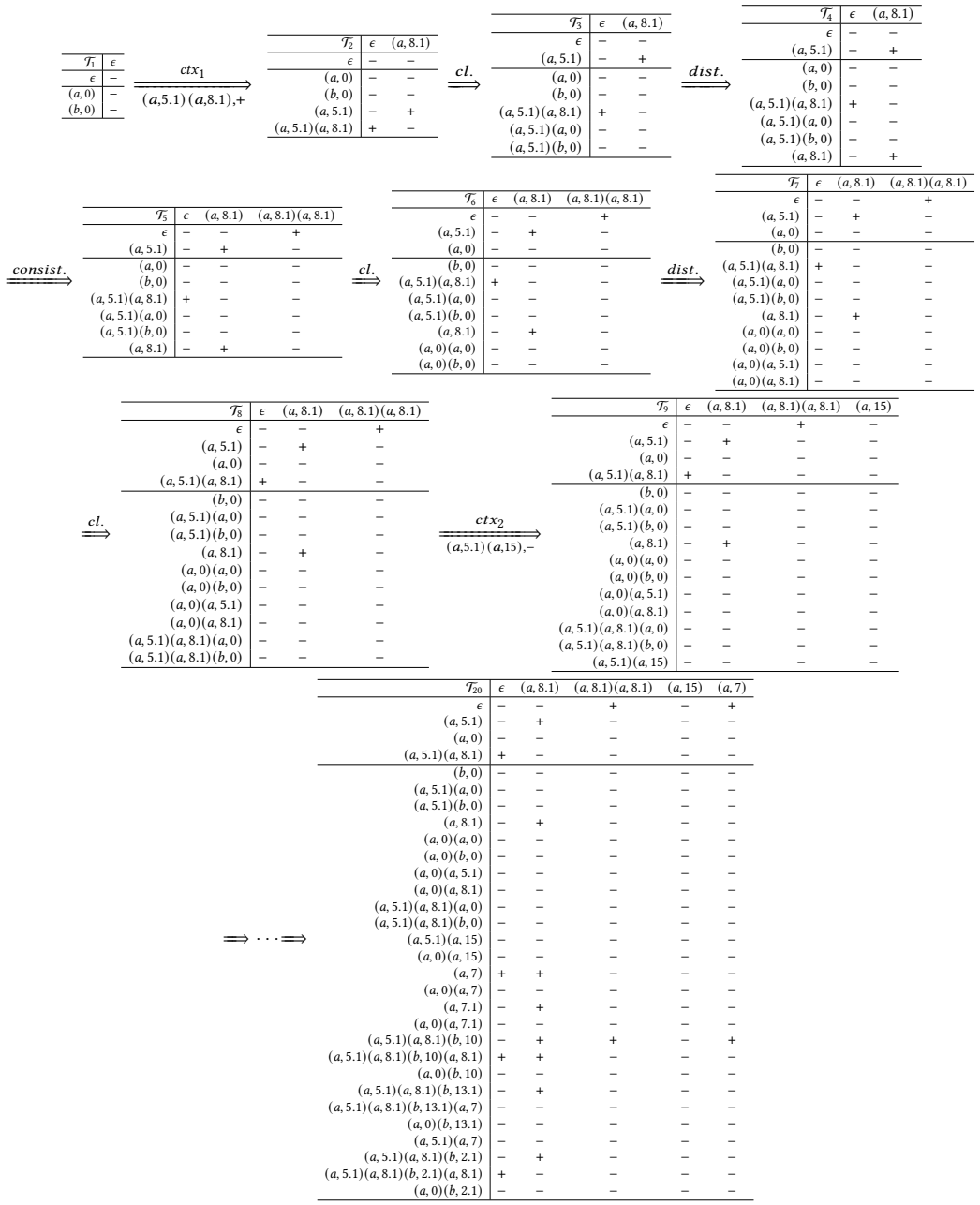


Fig. 6. Some of the table instances generated in the process of learning NRTA $\mathcal{A}$ in Fig. 2.

smaller (w.r.t. the number of locations) than the corresponding minimal DRTAs, which conforms to the contents on residual real-time languages in Section 4.

