SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2021, Vol. 64 192103:1–192103:17
https://doi.org/10.1007/s11432-019-2767-4

Learning real-time automata

Jie AN[1], Lingtai WANG[2,3], Bohua ZHAN[2,3*], Naijun ZHAN[2,3*] & Miaomiao ZHANG[1*]

[1]*School of Software Engineering, Tongji University, Shanghai* 201804*, China;*
[2]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190*, China;*
[3]*University of Chinese Academy of Sciences, Beijing* 100190*, China*

Received 20 January 2019/Revised 18 May 2019/Accepted 11 December 2019/Published online 5 August 2021

Abstract Real-time automata (RTAs) are a subclass of timed automata with only one clock which resets at each transition. In this paper, we present an active learning algorithm for deterministic real-time automata (DRTAs) in both continuous-time semantics and discrete-time semantics. For a target language recognized by a DRTA $\mathcal{A}$, we convert the problem of learning DRTA $\mathcal{A}$ to the problem of learning a canonical real-time automaton $\mathbb{A}$ with the same recognized language, i.e., $\mathcal{L}(\mathbb{A}) = \mathcal{L}(\mathcal{A})$. The algorithm is inspired by existing learning algorithms for symbolic automata.

Keywords automaton learning, active learning, real-time automata

Citation An J, Wang L T, Zhan B H, et al. Learning real-time automata. Sci China Inf Sci, 2021, 64(9): 192103, https://doi.org/10.1007/s11432-019-2767-4

1 Introduction

Angluin introduced the L^* algorithm for learning regular sets from queries and counterexamples in her seminal study [1]. This framework is called query learning or active learning, which is distinguished from passive learning (i.e., generating a model from a given set of examples) and many machine learning methods. In Angluin's active automaton learning, instead of training a model from a given data set, a learner wants to learn a regular language from a teacher who knows the regular language and has an oracle to answer queries from the learner. The teacher is assumed to be fully reliable in answering the queries. Under these settings, depending on the decision method for the language equivalence problem of deterministic finite automata (DFA), the L^* algorithm can guarantee to learn a correct DFA which recognizes the target regular language. Many efficient active learning algorithms follow Angluin's querying-answering framework to learn mealy machines [2], register automata [3–5], nondeterministic finite automata [6], Büchi automata [7, 8], and so on. There are also some automaton learning libraries, tools, and applications [9–11].

For timed systems where timing constraints play a key role, however, the situation is much more complicated, because the set of actions with timing information is infinite, making it fundamentally different from the finite alphabet of a classic finite automaton. Because the L^* algorithm cannot handle such an infinite set of timed actions, it is a really difficult but interesting problem to learn a formal model of a timed system. There are also some pioneering studies on learning timed models. A passive learning algorithm was given in [12] to learn deterministic real-time automata [13] from labeled time-stamped event sequences. The generated real-time automaton just accepts all positive labeled sequences and rejects all negative labeled sequences of a given set respectively. A passive learning algorithm for timed automata with one clock was proposed in [14]. Because the finite data set is only a part of the infinite behaviors of the target system or model, passive learning cannot guarantee to learn a correct model of the target system. Event-recording automata [15] are a kind of timed automata that, for every action a, use a clock that records the time of the last occurrence of a. Event-recording automata can be determinized. Its active learning algorithm in [16] is prohibitively complex, owing to the too many degrees of freedom and multiple clocks of event-recording automata.

* Corresponding author (email: bzhan@ios.ac.cn, znj@ios.ac.cn, miaomiao@tongji.edu.cn)

Inspired by the learning algorithms for symbolic automata [17, 18], we focus on the Angluin-style learning algorithm for real-time automata (RTAs) [13] under both continuous-time semantics and discrete-time semantics in this paper. A real-time automaton can be regarded as a timed automaton [19] with only one clock which resets at every transition. 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 [20, 21] and key-distribution protocols [22]. We define a subclass of RTAs named canonical real-time automata (CRTAs) and show that each deterministic real-time automaton (DRTA) can be transformed to a CRTA which has the same recognized language. Therefore, the problem of learning a DRTA can be converted to the problem of learning a CRTA with the same recognized language. The basic ideas are as follows. By preparing a real-time observation table to store the information gathered from membership queries for timed words, the learner can build a DFA M. Then the learner transforms it to a CRTA as a hypothesis $\mathcal{H}$ with a partition function mapping time values to several timing intervals. For an equivalence query, if the answer is positive, it indicates that the CRTA $\mathcal{H}$ recognizes the target language represented by a DRTA originally. Otherwise, the learner receives a counterexample. For the counterexamples which have non-integer time values, we define a refinement function g to normalize these time values. The learner adds the prefixes of the counterexample to the real-time observation table to construct a new hypothesis. The procedure continues until getting the positive answer for an equivalence query. Note that Dima [13] pointed out that RTAs can be determinized. Hence, our method can be applied to both deterministic and nondeterministic RTAs.

To solve the active learning problem for RTAs, we make the following extensions to the traditional L^* algorithm. First, in Subsection 3.2, we modify Angluin's observation table to the real-time observation table. The conditions of the real-time observation table are more complex than the conditions of the observation table in the L^* algorithm. Second, the operations on the real-time observation table are different. Third, two partition functions are introduced to handle infinite timed actions in Subsection 3.3. Fourth, an additional refinement function is used for solving the conflicts caused by the miss-distributions in Subsection 3.4. Finally, in Subsection 3.4, our method for deciding the language equivalence of two RTAs is totally different from the decision method for two DFAs in the L^* algorithm.

Related work. There are several existing studies on learning timed systems. Passive learning algorithms were presented in [12, 14, 23] for real-time automata and timed automata with one clock in discrete-time semantics. A passive learning method tries to learn a model from a given data set. There is no more information that can be gathered, except for the labeled timed words in the data set. The basic idea of the passive learning method for RTAs is as follows. First, the labeled traces in the data set are organized as a tree named prefix tree acceptor. Then the algorithm attempts to merge the nodes of the tree, guided by some heuristics. In the merging process, it needs to protect consistency with the data set. The model learned by such passive learning methods just accepts the positive labeled timed words, and rejects the negative labeled timed words of the given set of timed words respectively. Hence, it cannot guarantee that the generated model recognizes the target language. The discrete-time semantics means that the time values are non-negative integers, while the time value is real numbers in continuous-time semantics. The method for learning event-recording automata (ERAs) is prohibitively complex [16]. Dima [13] pointed out that ERAs are incomparable with RTAs. Genetic programming and machine learning methods are also used to learn timed systems [24, 25]. In this paper, depending on the decision method for the language equivalence problem of DRTAs, our active learning algorithm can efficiently generate correct DRTAs from a reliable teacher in both discrete-time and continuous-time semantics by making use of partition functions and a refinement function.

Structure. The remainder of this paper is organized as follows. In Section 2, we recall preliminaries including the L^* algorithm and real-time automata. The learning algorithm for deterministic real-time automata is introduced in Section 3 including the definitions of the partition functions and the refinement function. Section 4 presents the complexity analysis. Following the implementation of the algorithm and some experiments in Section 5, Section 6 concludes this paper.

2 Preliminaries

We utilize $\mathbb{R}_{\geqslant 0}$ and $\mathbb{N}$ to denote the set of non-negative real numbers and natural numbers, respectively. We fix a finite set Σ of letters, called alphabet. The discrete-time semantics and continuous-time semantics mean that the time values are in $\mathbb{N}$ and $\mathbb{R}_{\geqslant 0}$, respectively.

2.1 Learning deterministic finite automaton

We start by briefly reviewing Angluin's L^* algorithm [1] for learning regular sets from membership queries and equivalence queries. She proved that the class of regular languages could be learned efficiently (i.e., in time polynomial in the size of the canonical deterministic finite automaton for this language).

Definition 1 (DFA)**.** A DFA is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a non-empty finite set of states; Σ is a finite alphabet; $\delta : Q \times \Sigma \to Q$ is the transition relation, a partial function on $Q \times \Sigma$; $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting (final) states.

A word over Σ is a finite sequence $\omega = \sigma_1\sigma_2 \cdots \sigma_n$, where $\sigma_i \in \Sigma$ for $i = 1, 2, \ldots, n$. $|\omega| = n$ is the length of ω. ϵ is the empty word with length $|\epsilon| = 0$. A word ω is called an action if $|\omega| = 0$ or $|\omega| = 1$. Σ^* is the set of words over Σ. The transition function δ can be extended to $\hat{\delta} : Q \times \Sigma^* \to Q$, where $\hat{\delta}(q, \epsilon) = q$, and $\hat{\delta}(q, \omega \cdot \sigma) = \hat{\delta}(\hat{\delta}(q, \omega), \sigma)$ for $q \in Q$, $\sigma \in \Sigma$, and $\omega \in \Sigma^*$. A word $\omega \in \Sigma^*$ is accepted by $\mathcal{A}$ if $\hat{\delta}(q_0, \omega) \in F$. Without causing ambiguity, we also denote $\delta(q, \sigma) = q'$ as $(q, \sigma, q') \in \delta$. For a transition $(q, \sigma, q') \in \delta$, q and q' are called the source state and target state of the transition, respectively.

In the L^* algorithm, a learner is designed to construct a DFA which recognizes the unknown target language $\mathcal{L}$ by asking a reliable teacher questions. The teacher knows the target language $\mathcal{L}$ represented by a DFA and can answer the learner's questions. These questions are two types of queries: (1) membership query, i.e., "Is the word ω in $\mathcal{L}$?", and (2) equivalence query, i.e., "Is the recognized language $\mathcal{L}'$ of my current hypothesis DFA equal to $\mathcal{L}$?". The learner first makes multiple membership queries to gather enough information to construct a hypothesis. Then he makes an equivalence query. If the teacher's answer is positive, the learner will be sure that the hypothesis indeed recognizes the target language $\mathcal{L}$ and the algorithm terminates. Otherwise, the learner receives a counterexample word ctx miss-classified by the hypothesis. The learner should make membership queries guided by the counterexample to gather more information to construct a new hypothesis. This continues until termination. The observation table contains all information that the learner knows about $\mathcal{L}$ at any stage.

Definition 2 (Observation table)**.** An observation table for a DFA $\mathcal{A}$ is a 6-tuple $T = (\Sigma, S, R, E, f, \mathrm{row})$, where Σ is a finite alphabet; $S, R, E \subset \Sigma^*$ are finite sets, S is called the set of prefixes, R is called the boundary, and E is called the set of suffixes; $s \cdot \sigma \in R$ for all $s \in S$ and $\sigma \in \Sigma$; S, R are disjoint[1)]: $S \cup R = S \uplus R$; $S \cup R$ is a prefix-closed set; $f : (S \cup R) \cdot E \to \{-, +\}$ is a classification function such that for a word $\omega \cdot e \in (S \cup R) \cdot E$, $f(\omega \cdot e) = -$ if $\omega \cdot e \notin \mathcal{L}(\mathcal{A})$, and $f(\omega \cdot e) = +$ if $\omega \cdot e \in \mathcal{L}(\mathcal{A})$; row is a function that returns the vector of $f(\omega \cdot e)$ indexed by $e \in E$ for $\omega \in S \cup R$.

Before suggesting a hypothesis, the learner asks membership queries to make the observation table T closed and consistent:

- closed if for every $r \in R$, there exists $s \in S$ such that $\mathrm{row}(s) = \mathrm{row}(r)$.
- consistent if for every $\omega_1, \omega_2 \in S$, $\mathrm{row}(\omega_1) = \mathrm{row}(\omega_2)$ implies $\mathrm{row}(\omega_1 \cdot \sigma) = \mathrm{row}(\omega_2 \cdot \sigma)$ for $\forall \sigma \in \Sigma$.

If the table is not closed, there is some $r \in R$ such that $\mathrm{row}(r)$ is different from $\mathrm{row}(s)$ for all $s \in S$. The learner moves r from R to S, adds all words $r \cdot \sigma$ for $\sigma \in \Sigma$ to R, and makes membership queries to fill the extended observation table.

If the table is not consistent, one inconsistency is resolved through finding two words $\omega_1, \omega_2 \in S, \sigma \in \Sigma$ and $e \in E$ such that $\mathrm{row}(\omega_1) = \mathrm{row}(\omega_2)$ and $f(\omega_1\sigma \cdot e) \neq f(\omega_2\sigma \cdot e)$ and adding this new suffix $\sigma \cdot e$ to E. The learner also needs to fill the extended observation table by making membership queries. The observation table is consistent when no more such words can be found.

If the observation table $T = (\Sigma, S, R, E, f, \mathrm{row})$ is closed and consistent, the learner can construct a hypothesis DFA $H_{\mathcal{A}} = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{\mathrm{row}(s) | s \in S\}$, $F = \{\mathrm{row}(s) | f(s \cdot \epsilon) = +\}$, $q_0 = \mathrm{row}(\epsilon)$ and $\delta(\mathrm{row}(s), \sigma) = \mathrm{row}(s \cdot \sigma)$. When receiving a counterexample ctx, the learner adds all prefixes of ctx to S and the possible inconsistency should be fixed.

2.2 Real-time automata

Real-time automata are very similar to classical finite automata despite that they take time into account as well. RTAs can be defined under continuous-time semantics and discrete-time semantics. We recall the definitions of timed automata and real-time automata in continuous-time semantics as follows.

Definition 3 (Timed automaton [19])**.** A timed automaton is a 6-tuple $\mathcal{A} = (Q, \Sigma, C, q_0, F, E)$ that consists of the following components:

1) $\uplus$: disjoint union of two sets.

- Q is a finite set of states (locations);
- Σ is a finite set called an alphabet or actions of $\mathcal{A}$;
- C is a finite set called the clocks of $\mathcal{A}$;
- q_0 is the initial state;
- $F \subseteq Q$ is the set of accepting states;
- $E \subseteq Q \times \Sigma \times \mathcal{B}(C) \times \mathcal{P}(C) \times Q$ is a set of transitions, where $\mathcal{B}(C)$ is the set of clock (timing) constraints involving clocks from C, and $\mathcal{P}(C)$ is the power set of C. An edge (q, σ, ϕ, r, q') from E is a transition from state q to q' with performing action σ, satisfying guard (timing constraints) ϕ and reseting the clocks in the set r.

Let C be the finite set of real-valued clocks, denoted by x, y, z, etc. We define the set of clock (timing) constraints over C via the following grammar, where $k \in \mathbb{N}$ stands for any non-negative integer, and $\diamond \in \{=, <, >, \leqslant, \geqslant\}$ is a comparision operator: $\phi ::= \text{true} \mid x \diamond k \mid \neg\phi \mid \phi \wedge \phi$. Hence, we can also represent the timing constraints as real number intervals with endpoints in $\mathbb{N}$.

Definition 4 (RTA)**.** An RTA is a 6-tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$, where

- Q is a finite set of states (locations);
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of accepting states;
- $\lambda^c : \Delta \to 2^{\mathbb{R}_{\geqslant 0}}$ is the continuous-time labelling function which assigns a guard (timing constraint) to each transition. We assume that the range λ^c $(\mu \in \Delta)$ is a finite union of intervals whose endpoints are in $\mathbb{N} \cup \{+\infty\}$. λ^c is replaced by $\lambda^d : \Delta \to 2^{\mathbb{N}}$ in discrete-time situation.

A timed word over $\Sigma \times \mathbb{R}_{\geqslant 0}$ is a finite sequence $\boldsymbol{\omega} = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, where $\sigma_i \in \Sigma$ and $\tau_i \in \mathbb{R}_{\geqslant 0}$ for $1 \leqslant i \leqslant n$. $|\boldsymbol{\omega}| = n$ is the length of a timed word ω. We abbreviate (ϵ, t) to $\boldsymbol{\epsilon}$ as the empty word for all $t \in \mathbb{R}_{\geqslant 0}$ and let $|\boldsymbol{\epsilon}| = 0$. A timed word $\boldsymbol{\omega}$ is called a timed action if $|\boldsymbol{\omega}| = 0$ or $|\boldsymbol{\omega}| = 1$. The real number in a timed action represents the time when the action is performed. As to real-time systems, the time can be represented by either the global time or the local time. The global time means wall clock time (or physical time) and the local time means the delay time between two actions, which is measured by the local clock of the considered system.

A run of an RTA $\mathcal{A}$ is either a single initial state q_0 or a finite sequence

$$\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n,$$

where $n > 0$, $(q_{i-1}, \sigma_i, q_i) \in \Delta$, and $\tau_i \in \lambda^c((q_{i-1}, \sigma_i, q_i))$ for $1 \leqslant i \leqslant n$. For the sake of simplicity, we denote $\lambda^c((q_{i-1}, \sigma_i, q_i))$ as $\lambda^c(q_{i-1}, \sigma_i, q_i)$ for $(q_{i-1}, \sigma_i, q_i) \in \Delta$ in this paper.

The trace of a run ρ, is a timed word defined as follows: $\text{trace}(q_0) = \boldsymbol{\epsilon}$; if

$$\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n, \quad \text{trace}(\rho) = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n),$$

where $t_i = \sum_{k=1}^{i} \tau_k$ for $1 \leqslant i \leqslant n$. Here τ_i can be interpreted as the local delay time before σ_i happens and t_i is the global time when the action σ_i happens, so $\text{trace}(\rho)$ is also called the global-timed trace denoted as $\text{trace}^g(\rho)$. Owing to RTAs' specialization that the single clock resets at every transition, actually, τ_i is the clock valuation of the local time when the action σ_i happens. We therefore define the local-timed trace of ρ as a timed word: $\text{trace}^l(q_0) = \boldsymbol{\epsilon}$ and $\text{trace}^l(\rho) = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$ if

$$\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n$$

for $1 \leqslant i \leqslant n$. For an RTA $\mathcal{A}$, the recognized language can be defined on local-timed traces as $\mathcal{L}(\mathcal{A}) = \{\text{trace}^l(\rho) \mid \rho \text{ starts from } q_0 \text{ and ends in } q_n \in F\}$.

Given a global-timed word $\boldsymbol{\omega}^g = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$, it can be easily transformed to an unique local-timed word $\boldsymbol{\omega}^l = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, where $\tau_1 = t_1$ and $\tau_i = t_i - t_{i-1}$ for $2 \leqslant i \leqslant n$. For example, if $\boldsymbol{\omega}_1 = (a, 1.2)(b, 3)(a, 4)$ is a global-timed word, the corresponding local-timed word is $\boldsymbol{\omega}_2 = (a, 1.2)(b, 1.8)(a, 1)$.

An RTA $\mathcal{A}$ is a DRTA if and only if there is at most one run for a given timed word

$$\boldsymbol{\omega} = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n).$$

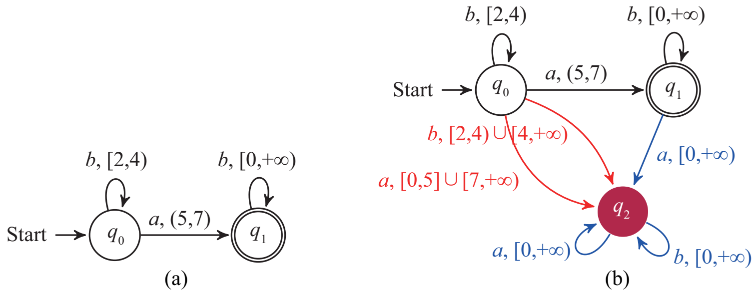


Figure 1 (Color online) (a) A DRTA $\mathcal{A}$ and (b) the corresponding CRTA $\mathbb{A}$. An initial state is indicated by 'Start' and an accepting state is represented by a double cycle in this paper.

Example 1. In Figure 1, the automaton on the left is a DRTA $\mathcal{A}$. The state or location set $Q = \{q_0, q_1\}$, the alphabet $\Sigma = \{a, b\}$, the initial state is q_0, the set of accepting states $F = \{q_1\}$, and the transition relation $\Delta = \{(q_0, b, q_0), (q_0, a, q_1), (q_1, b, q_1)\}$ with $\lambda^c(q_0, b, q_0) = [2, 4)$, $\lambda^c(q_0, a, q_1) = (5, 7)$ and $\lambda^c(q_1, b, q_1) = [0, +\infty)$. For the self-transition on the state q_0, the guard or timing constraint $[2, 4)$ means that the transition can only be fired when there is an action b performed after 2 to 4 time units (except for the integer 4). Given a local-timed word $\boldsymbol{\omega_1} = (b, 2.3)(a, 6)$, it corresponds to an accepting run in $\mathcal{A}$. For the first timed action $(b, 2.3)$, the self-transition on state q_0 can be fired, because 2.3 satisfies the timing constraint $[2, 4)$. After that, the local clock is reset and the transition from q_0 to q_1 can be fired after 6 additional time units, because 6 satisfies the guard (timing interval) $(5, 7)$. The automaton stops at an accepting state q_1. However, the local-timed word $\boldsymbol{\omega_2} = (b, 2.3)$ is the trace of an unaccepting run in $\mathcal{A}$, because the automaton stops at q_0 which is not an accepting state.

3 Learning real-time automata

For a target language recognized by a DRTA $\mathcal{A}$, we transform the problem of learning a DRTA $\mathcal{A}$ to the problem of learning a canonical real-time automaton $\mathbb{A}$ with the same recognized language. First, we give the definition of the canonical real-time automata in continuous-time semantics. The difference between continuous-time semantics and discrete-time semantics is still the difference between the labelling functions λ^c and λ^d. After that, we represent our methods under continuous-time semantics in the remainder of this paper.