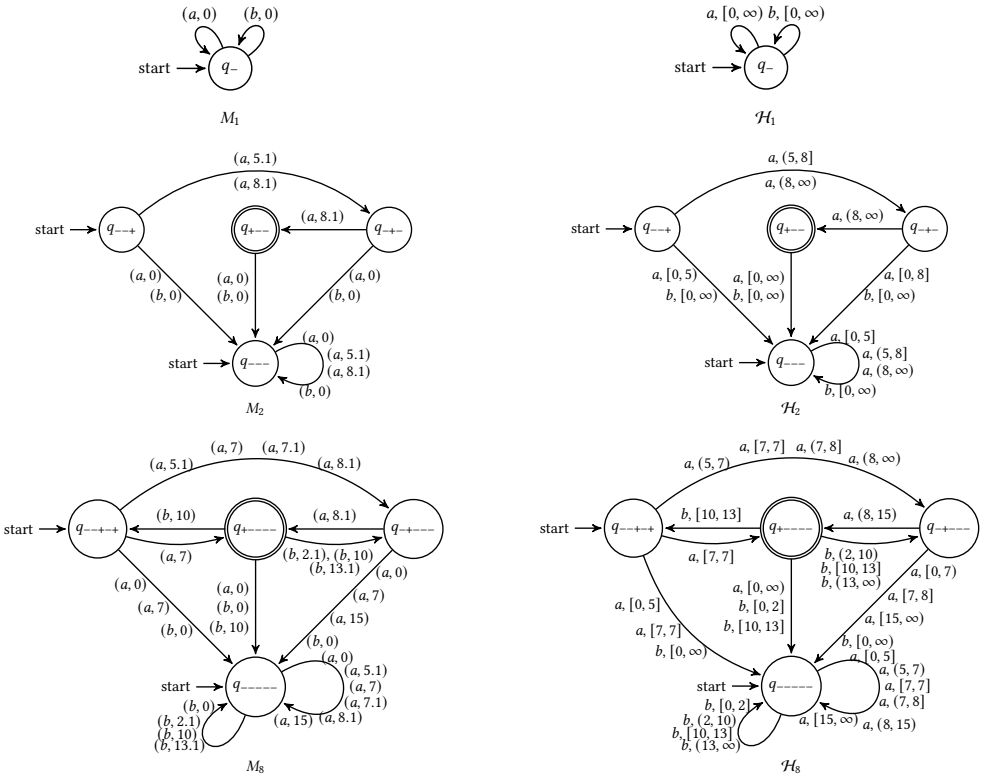


Fig. 7. Some of the NFAs and hypothesis NRTAs generated in the process of learning NRTA $\mathcal{A}$ in Fig. 2.

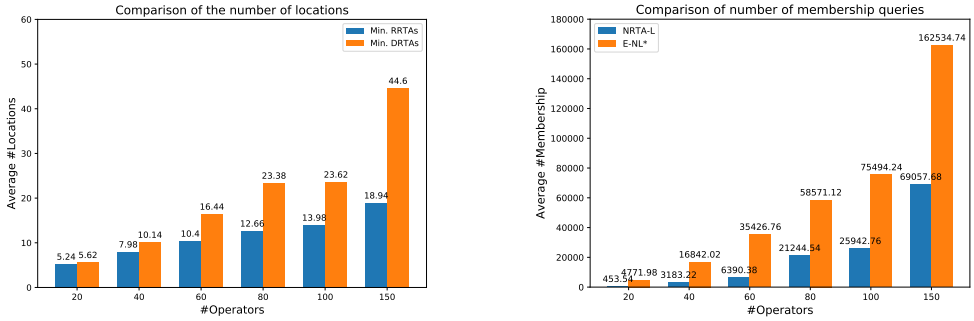


Fig. 8. Comparison results on randomly generated rational regular expressions. The expressions are divided into 6 groups according to the number of the used operators (#Operators). **Left:** Comparison of the average number of locations between the learnt minimal RRTAs and the corresponding minimal DRTAs. **Right:** Comparison of the average number of membership queries performed by NRTA-L and E-NL^*.