3.1 Canonical real-time automata

Definition 5 (CRTA)**.** A CRTA $\mathbb{A} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$ is a DRTA such that:

- For all $q \in Q$, $\Psi_q^\Sigma = \{\sigma | \ q_1 = q \text{ for } (q_1, \sigma, q_2) \in \Delta\}$ has the restriction that $\Psi_q^\Sigma = \Sigma$;
- For all $q \in Q$ and $\sigma \in \Sigma$, $\Psi_{q,\sigma}^{\lambda^c} = \{\lambda^c(q_1, \sigma, q_2) | \ q_1 = q \wedge \sigma' = \sigma \text{ for } (q_1, \sigma', q_2) \in \Delta\}$ has two restrictions: (1) the union of all elements of $\Psi_{q,\sigma}^{\lambda^c}$ should be $\mathbb{R}_{\geqslant 0}$, (2) the intersection of any two elements of $\Psi_{q,\sigma}^{\lambda^c}$ should be $\emptyset$.

Hence, for every state $q \in Q$ of a CRTA $\mathbb{A} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$, Ψ_q^Σ is equal to Σ. Each $\Psi_{q,\sigma}^{\lambda^c}$ is a partition of $\mathbb{R}_{\geqslant 0}$ for every $q \in Q, \sigma \in \Sigma$.

Given a DRTA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$, the corresponding CRTA can be constructed as follows: (1) augment Q with a "sink" state $q_s \notin Q$, and q_s is not an accepting state; (2) for every $q \in Q$, let (q, σ, q_s) be a new transition with $\lambda^c(q, \sigma, q_s) = [0, +\infty)$ for every $\sigma \in \Sigma \setminus \Psi_q^\Sigma$; (3) for every $q \in Q$ and $\sigma \in \Sigma$, let (q, σ, q_s) be a new transition with $\lambda^c(q, \sigma, q_s) = \mathbb{R}_{\geqslant 0} \setminus \bigcup_{I \in \Psi_{q,\sigma}^{\lambda^c}} I$, if $\bigcup_{I \in \Psi_{q,\sigma}^{\lambda^c}} I \neq \mathbb{R}_{\geqslant 0}$.

Example 2. Figure 1 shows a DRTA $\mathcal{A}$ and the corresponding CRTA $\mathbb{A}$. $q_s = q_2$ is the "sink" state which is not accepting. The blue transitions are added by operation (2) and the red transitions are added by operation (3). For the DRTA $\mathcal{A}$, $\Psi_{q_0}^\Sigma = \{a, b\} = \Sigma$ and $\Psi_{q_1}^\Sigma = \{b\} \neq \{a, b\} = \Sigma$. For the state q_1, let (q_1, a, q_2) be a new transition and $\lambda^c(q_1, a, q_2) = [0, +\infty)$. We get the blue transition from q_1 to q_2. After conducting the operation (2) on the new state q_2, we generate the two blue self transitions on state q_2. For the states q_0, q_1 and q_2, $\Psi_{q_0,a}^{\lambda^c} = \{(5, 7)\}$, $\Psi_{q_0,b}^{\lambda^c} = \{[2, 4)\}$, $\Psi_{q_1,a}^{\lambda^c} = \{[0, +\infty)\}$, $\Psi_{q_1,b}^{\lambda^c} = \{[0, +\infty)\}$, $\Psi_{q_2,a}^{\lambda^c} = \{[0, +\infty)\}$, and $\Psi_{q_2,b}^{\lambda^c} = \{[0, +\infty)\}$. Because $\bigcup_{I \in \Psi_{q_0,a}^{\lambda^c}} I = (5, 7) \neq [0, +\infty)$, let (q_0, a, q_2) be a new transition and $\lambda^c(q_0, a, q_2) = [0, 5] \cup [7, +\infty)$. Because $\bigcup_{I \in \Psi^{\lambda^c}_{q_0,b}} I = [2, 4) \neq [0, +\infty)$, let (q_0, b, q_2) be a new transition and $\lambda^c(q_0, b, q_2) = [0, 2) \cup [4, +\infty)$. We get the two red transitions from q_0 to q_2.

Theorem 1. Given a DRTA $\mathcal{A}$, there is a CRTA $\mathbb{A}$ such that $\mathcal{L}(\mathbb{A}) = \mathcal{L}(\mathcal{A})$.

Proof. With the above transformation process from a DRTA to a CRTA, the proof is straightforward.

3.2 Membership query and real-time observation table

In this subsection, we introduce the membership query for timed words and adapt the observation table to the real-time setting. It is similar to the situation in L^* algorithm except the additional notion of timed words and fewer restrictions on the boundary R.

For simplicity, all time values in timed words are local time values in the remainder of this paper. In RTAs cases, the mutual conversion between global-timed words and local-timed words is easily achieved, as described in Subsection 2.2. The learner can construct a unique local-timed word for any global-timed word which she wants to query. We also suppose that the counterexamples given by the teacher are also local-timed words.

To gather enough information to construct a hypothesis, the learner makes membership queries like "Is the timed word $\boldsymbol{\omega}$ in target language $\mathcal{L}$?". In practice, a membership query is often conducted by testing. In theory, we assume that the teacher has an oracle to answer the membership queries. In this paper, when the learner asks whether timed words $\boldsymbol{\omega} = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$ is in target language $\mathcal{L}$, the teacher gives a positive answer if there is a run

$$\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n$$

and q_n is a final state. Otherwise, the teacher gives a negative answer. Information gathered by membership queries is stored in a real-time observation table $\boldsymbol{T}$ defined below.

Definition 6 (Real-time observation table)**.** A real-time observation table for an RTA $\mathcal{A}$ is a 7-tuple $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \text{row})$, where Σ is a finite alphabet; $\boldsymbol{\Sigma} = \Sigma \times \mathbb{R}_{\geqslant 0}$ is the infinite set of timed actions; $\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E} \subset \boldsymbol{\Sigma}^*$ are three finite sets, $\boldsymbol{S}$ is called the set of prefixes, $\boldsymbol{R}$ is called the boundary, and $\boldsymbol{E}$ is called the set of suffixes; f and row are two functions.

- $\boldsymbol{S}, \boldsymbol{R}$ are disjoint: $\boldsymbol{S} \cup \boldsymbol{R} = \boldsymbol{S} \uplus \boldsymbol{R}$;
- The empty timed word $\boldsymbol{\epsilon} \in \boldsymbol{E}$ and $\boldsymbol{\epsilon} \in \boldsymbol{S}$;
- $f : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E} \to \{-, +\}$ is a classification function such that for a timed word $\boldsymbol{\omega} \cdot \boldsymbol{e} \in (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E}$, $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = -$ if $\boldsymbol{\omega} \cdot \boldsymbol{e} \notin L(\mathcal{A})$, and $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = +$ if $\boldsymbol{\omega} \cdot \boldsymbol{e} \in L(\mathcal{A})$;
- row : a function that returns the vector of $f(\boldsymbol{\omega} \cdot \boldsymbol{e})$ indexed by $\boldsymbol{e} \in \boldsymbol{E}$ for $\boldsymbol{\omega} \in \boldsymbol{S} \cup \boldsymbol{R}$.

The elements in the sets $\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}$ are timed words and $\boldsymbol{R}$ has no restriction that $\boldsymbol{s} \cdot \boldsymbol{\sigma} \in \boldsymbol{R}$ for all $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$. Actually, the set $\boldsymbol{\Sigma}$ is an infinite set. The learner cannot query and add all timed words $\boldsymbol{s} \cdot \boldsymbol{\sigma}$ for $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ to the table as in the setting of the L^* observation table.

Before constructing a hypothesis $\mathcal{H}$ based on the real-time observation table $\boldsymbol{T}$, the learner must ensure that the table $\boldsymbol{T}$ satisfies five conditions:

- Closed if $\forall \boldsymbol{r} \in \boldsymbol{R}$, $\exists \boldsymbol{s} \in \boldsymbol{S}$ such that $\text{row}(\boldsymbol{s}) = \text{row}(\boldsymbol{r})$.
- Reduced if $\forall \boldsymbol{s_1}, \boldsymbol{s_2} \in \boldsymbol{S}$, $\text{row}(\boldsymbol{s_1}) \neq \text{row}(\boldsymbol{s_2})$.
- Consistent if $\forall \boldsymbol{\omega_1}, \boldsymbol{\omega_2} \in \boldsymbol{S} \cup \boldsymbol{R}$, $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$, $\boldsymbol{\omega_1} \cdot \boldsymbol{\sigma}, \boldsymbol{\omega_2} \cdot \boldsymbol{\sigma} \in \boldsymbol{S} \cup \boldsymbol{R}$ and $\text{row}(\boldsymbol{\omega_1}) = \text{row}(\boldsymbol{\omega_2})$, then $\text{row}(\boldsymbol{\omega_1} \cdot \boldsymbol{\sigma}) = \text{row}(\boldsymbol{\omega_2} \cdot \boldsymbol{\sigma})$.
- Prefix-closed if $\boldsymbol{S} \cup \boldsymbol{R}$ is a prefix-closed set.
- Evidence-closed if $\forall \boldsymbol{s} \in \boldsymbol{S}$ and $\forall \boldsymbol{e} \in \boldsymbol{E}$, $\boldsymbol{s} \cdot \boldsymbol{e} \in \boldsymbol{S} \cup \boldsymbol{R}$.

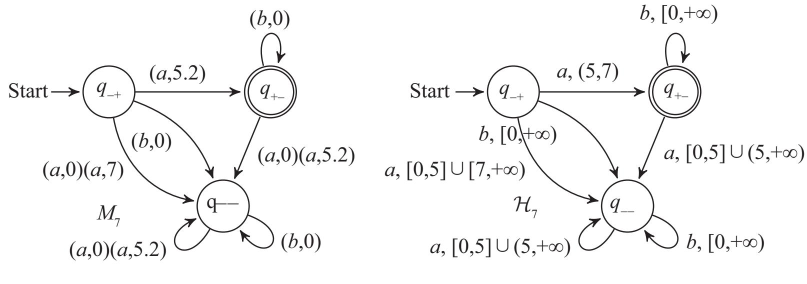
The operations to make the table closed, evidence-closed, prefix-closed, and consistent are as follows.

Making $\boldsymbol{T}$ closed. If the table $\boldsymbol{T}$ is not closed, there is some $\boldsymbol{r} \in \boldsymbol{R}$ such that $\text{row}(\boldsymbol{r})$ is different from $\text{row}(\boldsymbol{s})$ for all $\boldsymbol{s} \in \boldsymbol{S}$. The learner need to move the $\boldsymbol{r}$ from $\boldsymbol{R}$ to $\boldsymbol{S}$. What's more, $\boldsymbol{r} \cdot \boldsymbol{\sigma}$ should be added to $\boldsymbol{R}$, where $\boldsymbol{\sigma} = (\sigma, 0)$ for all $\sigma \in \Sigma$. The operation adding $\boldsymbol{r} \cdot \boldsymbol{\sigma}$ to $\boldsymbol{R}$ is important because it guarantees to deal with all actions $\sigma \in \Sigma$ for every state like the operation of L^* algorithm and give a bottom value 0 to the time value of the timed actions. It helps to form a precondition of the partition functions which we will describe at Subsection 3.3.

Making $\boldsymbol{T}$ evidence-closed. If the table $\boldsymbol{T}$ is not evidence-closed, the learner needs to add $\boldsymbol{s} \cdot \boldsymbol{e}$ to $\boldsymbol{R}$ for all $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{e} \in \boldsymbol{E}$, if $\boldsymbol{s} \cdot \boldsymbol{e} \notin \boldsymbol{S} \cup \boldsymbol{R}$. After that, the learner fills the table using membership queries.

$\boldsymbol{T}_7$	ϵ	$(a,5.2)$
ϵ	$-$	$+$
$(a,5.2)$	$+$	$-$
$(a,0)$	$-$	$-$
$(b,0)$	$-$	$-$
$(a,5.2)(a,0)$	$-$	$-$
$(a,5.2)(b,0)$	$+$	$-$
$(a,7)$	$-$	$-$
$(a,0)(a,5.2)$	$-$	$-$
$(a,5.2)(a,5.2)$	$-$	$-$
$(a,0)(a,0)$	$-$	$-$
$(a,0)(b,0)$	$-$	$-$

(a)



(b) (c)

Figure 2 The real-time observation table $\boldsymbol{T}_7$, the corresponding DFA M_7, and the hypothesis $\mathcal{H}_7$ in Examples 3 and 4.

Making $\boldsymbol{T}$ prefix-closed. If the table $\boldsymbol{T}$ is not prefix-closed, the learner should add any necessary prefixes of $\boldsymbol{\omega} \in \boldsymbol{S} \cup \boldsymbol{R}$ to $\boldsymbol{R}$ so that $\boldsymbol{S} \cup \boldsymbol{R}$ is prefix-closed. The learner also needs to fill the extended observation table by asking membership queries.

Making $\boldsymbol{T}$ consistent. If the table $\boldsymbol{T}$ is not consistent, one inconsistency is resolved by adding $\boldsymbol{\sigma} \cdot \boldsymbol{e}$ to $\boldsymbol{E}$ through finding two timed words $\boldsymbol{\omega_1}, \boldsymbol{\omega_2} \in \boldsymbol{S} \cup \boldsymbol{R}$ and $\boldsymbol{\omega_1} \cdot \boldsymbol{\sigma}, \boldsymbol{\omega_1} \cdot \boldsymbol{\sigma} \in \boldsymbol{S} \cup \boldsymbol{R}$ for some $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ such that $\mathrm{row}(\boldsymbol{\omega_1}) = \mathrm{row}(\boldsymbol{\omega_2})$ but $\mathrm{row}(\boldsymbol{\omega_1} \cdot \boldsymbol{\sigma}) \neq \mathrm{row}(\boldsymbol{\omega_2} \cdot \boldsymbol{\sigma})$, and utilizing a timed word $\boldsymbol{e} \in \boldsymbol{E}$ such that $f(\boldsymbol{\omega_1}\boldsymbol{\sigma} \cdot \boldsymbol{e}) \neq f(\boldsymbol{\omega_2}\boldsymbol{\sigma} \cdot \boldsymbol{e})$. After that, the learner fills the table by membership queries.

An reduced table will be guaranteed by the above operations and the counterexample processing described in Subsection 3.4. If the table satisfies the five conditions, we call the table prepared. A table may need several rounds to conduct the operations before it is prepared, because inconsistencies and unclosed conditions may not be solved at once according to the above operations.

3.3 Hypothesis construction

When the real-time observation table $\boldsymbol{T}$ is prepared, a hypothesis can be generated. Hypothesis construction is divided into two steps. The learner first attempts to build a DFA $M = (Q_M, \Sigma_M, \delta_M, q_0, F_M)$ based on the information in the table. Then the learner transforms M to an hypothesis $\mathcal{H}$.

Given a prepared real-time observation table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \mathrm{row})$, the learner builds a DFA $M = (Q_M, \Sigma_M, \delta_M, q_0, F_M)$ as follows:

- $Q_M = \{q_{\mathrm{row}(\boldsymbol{s})} | \boldsymbol{s} \in \boldsymbol{S}\}$;
- The initial state $q_0 = q_{\mathrm{row}(\boldsymbol{\epsilon})}$ for $\boldsymbol{\epsilon} \in \boldsymbol{S}$;
- The set of accepting states $F = \{q_{\mathrm{row}(\boldsymbol{s})} \,|\, f(\boldsymbol{s} \cdot \boldsymbol{\epsilon}) = +$ for $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{\epsilon} \in \boldsymbol{E}\}$;
- If $\boldsymbol{\omega} \cdot \boldsymbol{\sigma} \in \boldsymbol{S} \cup \boldsymbol{R}$ for $\boldsymbol{\omega} \in \boldsymbol{\Sigma}^*$ and $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$, then $\boldsymbol{\sigma} \in \Sigma_M$;
- If $\boldsymbol{\omega} \cdot \boldsymbol{\sigma} \in \boldsymbol{S} \cup \boldsymbol{R}$ for $\boldsymbol{\omega} \in \boldsymbol{\Sigma}^*$ and $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$, then $(q_{\mathrm{row}(\boldsymbol{\omega})}, \boldsymbol{\sigma}, q_{\mathrm{row}(\boldsymbol{\omega} \cdot \boldsymbol{\sigma})}) \in \delta_M$.

Example 3. Consider the table $\boldsymbol{T}_7$ in Figure 2, we describe the construction of the DFA $M_7 = (Q_{M_7}, \Sigma_{M_7}, \delta_{M_7}, q_0, F_{M_7})$. Because there are three timed words $\boldsymbol{\epsilon}$, $(a, 5.2)$ and $(a, 0)$ in $\boldsymbol{S}$, the states set $Q_{M_7} = \{q_{-+}, q_{+-}, q_{--}\}$ for $\mathrm{row}(\boldsymbol{\epsilon}) = -+$, $\mathrm{row}((a, 5.2)) = +-$ and $\mathrm{row}((a, 5.0)) = --$. The initial state $q_0 = q_{-+}$; the set of accepting states $F_{M_7} = \{q_{+-}\}$ as $f((a, 5.2) \cdot \boldsymbol{\epsilon}) = +$; the alphabet $\Sigma_{M_7} = \{(a, 0), (a, 5.2), (a, 7), (b, 0)\}$; and the transition relation $\delta_{M_7} = \{(q_{-+}, (a, 5.2), q_{+-}),\ (q_{-+}, (a, 0), q_{--}),$ $(q_{-+}, (a, 7), q_{--}), (q_{-+}, (b, 0), q_{--}), (q_{+-}, (a, 0), q_{--}), (q_{+-}, (b, 0), q_{+-}), (q_{--}, (a, 0), q_{--}), (q_{--}, (a, 5.2),$ $q_{--}),\ (q_{--}, (b, 0), q_{--})\}$. We combine some transitions if they have the same source state, the same action in Σ and the same target state.

Lemma 1. Given a prepared real-time observation table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \mathrm{row})$, the constructed DFA $M = (Q_M, \Sigma_M, \delta_M, q_0, F_M)$ preserves the condition that it accepts the timed word $\boldsymbol{\omega} \cdot \boldsymbol{e}$ for $\forall \boldsymbol{\omega} \cdot \boldsymbol{e} \in (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E}$ if $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = +$ and does not accept any timed word $\boldsymbol{\omega} \cdot \boldsymbol{e}$ for $\forall \boldsymbol{\omega} \cdot \boldsymbol{e} \in (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E}$ if $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = -$.

Proof. Given a timed word $\boldsymbol{\omega} \in \boldsymbol{S} \cup \boldsymbol{R}$, there are two conditions: $\boldsymbol{\omega} \in \boldsymbol{S}$ or $\boldsymbol{\omega} \in \boldsymbol{R}$. For the first condition, if $\boldsymbol{\omega} \in \boldsymbol{S}$, then $\boldsymbol{\omega} \cdot \boldsymbol{e} \in \boldsymbol{S} \cup \boldsymbol{R}$ for $\forall \boldsymbol{e} \in \boldsymbol{E}$ because the table $\boldsymbol{T}$ is evidence-closed. In other words, there is a timed words $\boldsymbol{\omega}' \in \boldsymbol{S} \cup \boldsymbol{R}$ such that $\boldsymbol{\omega}' = \boldsymbol{\omega} \cdot \boldsymbol{e}$. If $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = +$, then $f(\boldsymbol{\omega}' \cdot \boldsymbol{\epsilon}) = +$ which means that the $\boldsymbol{\omega}'$ ends in $q_{\mathrm{row}(\boldsymbol{\omega}')} \in F_M$. Therefore the constructed DFA M accepts $\boldsymbol{\omega}' = \boldsymbol{\omega} \cdot \boldsymbol{e}$. If $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = -$, then $f(\boldsymbol{\omega}' \cdot \boldsymbol{\epsilon}) = -$ which means that the $\boldsymbol{\omega}'$ ends in $q_{\mathrm{row}(\boldsymbol{\omega}')} \notin F_M$. Therefore the constructed DFA M does not accept $\boldsymbol{\omega} \cdot \boldsymbol{e}$. For the second condition, if $\boldsymbol{\omega} \in \boldsymbol{R}$, then there is a $\boldsymbol{\omega}' \in \boldsymbol{S}$ with that $\mathrm{row}(\boldsymbol{\omega}') = \mathrm{row}(\boldsymbol{\omega})$ because the table $\boldsymbol{T}$ is closed. $\mathrm{row}(\boldsymbol{\omega}') = \mathrm{row}(\boldsymbol{\omega})$ ensures $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = f(\boldsymbol{\omega}' \cdot \boldsymbol{e})$ for $\forall \boldsymbol{e} \in \boldsymbol{E}$. Then we find a $\boldsymbol{\omega}' \in \boldsymbol{S}$ to represent $\boldsymbol{\omega}$ and it comes to the first condition.

After constructing the DFA M, the learner transforms M to an RTA hypothesis $\mathcal{H} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$. The set of states Q of the hypothesis is equal to the set of states Q_M of M. According to the operations making the table prepared (i.e., we add $\boldsymbol{r} \cdot \boldsymbol{\sigma}$ to $\boldsymbol{R}$, where $\boldsymbol{\sigma} = (\sigma, 0)$ for all $\sigma \in \Sigma$, if $\boldsymbol{r}$ is moved to $\boldsymbol{S}$), we can ensure that $\Psi_q^{\Sigma} = \Sigma$ for all $q \in Q_M$. Then we need to compute each set $\Psi_{q,\sigma}^{\lambda^c}$ for all $q \in Q_M$ and $\sigma \in \Sigma$.

Every action $\boldsymbol{\sigma}$ in Σ_M is a timed word (σ, τ), where $\sigma \in \Sigma$ and $\tau \in \mathbb{R}_{\geqslant 0}$. $\Psi_{q,\sigma} = \{\tau \mid q_1 = q \wedge \sigma' = \sigma \text{ for } (q_1, (\sigma', \tau), q_2) \in \delta_M\}$ is the set of time values for a state $q \in Q_M$ and a word $\sigma \in \Sigma$. We can define a partition function P^c which maps the time values in $\Psi_{q,\sigma}$ to several intervals under continuous-time semantics (a partition function P^d is also defined in discrete-time semantics). These intervals form the partition $\Psi_{q,\sigma}^{\lambda^c}$.