6.2 Randomly generated rational regular expressions

In the second experiment, following [13], we fixed a set of operators $\{\emptyset, \Sigma, *, \cdot, +\}$, where $\Sigma = \{a, b, c\}$ and $*, \cdot, +$ represent the Kleene star, concatenation and union respectively. Then we randomly generated 300 different rational regular expressions based on the operators. These real-time languages

Table 1. Experimental results on the randomly generated NRTAs.

Group ID	$\|\Delta\|$	n_{DRTA}	Method	#Membership			#Equivalence			$\|Q_{\mathcal{H}}\|$	t (s)
				$N_{\min}$	N_{mean}	$N_{\max}$	$N_{\min}$	N_{mean}	$N_{\max}$		
10_2_20	23.5	66.5	NRTA-L	3520	5665.3	9660	31	36.4	40	11.1	1.7
			E-NL^*	15725	26571.5	35310	3	4.7	6		1.1
			DRTA-L	38461	223977.9	670380	6	9.1	13	66.5	19.3
10_4_20	32.3	62.7	NRTA-L	4975	9664.4	15192	33	49.2	62	11.1	3.8
			E-NL^*	31433	55362.6	75618	4	5.9	9		2.3
			DRTA-L	68561	347454.2	781231	7	8.7	11	62.7	24.6
10_6_20	43.7	79.0	NRTA-L	8876	17346.3	26910	52	62.8	76	11.0	10.6
			E-NL^*	68817	79628.1	96792	3	5.7	8		3.4
			DRTA-L	171886	577082.6	1234835	7	8.9	11	79.0	44.1
10_8_20	56.5	101.7	NRTA-L	14756	20309.8	29886	65	83.1	97	11.0	13.5
			E-NL^*	78642	109356.2	191550	4	5.9	7		4.6
			DRTA-L	354175	1259531.4	2848321	7	9.7	13	101.7	122.3
10_10_20	64.1	109.2	NRTA-L	12078	26558.5	56840	67	92.0	118	11.0	24.4
			E-NL^*	100816	138915.4	174746	4	6.0	8		6.5
			DRTA-L	807271	1632323.2	2763231	8	10.5	13	109.2	176.0
12_4_20	42.1	127.1	NRTA-L	8652	14968.3	21096	50	66.0	80	13.0	7.8
			E-NL^*	47899	74348.9	111755	4	6.0	8		3.1
			DRTA-L	88567	1071517.5	1958611	10	12.5	16	127.1	117.8
14_4_20	49.5	130.3	NRTA-L	10290	21305.2	38372	58	73.6	85	15.1	13.6
			E-NL^*	59997	101304.0	145188	4	6.7	9		4.3
			DRTA-L	422722	1014350.4	3067064	11	11.8	14	130.3	97.1
16_4_20	56.2	325.9	NRTA-L	18258	41004.4	132928	73	85.5	103	17.1	51.0
			E-NL^*	99819	128872.7	182818	3	6.2	9		5.8
			DRTA-L	408270	3883803.1	13490655	10	15.0	20	325.9	909.8
18_4_20	61.8	330.7	NRTA-L	14256	35404.0	61464	73	97.5	128	19.4	34.3
			E-NL^*	140486	178679.7	254072	7	8.6	11		7.9
			DRTA-L	613914	4989360.0	11884279	11	16.6	23	330.7	899.5
20_4_20	68.7	422.7	NRTA-L	26255	56817.1	135675	86	104.0	117	21.0	84.1
			E-NL^*	154600	211740.0	260107	6	7.6	10		9.9
			DRTA-L	2921594	6720561.6	13160576	17	18.6	22	422.7	1392.3