Definition 7 (Partition function in continuous-time semantics)**.** Given a monotone increasing list $\ell = \tau_0, \tau_1, \ldots, \tau_n$, where $\tau_0 = 0, \tau_i \in \mathbb{R}_{>0}$ for $1 \leqslant i \leqslant n$, and $\lfloor \tau_i \rfloor \neq \lfloor \tau_j \rfloor$ if $\tau_i, \tau_j \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ and $i \neq j$ for $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n$, the partition function $P^c(\ell) = \Psi = I_0, I_1, I_2, \ldots, I_n$, where $I_i \in 2^{\mathbb{R}_{\geqslant 0}}$ for $0 \leqslant i \leqslant n$ such that:

- $\bigcup_{I_i \in \Psi} I_i = [0, +\infty)$;
- $I_i \cap I_j = \emptyset$ if $i \neq j$ for $0 \leqslant i \leqslant n, 0 \leqslant j \leqslant n$;
- $\tau_i \in I_i$ for $0 \leqslant i \leqslant n$;
- for $0 \leqslant i \leqslant n$,

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

Definition 8 (Partition function in discrete-time semantics)**.** Given a monotone increasing list $\ell = \tau_0, \tau_1, \ldots, \tau_n$, where $\tau_0 = 0, \tau_i \in \mathbb{N}_{>0}$ for $1 \leqslant i \leqslant n$, the partition function $P^d(\ell) = \Psi = I_0, I_1, I_2, \ldots, I_n$, where $I_i \in 2^{\mathbb{N}}$ for $0 \leqslant i \leqslant n$ such that:

- $\bigcup_{I_i \in \Psi} I_i = [0, +\infty)$;
- $I_i \cap I_j = \emptyset$ if $i \neq j$ for $0 \leqslant i \leqslant n, 0 \leqslant j \leqslant n$;
- $\tau_i \in I_i$ for $0 \leqslant i \leqslant n$;
- for $0 \leqslant i \leqslant n$, $I_i = [\tau_i, \tau_{i+1} - 1]$ or $[\tau_n, +\infty)$.

Note that the two definitions of partition functions are modified from the paper [18] in order to adapt to both continuous-time semantics and discrete-time semantics.

For every $q \in Q_M$ and $\sigma \in \Sigma$, we can generate a set $\Psi_{q,\sigma} = \{\tau \mid q_1 = q \wedge \sigma' = \sigma \text{ for } (q_1, (\sigma', \tau), q_2) \in \delta_M\}$ and a monotone increasing list $\ell_{q,\sigma} = \text{Quicksort}(\Psi_{q,\sigma})$. $\ell_{q,\sigma} = \tau_0, \tau_1, \tau_2, \ldots, \tau_n$ satisfies the preconditions of the partition function P^c owing to the operations making the real-time observation table prepared and the refinement function described in Subsection 3.4.

Now we can transform a DFA M to a hypothesis $\mathcal{H}$ as follows. The states set Q is equal to Q_M as we described before. The initial state q_0 and the set of accepting states F are also equal to the corresponding items of M respectively. For every $\ell_{q,\sigma} = \tau_0, \tau_1, \tau_2, \ldots, \tau_n$, $\Psi_{q,\sigma}^{\lambda^c} = P^c(\ell_{q,\sigma})$. For every $(q_1, (\sigma', \tau), q_2) \in \delta_M$, let $(q_1, \sigma', q_2) \in \Delta$ be a new transition with $\lambda^c(q_1, \sigma', q_2) = I$ if $q_1 = q$, $\sigma' = \sigma$, $\tau \in I$ where $I \in \Psi_{q,\sigma}^{\lambda^c} = P^c(\ell_{q,\sigma})$.

Example 4. In Figure 2, for the DFA $M_7 = (Q_{M_7}, \Sigma_{M_7}, \delta_{M_7}, q_0, F_{M_7})$, we transform it to a hypothesis $\mathcal{H}_7 = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$. The set of states Q, the initial state q_0 and the set of accepting states F are equal to the corresponding items of M_7, respectively. For the state q_{-+}, $\Psi_{q_{-+}}^{\Sigma} = \{a, b\}$ and $\Psi_{q_{-+},\sigma} = \{0, 7, 5.2\}$, so $\ell_{q_{-+},\sigma} = 0, 5.2, 7$ and $\Psi_{q_{-+},\sigma}^{\lambda^c} = P^c(\ell_{q_{-+},\sigma}) = \{[0, 5], (5, 7), [7, +\infty)\}$. Then for the transition $(q_{-+}, (a, 0), q_{--}) \in \delta_{M_7}$, let (q_{-+}, a, q_{--}) be a new transition with $\lambda^c(q_{-+}, a, q_{--}) = [0, 5]$. For the transition $(q_{-+}, (a, 5.2), q_{--})$, let (q_{-+}, a, q_{+-}) be a new transition with $\lambda^c(q_{-+}, a, q_{+-}) = (5, 7)$. For the transition $(q_{-+}, (a, 7), q_{--})$, let (q_{-+}, a, q_{--}) be a new transition with $\lambda^c(q_{-+}, a, q_{--}) = [7, +\infty)$. Note that we combine the first and third new transitions. With the same methods, we can finish the transformation.

Lemma 2. Given a DFA $M = (Q_M, \Sigma_M, \delta_M, q_0, F_M)$ which is generated from a prepared real-time table $\boldsymbol{T}$, if a hypothesis RTA $\mathcal{H} = (Q, \Sigma, \Delta, q_0, F, \lambda^c)$ is transformed from M, then $\mathcal{H}$ preserves that it accepts the timed word $\boldsymbol{\omega} \cdot \boldsymbol{e}$ for all $\boldsymbol{\omega} \cdot \boldsymbol{e} \in (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E}$ if $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = +$ and does not accept any timed word $\boldsymbol{\omega} \cdot \boldsymbol{e}$ for all $\boldsymbol{\omega} \cdot \boldsymbol{e} \in (\boldsymbol{S} \cup \boldsymbol{R}) \cdot \boldsymbol{E}$ if $f(\boldsymbol{\omega} \cdot \boldsymbol{e}) = -$.

Proof. Given a DFA M, we map the time values in the timed actions to time value intervals. For a transition $(q_1,(\sigma_1,\tau_1),q_2)\in\delta_M$, we build a corresponding transition (q_1,σ_1,q_2) with $\lambda^c(q_1,\sigma_1,q_2)=I$ and $\tau_1\in I\in\Psi^{\lambda^c}_{q,\sigma}$ in the hypothesis $\mathcal{H}$. Given a timed words $(\sigma_1,\tau_1)\cdots(\sigma_n,\tau_n)$, $\mathcal{H}$ accepts the timed words if M accepts it and vice versa. With Lemma 1, $\boldsymbol{\omega}\cdot\boldsymbol{e}$ ends in an accepting state in F_M if $f(\boldsymbol{\omega}\cdot\boldsymbol{e})=+$. Hence, $\mathcal{H}$ accepts the timed words $\boldsymbol{\omega}\cdot\boldsymbol{e}$ if $f(\boldsymbol{\omega}\cdot\boldsymbol{e})=+$. The same reasoning process for the condition $f(\boldsymbol{\omega}\cdot\boldsymbol{e})=-$ is omitted.

Theorem 2. A hypothesis RTA $\mathcal{H}$ is a CRTA.

Proof. When we move an element $\boldsymbol{r}\in\boldsymbol{R}$ to $\boldsymbol{S}$, we add timed words $\boldsymbol{r}\cdot(\sigma,0)$ for every $\sigma\in\Sigma$ to $\boldsymbol{R}$. It helps to distribute the actions in alphabet Σ to the transitions of which the source state is $q_{\mathrm{row}(\boldsymbol{r})}$. It ensures that $\Psi_{q_{\mathrm{row}(\boldsymbol{r})},\sigma}=\Sigma$. The partition function guarantees that $\Psi^{\lambda^c}_{q_{\mathrm{row}(\boldsymbol{r})},\sigma}$ is a partition of $\mathbb{R}_{\geqslant 0}$ for each $\sigma\in\Sigma$. Hence, the hypothesis $\mathcal{H}$ satisfies the CRTA definition in Subsection 3.1.

3.4 Equivalence query and counterexample processing

Now we introduce the equivalence query and the counterexample processing. The learner submits a hypothesis $\mathcal{H}$ to the teacher for an equivalence query "Is the recognized language $\mathcal{L}(\mathcal{H})$ equal to the target language $\mathcal{L}$?". In practice, teachers with complete knowledge of the target language are often not available, so other methods (such as conformance testing [26]) are used. In theory, just like the L^* algorithm, the teacher is assumed to have an oracle to easily answer the question and to give a counterexample if the answer is negative. In this paper, the oracle knows exactly the DRTA $\mathcal{A}$ which recognizes the target language and has the capacity to answer the language-equivalence problem whether $\mathcal{L}(\mathcal{H})=\mathcal{L}(\mathcal{A})$.

According to Theorem 1, there exists a CRTA $\mathbb{A}$ such that $\mathcal{L}(\mathbb{A})=\mathcal{L}(\mathcal{A})$. Hence, the language equivalence problem whether $\mathcal{L}(\mathcal{H})=\mathcal{L}(\mathcal{A})$ can be converted to the problem whether $\mathcal{L}(\mathcal{H})=\mathcal{L}(\mathbb{A})$, where $\mathcal{H}$ and $\mathbb{A}$ are two CRTAs. This can be divided into two language inclusion problems whether $\mathcal{L}(\mathcal{H})\subseteq\mathcal{L}(\mathbb{A})$ and $\mathcal{L}(\mathbb{A})\subseteq\mathcal{L}(\mathcal{H})$. The most of decision procedures for language inclusion proceed by complementation and emptiness checking of the intersection [27]: $\mathcal{L}(A)\subseteq\mathcal{L}(B)$ iff $\mathcal{L}(A)\cap\overline{\mathcal{L}(B)}=\emptyset$. There is an important result that the language inclusion problem of timed automata with one clock is decidable by converting it to a reachability problem on an infinite graph [28]. So the language inclusion problem of real-time automata is decidable. But we also know that the timed automata with a single clock cannot be complemented [19]. However, real-time automata can be complemented [13,29]. Hence, for real-time automata, we can decide the language inclusion problem by complementation and emptiness checking of the intersection. A timed word $\boldsymbol{\omega}\in\mathcal{L}(\mathcal{H})\cap\overline{\mathcal{L}(\mathbb{A})}$ is a negative counterexample (i.e., $\mathrm{ctx}_-=(\boldsymbol{\omega},-)$) if $\mathcal{L}(\mathcal{H})\cap\overline{\mathcal{L}(\mathbb{A})}\neq\emptyset$ and a timed word $\boldsymbol{\omega}'\in\overline{\mathcal{L}(\mathcal{H})}\cap\mathcal{L}(\mathbb{A})$ is a positive counterexample (i.e., $\mathrm{ctx}_+=(\boldsymbol{\omega}',+)$) if $\overline{\mathcal{L}(\mathcal{H})}\cap\mathcal{L}(\mathbb{A})\neq\emptyset$. The teacher gives a positive answer (i.e., YES) for the equivalence query if $\mathcal{L}(\mathcal{H})\cap\overline{\mathcal{L}(\mathbb{A})}=\emptyset$ and $\overline{\mathcal{L}(\mathcal{H})}\cap\mathcal{L}(\mathbb{A})=\emptyset$. Otherwise, the teacher gives a negative answer (i.e., NO) with a counterexample ctx either positive or negative. The algorithm for the equivalence query is described in Algorithm 1.

Algorithm 1 equivalence_query($\mathcal{H}$)

Input: a hypothesis $\mathcal{H}$.
Output: equivalent : a Boolean value to identify whether $\mathcal{L}(\mathcal{H})=\mathcal{L}(\mathbb{A})$, where CRTA $\mathbb{A}$ recognizes the target language;
ctx : a counterexample.

```
equivalent ← false; ctx ← ϵ;
flag_−, flag_+ ← true;
if L(H) ∩ ¬L(A) ≠ ∅ then
    flag_− ← false;
    Select a timed word ω from L(H) ∩ ¬L(A); //Negative counterexample
    ctx_− ← (ω, −);
end if
if ¬L(H) ∩ L(A) ≠ ∅ then
    flag_+ ← false;
    Select a timed word ω′ from ¬L(H) ∩ L(A); //Positive counterexample
    ctx_+ ← (ω′, +);
end if
equivalent ← flag_− ∧ flag_+;
if equivalent = false then
    ctx ← select a counterexample from ctx_+ and ctx_−;
end if
return equivalent, ctx.
```

In lines 5, 10 and 15, the teacher selects a timed word randomly and does not always need to select a counterexample with exact endpoints of the intervals. The inexact intervals of the partitions will be corrected by our partition function step by step because the teacher can always indicate the difference between the current hypothesis and the target.

When receiving a counterexample ctx $= (\boldsymbol{\omega}, +)$ or $(\boldsymbol{\omega}, -)$, where $\boldsymbol{\omega} = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, we utilize a refinement function g to normalize τ_i to a "symbolic" number $g(\tau_i)$ if $\tau_i \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ for $1 \leqslant i \leqslant n$ under continuous-time semantics.

Definition 9 (Refinement function)**.** A refinement function $g : \mathbb{R}_{\geqslant 0} \setminus \mathbb{N} \to \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ such that $g(c) = \lfloor c \rfloor + \theta$ where θ is a constant in $(0, 1)$ and $g(c_1) = g(c_2)$ if $\lfloor c_1 \rfloor = \lfloor c_2 \rfloor$ for all $c_1, c_2 \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$.

Given a constant $\theta \in (0, 1)$, for the timed word $\boldsymbol{\omega}$ with non-integer time values, we transform it to $\boldsymbol{\omega}_r = \cdots (\sigma_i, g(\tau_i)) \cdots$ if $\tau_i \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ for $1 \leqslant i \leqslant n$. The main reason of the refinement is as follows. We need to solve the conflict caused by the miss-distributions in the generated DFA where there exist two timed actions (σ, c_1) and (σ, c_2) with the same action σ and $c_1, c_2 \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ and $\lfloor c_1 \rfloor = \lfloor c_2 \rfloor$, but located on two transitions which have the same source state and different target states. Such miss-distributions also cause the violation of the precondition of the partition function, which cannot be rectified. We give an illustrated explanation in Example 5.

Theorem 3. Given a counterexample ctx $= (\boldsymbol{\omega}, +/-)$ where $\boldsymbol{\omega} = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, ctx$' = (\boldsymbol{\omega}_r, +/-)$ is also a counterexample, where $\boldsymbol{\omega}_r = \cdots (\sigma_i, g(\tau_i)) \cdots$ if $\tau_i \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ for $1 \leqslant i \leqslant n$.

Proof. We first consider a positive counterexample $(\boldsymbol{\omega}, +)$. It means that the hypothesis has a run

$$\rho = q_0 \xrightarrow[\tau_1]{\sigma_1} q_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q_n$$

with $q_n \notin F$ and the target automaton has a run

$$\rho' = q'_0 \xrightarrow[\tau_1]{\sigma_1} q'_1 \xrightarrow[\tau_2]{\sigma_2} \cdots \xrightarrow[\tau_n]{\sigma_n} q'_n$$

with $q'_n \in F'$. For each

$$q_{i-1} \xrightarrow[\tau_i]{\sigma_i} q_i,$$

where $\tau_i \in \mathbb{R}_{\geqslant 0} \setminus \mathbb{N}$ and $1 \leqslant i \leqslant n$, there exist a transition (q_{i-1}, σ_i, q_i) such that $\tau_i \in \lambda^c(q_{i-1}, \sigma_i, q_i)$ in the hypothesis and a transition $(q'_{i-1}, \sigma_i, q'_i)$ such that $\tau_i \in \lambda^c(q'_{i-1}, \sigma_i, q'_i)$ in the target automaton. Because $\lambda^c(q_{i-1}, \sigma_i, q_i)$ is a union of intervals whose endpoints are in $\mathbb{N} \cup \{+\infty\}$ and $\lfloor \tau_i \rfloor = \lfloor g(\tau_i) \rfloor$, then $g(\tau_i) \in \lambda^c(q_{i-1}, \sigma_i, q_i)$. Hence, there exists a timed action $(\sigma_i, g(\tau_i))$ such that

$$q_{i-1} \xrightarrow[g(\tau_i)]{\sigma_i} q_i$$

in the hypothesis and

$$q'_{i-1} \xrightarrow[g(\tau_i)]{\sigma_i} q'_i$$

in the target automaton. For the timed word $\boldsymbol{\omega}_r$, there exist a run

$$\rho_r = q_0 \cdots q_{i-1} \xrightarrow[g(\tau_i)]{\sigma_i} q_i \cdots q_n$$

in the hypothesis and a run

$$\rho'_r = q'_0 \cdots q'_{i-1} \xrightarrow[g(\tau_i)]{\sigma_i} q'_i \cdots q'_n$$

in the target automaton. Then ctx$' = (\boldsymbol{\omega}_r, +)$ is still a positive counterexample. The proof for a negative counterexample proceeds similarly.

Owing to Theorem 3, θ can be any number in $(0, 1)$. In the remainder of this paper, let $\theta = 0.2$.

Given a refined counterexample $(\boldsymbol{\omega}_r, +/-)$, we add all prefixes of $\boldsymbol{\omega}_r$ to $\boldsymbol{R}$ except those already in $\boldsymbol{S} \cup \boldsymbol{R}$. Note that we do not need the refinement function g under discrete-time semantics.

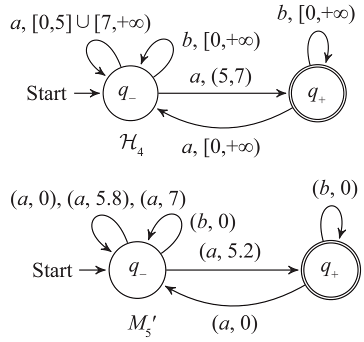
Example 5. Consider the prepared table $\boldsymbol{T}_4$ and the corresponding hypothesis $\mathcal{H}_4$ in Figure 3, the recognized language of $\mathcal{H}_4$ is not the same as that for the target automaton $\mathbb{A}$ in Figure 1. The teacher gives a counterexample $((a, 0)(a, 5.8), -)$. If we add the prefixes of $(a, 0)(a, 5.8)$ to $\boldsymbol{R}$ directly, the table $\boldsymbol{T}'_5$ is shown in Figure 3. $\boldsymbol{T}'_5$ is prepared and we build a DFA M'_5. We find that the time value 5.8 is

T_4	ϵ
ϵ	−
(a, 5.2)	+
(a, 0)	−
(b, 0)	−
(a, 5.2)(a, 0)	−
(a, 5.2)(b, 0)	+
(a, 7)	−

(a, 0)(a, 5.8), − ⟶

T_5'	ϵ
ϵ	−
(a, 5.2)	+
(a, 0)	−
(b, 0)	−
(a, 5.2)(a, 0)	−
(a, 5.2)(b, 0)	+
(a, 7)	−
(a, 0)(a, 5.8)	−

(a)



(b)

Figure 3 The new table $\boldsymbol{T}_5'$ after adding the counterexample $((a,0)(a,5.8),-)$ directly and the generated DFA M_5'.

T_1	ϵ
ϵ	−
(a, 0)	−
(b, 0)	−

(a, 5.1), + ⟹ g (5.1)=5.2 ⟹ (a, 5.2), + ⟹

T_2	ϵ
ϵ	−
(a, 0)	−
(b, 0)	−
(a, 5.2)	+

Closed ⟹

T_3	ϵ
ϵ	−
(a, 5.2)	+
(a, 0)	−
(b, 0)	−
(a, 5.2)(a, 0)	−
(a, 5.2)(b, 0)	+

(a, 7), − ⟹

T_4	ϵ
ϵ	−
(a, 5.2)	+
(a, 0)	−
(b, 0)	−
(a, 5.2)(a, 0)	−
(a, 5.2)(b, 0)	+
(a, 7)	−

⟹ (a, 0)(a, 5.8), − ⟹ g (5.8)=5.2 ⟹ (a, 0)(a, 5.2), − ⟹

T_5	ϵ
ϵ	−
(a, 5.2)	+
(a, 0)	−
(b, 0)	−
(a, 5.2)(a, 0)	−
(a, 5.2)(b, 0)	+
(a, 7)	−
(a, 0)(a, 5.2)	−

Consistent, evidence-closed ⟹

T_6	ϵ	(a, 5.2)
ϵ	−	+
(a, 5.2)	+	−
(a, 0)	−	−
(b, 0)	−	−
(a, 5.2)(a, 0)	−	−
(a, 5.2)(b, 0)	+	−
(a, 7)	−	−
(a, 0)(a, 5.2)	−	−
(a, 5.2)(a, 5.2)	−	−

Closed ⟹

T_7	ϵ	(a, 5.2)
ϵ	−	+
(a, 5.2)	+	−
(a, 0)	−	−
(b, 0)	−	−
(a, 5.2)(a, 0)	−	−
(a, 5.2)(b, 0)	+	−
(a, 7)	−	−
(a, 0)(a, 5.2)	−	−
(a, 5.2)(a, 5.2)	−	−
(a, 0)(a, 0)	−	−
(a, 0)(b, 0)	−	−

(b, 2)(a, 5.2), + ⟹

T_8	ϵ	(a, 5.2)
ϵ	−	+
(a, 5.2)	+	−
(a, 0)	−	−
(b, 0)	−	−
(a, 5.2)(a, 0)	−	−
(a, 5.2)(b, 0)	+	−
(a, 7)	−	−
(a, 0)(a, 5.2)	−	−
(a, 5.2)(a, 5.2)	−	−
(a, 0)(a, 0)	−	−
(a, 0)(b, 0)	−	−
(b, 2)	−	+
(b, 2)(a, 5.2)	+	−

(b, 4)(a, 5.2), − ⟹

T_9	ϵ	(a, 5.2)
ϵ	−	+
(a, 5.2)	+	−
(a, 0)	−	−
(b, 0)	−	−
(a, 5.2)(a, 0)	−	−
(a, 5.2)(b, 0)	+	−
(a, 7)	−	−
(a, 0)(a, 5.2)	−	−
(a, 5.2)(a, 5.2)	−	−
(a, 0)(a, 0)	−	−
(a, 0)(b, 0)	−	−
(b, 2)	−	+
(b, 2)(a, 5.2)	+	−
(b, 4)	−	−
(b, 4)(a, 5.2)	−	−

Figure 4 The real-time observation tables for the illustrative example.

miss-distributed to a wrong transition, because the timed action $(a, 5.8)$ should be accepted. Hence, the time actions $(a, 5.2)$ and $(a, 5.8)$ should be in the same transition with the source state q_-. The monotone increasing list $\ell_{q_-,a} = 0, 5.2, 5.8, 7$ violates the precondition of the partition function. It cannot be handled by our partition function. So, the whole learning process is unable to continue and the miss-distributed situation will never be solved. However, we will get a refined counterexample $((a,0)(a,5.2),-)$ by using a refinement function g with $\theta = 0.2$. The new table $\boldsymbol{T}_5$ shown in Figure 4 is not consistent, which will be solved by the operations for restoring consistency.