Group ID: each group has ID of the form $|Q|\_|\Sigma|\_\kappa$, where $|Q|$ is the number of locations, $|\Sigma|$ is the size of the alphabet, and κ is the maximum constant appearing in the clock constraints.
$|\Delta|$: average number of transitions of an NRTA in the corresponding group.
n_{DRTA}: average number of locations of the corresponding minimal DRTAs for each group.
Method: NRTA-L, E-NL^*, and DRTA-L representing the learning algorithms NRTALearning, Extended-NL^* and the learning algorithm for DRTAs in [5], respectively.
#Membership & **#Equivalence**: number of membership and equivalence queries, respectively. $N_{\min}$: minimal, N_{mean}: mean, $N_{\max}$: maximum.
$|Q_{\mathcal{H}}|$: average number of locations of the learned automata for each group.
t: average wall-clock time in seconds, including both time taken by the learner and time taken by the teacher.

are divided into 6 groups according to the number of operators used. The distribution p on operators used for random generation is $p_\epsilon = 0.02, p_\Sigma = 0.1, p_* = 0.13, p_\cdot = 0.5$ and $p_+ = 0.25$. Both NRTA-L and E-NL^* are successful in learning all real-time languages. In Fig. 8, the chart on the left shows that the learnt RRTAs are much smaller than the corresponding minimal DRTAs. The chart on the right shows NRTA-L takes fewer membership queries than E-NL^* on average.

6.3 A scheduling example

For the third experiment, we follow a case study conducted in [6] (as Experiment 6.2). It considers the scheduling of final testing of integrated circuits. We first review the setting of the case study in that paper. There are two kinds of products A and B which are processed in two stages: `testing` and `burn_in`. The two jobs for A and B are grouped into 5 and 2 lots respectively. In the `testing` stage, the lots are processed serially on two parallel machines, and it takes one machine 3 time units to handle each A lot and 4 time units to handle each B lot. In the `burn_in` stage, all lots are processed in a batch manner on the two machines, and it takes one machine 10 time units to finish each batch. The maximum batch size for one machine is 5. During `testing`, if a machine is allocated to process A products, it cannot handle B products until all A jobs are finished. Likewise for machines that are allocated to process B products. We assume that it takes $t \in (0, 1]$ time to prepare one machine and $t \in [1, 2]$ time to prepare two machines in each stage. The operations in one stage can start only after both machines are allocated and prepared. We also assume that products are sent to the `burn_in` stage only after the `testing` stage is completely finished.

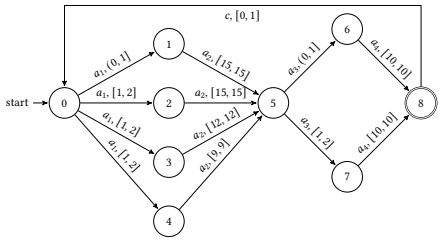
The schedules for processing A and B products are modeled by NRTAs M_A and M_B (see Fig. 9(a) and 9(b)), respectively. M_A has nine locations $0, \ldots, 8$. Here 0 is the initial location, 1 for allocating one machine to test A products, and 2, 3, 4 for allocating two machines, but with different scheduling policies. In 2, one machine has 5 lots and the other has no lots; in 3, one machine has 4 lots and the other has 1 lot; in 4, one machine has 3 lots and the other has 2 lots. Location 5 indicates the `testing` stage has finished. Locations 6, 7 stand for allocating A lots for `burn_in` on one machine and two machines respectively. The accepting location 8 stands for completion of processing, with return to initial location after a delay in $[0, 1]$. The labels a_1, a_2, a_3, a_4 and c represent allocating machines for `testing`, `testing` finished, allocating machines for `burn_in`, `burn_in` finished, and completion, respectively. The NRTA M_B with the labels b_1, b_2, b_3 and b_4 can be understood similarly.

Assuming that we always allocate one machine for each kind of product during the second stage, the two schedules can be composed to form a schedule model for two products, given by an NRTA M_{AB} with 23 locations connected by 39 transitions. The composed model is shown in Fig. 9(c). Using the method NRTA-L, the CRRTA $\mathcal{H}$ (see Fig 9(d)) for the composed model is learned in 12.4 seconds after 10670 membership queries and 50 equivalence queries. The resulting $\mathcal{H}$ is equivalent to M_{AB} but simpler, in particular it has only 17 locations excluding the sink location, connected by 33 transitions.

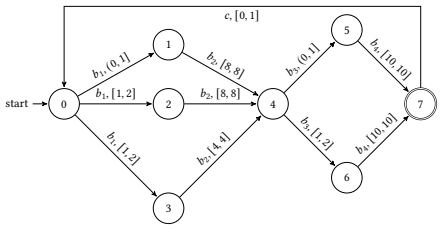
7 CONCLUSION AND DISCUSSION

In this paper, we presented two active learning algorithms for NRTAs. One is in the standard MAT framework, and the other is with additionally assuming that the maximum constant appearing in the clock constraints is known before learning. Before that, we proved a corresponding Myhill-Nerode theorem for real-time languages which shows that there exists a unique minimal DRTA for a given real-time language but it does not hold for NRTAs. In order to set a learning target, we defined residual real-time automata and proved that for a real-time language there is a unique minimal RRTA recognizing it. As a result, learning NRTAs is transformed to learn RRTAs.

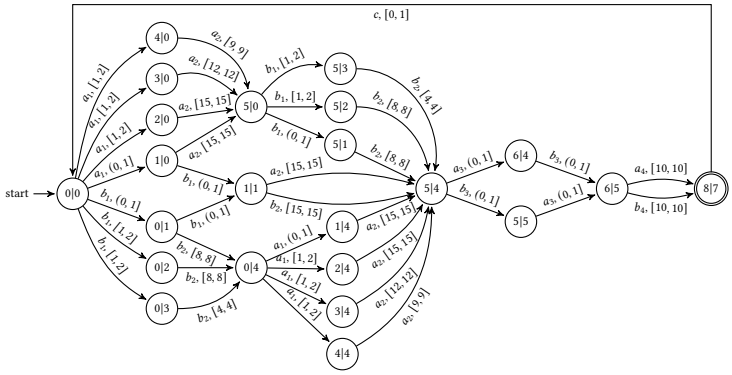
As mentioned before, RTAs are a kind of one-clock timed automata which reset the unique clock at every transition. Hence, it is restricted in that it cannot represent the common timing constraints involving multiple locations and actions. However, RTAs are still powerful enough for specifying the functionality of key-distribution protocols and modelling scheduling problems in practice. We leave for future work to adapt our algorithm to learn nondeterministic one-clock automata whose expressiveness is strictly stronger than deterministic ones and RTAs. For learning general timed automata, one way is to construct the corresponding region automata. However, it



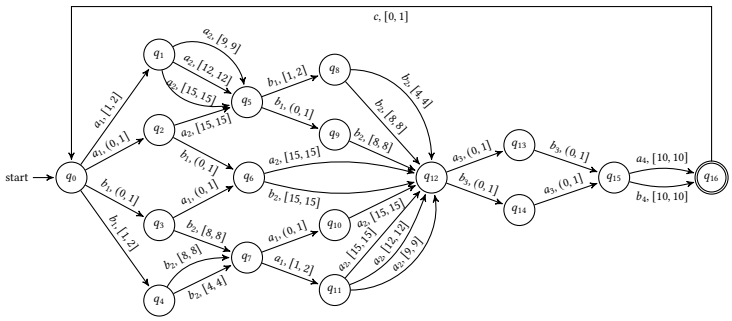
(a) RTA M_A for the schedule of product A



(b) RTA M_B for the schedule of product B



(c) NRTA M_{AB} for the composed schedule of two products A and B. The locations are in the form of $q_A|q_B$, where q_A (q_B) represents the corresponding location of M_A (M_B).