3.5 Learning algorithm

The initial real-time observation table is $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \text{row})$, where $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$, $\boldsymbol{E} = \{\boldsymbol{\epsilon}\}$ and $\boldsymbol{R} = \{(\sigma, 0) \mid \sigma \in \Sigma\}$. The table is filled by membership queries for timed words $\boldsymbol{\omega} \cdot \boldsymbol{e}$ where $\boldsymbol{\omega} \in (\boldsymbol{S} \cup \boldsymbol{R})$ and $\boldsymbol{e} \in \boldsymbol{E}$. If the table is not prepared, we check which conditions the table violates and conduct the corresponding operations described in Subsection 3.2. When the table is prepared, we build a hypothesis $\mathcal{H}$ and ask an equivalence query. If the answer is positive, the recognized language $\mathcal{L}(\mathcal{H})$ of the current hypothesis is equal to the target language $\mathcal{L}$. Otherwise, we receive a counterexample and conduct the counterexample processing described in Subsection 3.4. The whole procedure repeats until the teacher gives a positive answer for an equivalence query. The learning algorithm can be represented as pseudo-code in Algorithm 2. In a way analogous to [18, Theorem 1], we show the following.

Algorithm 2 Learning real-time automaton

Input: the real-time observation table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \text{row})$.
Output: the hypothesis $\mathcal{H}$ recognizing the target language.

```
S ← {ϵ}; R ← {(σ, 0) | σ ∈ Σ}; E ← {ϵ};
Fill T by membership queries;
equivalent ← false;
while equivalent = false do
    prepared ← is_prepared(T); // Whether the table is prepared
    while prepared = false do
        if T is not closed then
            make_closed(T);
        end if
        if T is not consistent then
            make_consistent(T);
        end if
        if T is not evidence-closed then
            make_evidence_closed(T);
        end if
        if T is not prefixed-closed then
            make_prefix_closed(T);
        end if
        prepared ← is_prepared(T);
    end while
    H ← build_hypothesis(T); // Constrcuting a hypothesis H
    equivalent, ctx ← equivalence_query(H);
    if equivalent = false then
        ctx_processing(T, ctx); //The counterexample processing
    end if
end while
return H.
```

Theorem 4. Algorithm 2 terminates and returns a minimal CRTA $\mathcal{H}$ which recognizes the target language.

Proof. By Lemmas 1 and 2, Theorems 2 and 3, the algorithm always constructs CRTAs as hypotheses. $\boldsymbol{S}$ indicates the states and we always add a new element to $\boldsymbol{S}$ when a new state is needed. The learning algorithm can be thought of as a product of a L^* process for the alphabet Σ and a L^* process with partition and refinement steps for the interval $\mathbb{R}_{\geqslant 0}$. Hence, the algorithm terminates and returns a CRTA which has a minimal number of states and recognizes the target language.

3.6 Illustrative example

Let us illustrate the learning process for a target language $\mathcal{L}$ defined over $\boldsymbol{\Sigma} = \Sigma \times \mathbb{R}_{\geqslant 0}$ where $\Sigma = \{a, b\}$. $\mathcal{L}$ is recognized by the DRTA $\mathcal{A}$ and is also recognized by the CRTA $\mathbb{A}$ in Figure 1. The real-time observation tables, the corresponding DFAs and hypotheses constructed during the learning process are shown in Figures 4 and 5.

We initialize the real-time observation table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{E}, f, \text{row})$ with $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$, $\boldsymbol{R} = \{(a, 0), (b, 0)\}$ and $\boldsymbol{E} = \{\boldsymbol{\epsilon}\}$ as described in Algorithm 2. We denote it as $\boldsymbol{T}_1$ in Figure 4. Fortunately, $\boldsymbol{T}_1$ is prepared. We build a DFA M_1 and transform it to a hypothesis $\mathcal{H}_1$. We make an equivalence query and get a counterexample $\text{ctx}_1 = ((a, 5.1), +)$. With the refinement function $g(5.1) = 5.2$, we get a refined counterexample $((a, 5.2), +)$. We add the prefixes of the refined counterexample to $\boldsymbol{R}$ and then get the table $\boldsymbol{T}_2$. $\boldsymbol{T}_2$ is not closed because $(a, 5.2) \in \boldsymbol{R}$ with $\text{row}((a, 5.2)) = +$ but there is no $\boldsymbol{s} \in \boldsymbol{S}$ such that

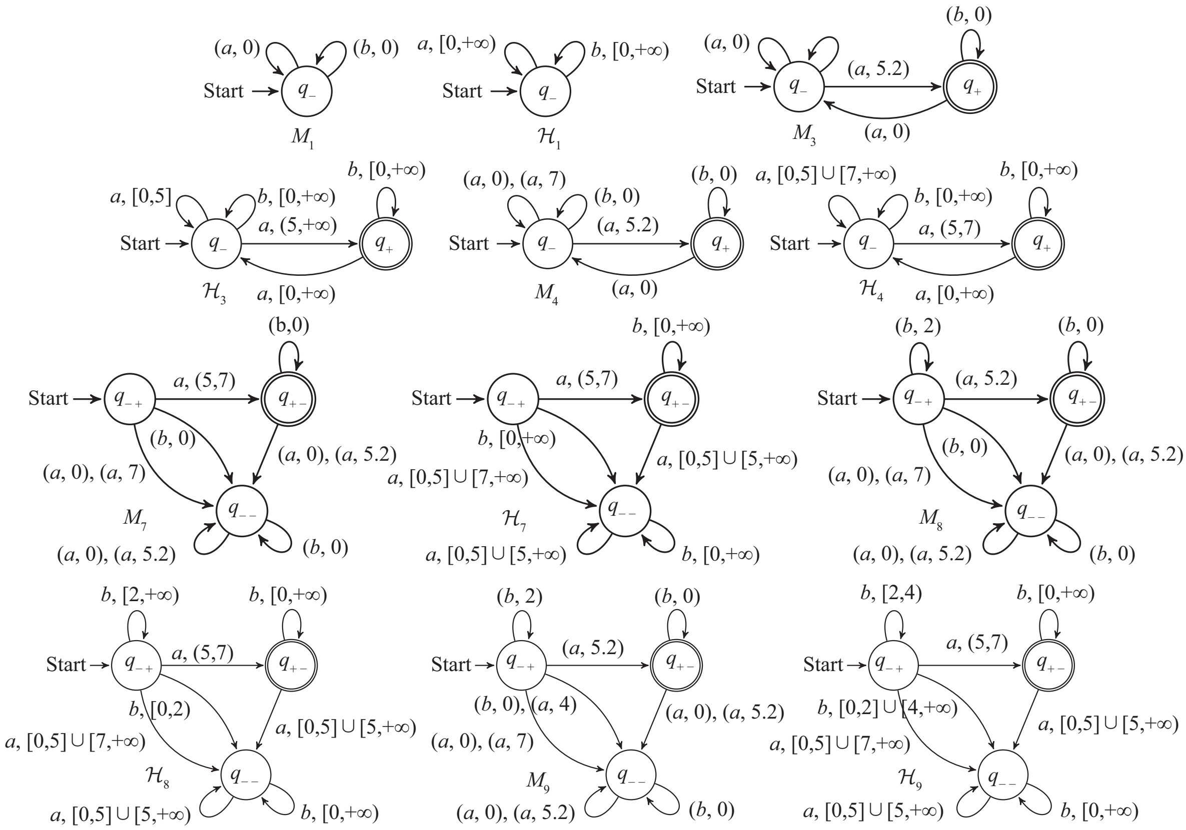


Figure 5 The DFAs and hypotheses for the illustrative example.