(d) The learnt NRTA $\mathcal{H}$ excluding the "sink" location and relevant transitions

Fig. 9. (a) NRTA M_A; (b) NRTA M_B; (c) The composed NRTA M_{AB}; (d) The learnt NRTA $\mathcal{H}$.

brings an exponential explosion on the size of model. How to learn a succinct model still needs more work.

ACKNOWLEDGMENTS

This work is supported in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248, the CAS Pioneer Hundred Talents Program under grant No. Y9RC585036, and the NSFC under grants No. 61625206, 61732001 and 61972284.

REFERENCES

[1] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. 2020. From Passive to Active: Learning Timed Automata Efficiently. In *Proceedings of the 12th International Symposium NASA Formal Methods, NFM 2020 (LNCS, Vol. 12229)*. Springer, 1–19. https://doi.org/10.1007/978-3-030-55754-6_1

[2] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theoretical Computer Science* 126, 2 (1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8

[3] Rajeev Alur, P. Madhusudan, and Wonhong Nam. 2005. Symbolic Compositional Verification by Learning Assumptions. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005 (LNCS, Vol. 3576)*. Springer, Heidelberg, 548–562. https://doi.org/10.1007/11513988_52

[4] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2020. Learning One-Clock Timed Automata. In *Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2020 (LNCS, Vol. 12078)*. Springer, Heidelberg, 444–462. https://doi.org/10.1007/978-3-030-45190-5_25

[5] Jie An, Lingtai Wang, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2020. Learning real-time automata. *SCIENCE CHINA Information Sciences* (2020). https://www.sciengine.com/doi/10.1007/s11432-019-2767-4 In press.

[6] Jie An, Naijun Zhan, Xiaoshan Li, Miaomiao Zhang, and Wang Yi. 2018. Model Checking Bounded Continuous-time Extended Linear Duration Invariants. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018*. ACM, 81–90. https://doi.org/10.1145/3178126.3178147

[7] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Information and Computation* 75, 2 (1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[8] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. 2009. Angluin-Style Learning of NFA. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*. 1004–1009.

[9] Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013*. ACM, 457–468. https://doi.org/10.1145/2429069.2429124

[10] Ben Caldwell, Rachel Cardell-Oliver, and Tim French. 2016. Learning Time Delay Mealy Machines From Programmable Logic Controllers. *IEEE Trans Autom. Sci. Eng.* 13, 2 (2016), 1155–1164. https://doi.org/10.1109/TASE.2015.2496242

[11] C. de la Higuera. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

[12] François Denis, Aurélien Lemay, and Alain Terlutte. 2001. Residual Finite State Automata. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2001 (LNCS, Vol. 2010)*. Springer, Heidelberg, 144–157. https://doi.org/10.1007/3-540-44693-1_13

[13] François Denis, Aurélien Lemay, and Alain Terlutte. 2004. Learning regular languages using RFSAs. *Theoretical Computer Science* 313, 2 (2004), 267–294. https://doi.org/10.1016/j.tcs.2003.11.008

[14] Dorothy E. Denning and Giovanni Maria Sacco. 1981. Timestamps in Key Distribution Protocols. *Commun. ACM* 24, 8 (1981), 533–536. https://doi.org/10.1145/358722.358740

[15] Catalin Dima. 2001. Real-Time Automata. *Journal of Automata, Languages and Combinatorics* 6, 1 (2001), 3–23. https://doi.org/10.25596/jalc-2001-003

[16] Samuel Drews and Loris D'Antoni. 2017. Learning Symbolic Automata. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017 (LNCS, Vol. 10205)*. Springer, Heidelberg, 173–189. https://doi.org/10.1007/978-3-662-54577-5_10

[17] Paul Fiterau-Brostean, Ramon Janssen, and Frits W. Vaandrager. 2016. Combining Model Learning and Model Checking to Analyze TCP Implementations. In *Proceedings of the 28th International Conference on Computer Aided Verification, CAV 2016 (LNCS, Vol. 9780)*. Springer, Heidelberg, 454–471. https://doi.org/10.1007/978-3-319-41540-6_25

[18] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. 2010. Learning of event-recording automata. *Theoretical Computer Science* 411, 47 (2010), 4029–4054. https://doi.org/10.1016/j.tcs.2010.07.008

[19] Léo Henry, Thierry Jéron, and Nicolas Markey. 2020. Active Learning of Timed Automata with Unobservable Resets. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2020 (LNCS, Vol. 12288)*. Springer, 144–160. https://doi.org/10.1007/978-3-030-57628-8_9

[20] Xiangyu Jin, Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. 2021. Inferring Nonlinear Switched Dynamical Systems. *Formal Aspects of Computing* (2021). https://doi.org/10.1007/s00165-021-00542-7

[21] Oded Maler and Amir Pnueli. 1995. On the Learnability of Infinitary Regular Sets. *Information and Computation* 118, 2 (1995), 316–326. https://doi.org/10.1006/inco.1995.1070

[22] Ronald L. Rivest and Robert E. Schapire. 1993. Inference of Finite Automata Using Homing Sequences. *Information and Computation* 103, 2 (1993), 299–347. https://doi.org/10.1006/inco.1993.1021

[23] Jana Schmidt, Asghar Ghorbani, Andreas Hapfelmeier, and Stefan Kramer. 2013. Learning probabilistic real-time automata from multi-attribute event logs. *Intelligent Data Analysis* 17, 1 (2013), 93–123. https://doi.org/10.3233/IDA-120569

[24] Miriam García Soto, Thomas A. Henzinger, Christian Schilling, and Luka Zeleznik. 2019. Membership-Based Synthesis of Linear Hybrid Automata. In *Proceedings of the 31st International Conference on Computer Aided Verification, CAV 2019 (LNCS, Vol. 11561)*. Springer, Heidelberg, 297–314. https://doi.org/10.1007/978-3-030-25540-4_16

[25] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. 2011. The Digraph Real-Time Task Model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011*. IEEE Computer Society, 71–80. https://doi.org/10.1109/RTAS.2011.15

[26] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. 2019. Time to Learn - Learning Timed Automata from Tests. In *Proceedings of the 17th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2019 (LNCS, Vol. 11750)*. Springer, 216–235. https://doi.org/10.1007/978-3-030-29662-9_13

[27] Frits W. Vaandrager. 2017. Model learning. *Commun. ACM* 60, 2 (2017), 86–95. https://doi.org/10.1145/2967606

[28] Frits W. Vaandrager, Roderick Bloem, and Masoud Ebrahimi. 2021. Learning Mealy Machines with One Timer. In *Proceedings of the 15th International Conference on Language and Automata Theory and Applications, LATA 2021 (LNCS, Vol. 12638)*. Springer, 157–170. https://doi.org/10.1007/978-3-030-68195-1_13

[29] Sicco Verwer. 2010. *Efficient Identification of Timed Automata: Theory and practice*. Ph.D. Dissertation. Delft University of Technology, Netherlands.

[30] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. 2009. One-Clock Deterministic Timed Automata Are Efficiently Identifiable in the Limit. In *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications, LATA 2009 (LNCS, Vol. 5457)*. Springer, 740–751. https://doi.org/10.1007/978-3-642-00982-2_63

[31] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. 2011. The efficiency of identifying timed automata and the power of clocks. *Information and Computation* 209, 3 (2011), 606–625. https://doi.org/10.1016/j.ic.2010.11.023

[32] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. 2012. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning* 86, 3 (2012), 295–333. https://doi.org/10.1007/s10994-011-5265-4

[33] Sicco Verwer, Mathijs De Weerdt, and Cees Witteveen. 2007. An algorithm for learning real-time automata. *Electrical Engineering Mathematics & Computer Science* (2007).

[34] Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018 (PMLR, Vol. 80)*. PMLR, 5244–5253.

[35] Gail Weiss, Yoav Goldberg, and Eran Yahav. 2019. Learning Deterministic Weighted Automata with Queries and Counterexamples. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. 8558–8569.