row($\boldsymbol{s}$) = +. We move $(a, 5.2)$ to $\boldsymbol{S}$ and add two timed words $(a, 5.2)(a, 0)$, $(a, 5.2)(b, 0)$ to $\boldsymbol{R}$. The table $\boldsymbol{T}_3$ is prepared and we build M_3 and $\mathcal{H}_3$. After an equivalence query, we add the counterexample $\text{ctx}_2 = ((a, 7), -)$ to $\boldsymbol{R}$. Because $\boldsymbol{T}_4$ is prepared, we build M_4 and $\mathcal{H}_4$. Note that we combine the transitions which have the same source state, the same action in Σ and the same target state. Hence, in M_4, there is a transition with two actions $(a, 0), (a, 7) \in \Sigma_{M_4}$. The hypothesis $\mathcal{H}_4$ cannot recognize the target language $\mathcal{L}$. After receiving a counterexample $\text{ctx}_3 = ((a, 0)(a, 5.8), -)$, we generate a refined counterexample $((a, 0)(a, 5.2), -)$. We need to add all prefixes of $(a, 0)(a, 5.2)$ to $\boldsymbol{R}$. Because the prefixes $\boldsymbol{\epsilon}$ and $(a, 0)$ have been already in $\boldsymbol{S} \cup \boldsymbol{R}$, we just add the prefix $(a, 0)(a, 5.2)$ to $\boldsymbol{R}$. The table $\boldsymbol{T}_5$ is not consistent because $\text{row}(\boldsymbol{\epsilon}) = - = \text{row}((a, 0))$ while $\text{row}(\boldsymbol{\epsilon} \cdot (a, 5.2)) = \text{row}((a, 5.2)) = + \neq - = \text{row}((a, 0) \cdot (a, 5.2))$. It means that $\boldsymbol{\epsilon}$ and $(a, 0)$ actually lead to different states and we need a new state to handle this. Because $f(\boldsymbol{\epsilon} \cdot (a, 5.2) \cdot \boldsymbol{\epsilon}) = + \neq - = f((a, 0) \cdot (a, 5.2) \cdot \boldsymbol{\epsilon})$, we add a new timed word $\boldsymbol{e} = (a, 5.2) \cdot \boldsymbol{\epsilon} = (a, 5.2)$ to $\boldsymbol{E}$ to solve the inconsistency. After adding $(a, 5.2)(a, 5.2)$ to $\boldsymbol{R}$ to make the table evidence-closed, we get the table $\boldsymbol{T}_6$. $\boldsymbol{T}_6$ is not closed because $\text{row}((a, 0)) = --$ and there is no timed word $\boldsymbol{\sigma} \in \boldsymbol{S}$ with $\text{row}(\boldsymbol{\sigma}) = --$. Hence, $(a, 0)$ is moved to $\boldsymbol{S}$. Then the table $\boldsymbol{T}_7$ is prepared. We add the prefixes $(b, 2)$ and $(b, 2)(a, 5.2)$ of the counterexample $\text{ctx}_4 = ((b, 2)(a, 5.2), +)$ to the table after an equivalence query for $\mathcal{H}_7$. $\boldsymbol{T}_8$ is also prepared. We generate the automata M_8 and $\mathcal{H}_8$. The counterexample ctx_4 just adds new evidences to approach the right partitions. After adding the prefixes of the counterexample $\text{ctx}_5 = ((b, 4)(a, 5.2), -)$ to the table, we get a prepared table $\boldsymbol{T}_9$. Finally, we get a positive answer after submitting the generated hypothesis $\mathcal{H}_9$ to the teacher. The whole process terminates and the last hypothesis $\mathcal{H}_9$ is the same as the CRTA $\mathbb{A}$ in Figure 1 after computing the unions of time intervals on two transitions (q_{--}, a, q_{--}) and (q_{+-}, a, q_{--}).

4 Complexity

Given a target language $\mathcal{L}$ which is recognized by the minimal CRTA $\mathbb{A}$, let the state sets size $|Q| = n$, the alphabet size $|\Sigma| = k$ and the maximal partition size $m \geqslant |\Psi^{\lambda^c}_{q,\sigma}|$ for $\forall q \in Q, \sigma \in \Sigma$.

In our algorithm, $\boldsymbol{S}$ indicates the states and $\boldsymbol{E}$ distinguishes the states. The number of the timed

Table 1 The information of the experiments in which the alphabet size $|\Sigma| = k = 4$ and the maximal partition size $m = 4 \geqslant |\Psi^{\lambda^c}_{q,\sigma}|$ and the number of states $|Q| = n$ ranges in $\{5, 7, 9, 11, 13, 15\}$

Case ID	$\|Q\|$	$\|\Delta\|_{\text{mean}}$	Membership				Equivalence				t_{mean}
			$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	
4_4_4	5	35.8	248	295.5	278	376	17	28.1	28	38	3.4
6_4_4	7	54.6	505	699.8	708	948	33	45.4	46	65	29.0
8_4_4	9	68.0	888	1138.2	1130	1488	40	54.0	54	66	40.7
10_4_4	11	83.7	1225	1824.6	1864	2560	50	68.4	69	90	145.1
12_4_4	13	99.6	1561	2476.8	2620	3278	64	79.9	79	97	280.0
14_4_4	15	117.6	2376	3258.7	3050	4914	78	97.9	98	114	500.1

words in $\boldsymbol{E}$ is bounded by n (actually a number between $\lceil \log_2 n \rceil$ and n).

Every counterexample helps to approach $\mathbb{A}$ in two ways: one for refining the partitions and the other for adding a new state. There are $k \times m \times n$ intervals of partitions at most. We need $O(kmn)$ equivalence queries for refining the partitions, because the teacher may not give a counterexample to identify an exact interval of a partition every time. Because the number of states is n, we need at most n equivalence queries for adding new states. Therefore, the number of equivalence queries is bounded by $O(kmn)$.

Obviously, we need at most n^2 membership queries to fill the table rows in $\boldsymbol{S}$. Depending the operation making table closed, we add at most kn timed words in $\boldsymbol{R}$. We also add $O(kmn)$ prefixes of counterexamples in $\boldsymbol{R}$. What's more, the evidence-closed operation adds $O(n^2)$ timed words in $\boldsymbol{R}$. Totally, we need $O(kn^2 + kmn^2 + n^3)$ membership queries to fill the table rows in $\boldsymbol{R}$. Therefore, the number of the membership queries is bounded by $O(kn^2 + kmn^2 + n^3)$ at most.

5 Implementation and experiments

Based on the methods reported above, we have developed a prototypical tool for learning deterministic real-time automata. The tool is implemented in Python. All of the experiments have been evaluated on a 3.6 GHz Intel Core-i7 processor with 8 GB RAM running 64-bit Ubuntu 16.04.

Our method is the first work on active learning for RTAs and guarantees to generate a correct DRTA when given a target language which can be represented by a DRTA. In Angluin's framework, if the correct model can always be learned, the evaluation for the active automaton learning is not correctness, but the number of the two kinds of queries is used to generate a correct automaton. Hence, the main goal of the experiments is to support the complexity analysis in Section 4. We randomly generate 340 DRTAs without redundant states (i.e., (1) the unreachable states from the initial state and (2) the states from which the automaton has no run to reach any final state) as the target automata. These automata are all learned by our tool successfully. The information of the experiments is compressed in three tables. Each table has 6 cases and each case includes 20 different DRTAs. The case 7_4_4 is reused in Tables 2 and 3. Every case ID is composed by three numbers. They present the state number of every DRTA in the case, the size of the alphabet $|\Sigma|$ and the maximal partition size $m \geqslant |\Psi^{\lambda^c}_{q,\sigma}|$ respectively. $|\Delta|_{\text{mean}}$ is the mean number of the transitions of the corresponding CRTAs in a case. The membership and equivalence columns contain the statistical data of membership queries and equivalence queries to learn a corresponding CRTA in a case respectively. Each of them has four elements $N_{\min}$, N_{mean}, N_{median} and $N_{\max}$ which denote the minimal number, mean number, median number and maximal number respectively. t_{mean} is the mean value of wall-clock time in seconds. We also conduct other experiments with larger scale of RTAs[2)]. Besides, it takes 0.07 s to learn the illustrative example.

In Table 1, we fix the size of the alphabet $|\Sigma| = k = 4$ and the maximal partition size $m = 4$. The number of states of the DRTAs range in set $\{4, 6, 8, 10, 12, 14\}$. These DRTAs have no redundant states. Hence the number of states $|Q| = n$ of the corresponding CRTA is in the set $\{5, 7, 9, 11, 13, 15\}$. Figure 6(a) shows the relation between the number of states $|Q| = n$ and the number of membership queries MEM_QUERY, and Figure 6(d) presents the relation between the number of states $|Q| = n$ and the number of equivalence queries EQ_QUERY. We find that the number of membership queries is bounded by $O(n^3)$ and the number of equivalence queries is bounded by $O(n)$.

In Table 2, we fix the size of the alphabet $|\Sigma| = k = 4$ and the number of states of all corresponding CRTA to $|Q| = n = 8$. The maximal partition size m ranges from 2 to 7. Figures 6(b) and (e) show that

2) More experiments can be found at the tool page: https://github.com/Leslieaj/RTALearning.

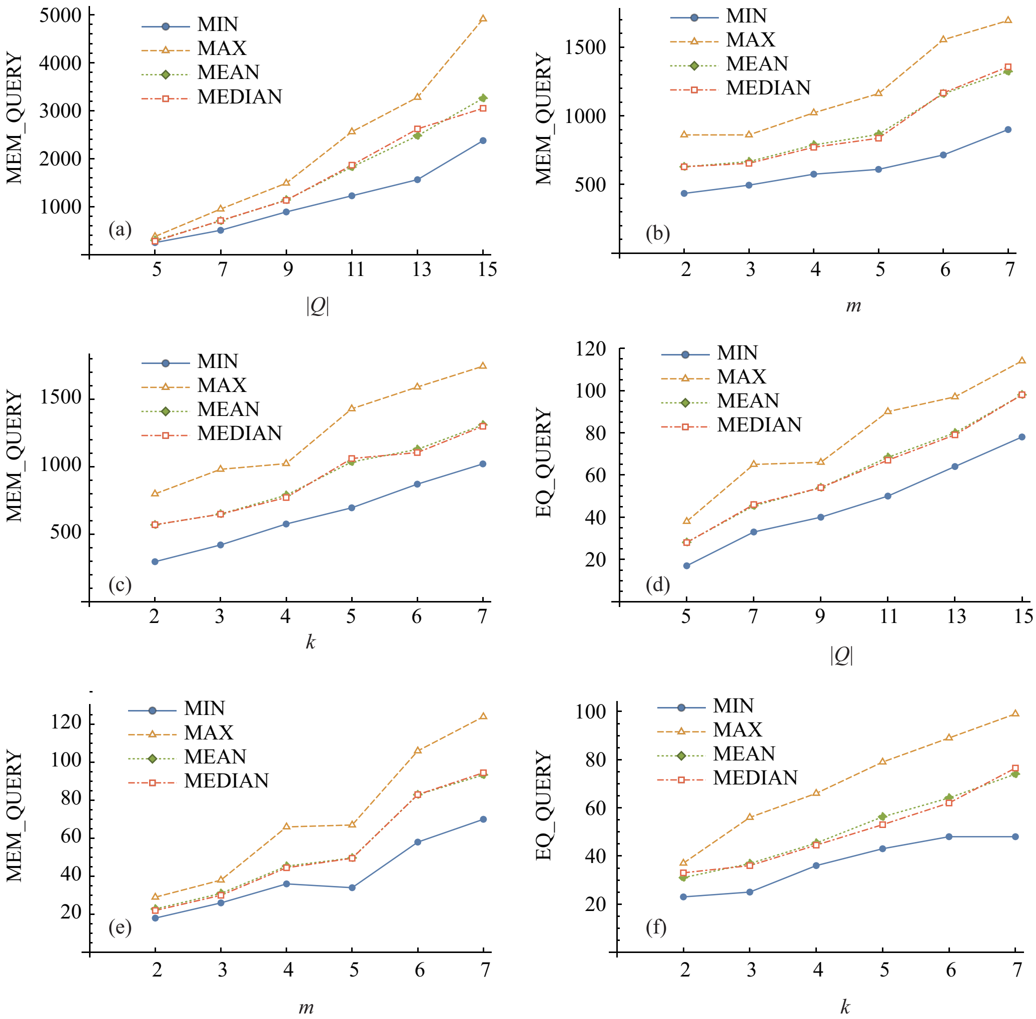


Figure 6 (Color online) (a) The relation between $|Q|$ and the number of membership queries; (b) the relation between m and the number of membership queries; (c) the relation between k and the number of membership queries; (d) the relation between $|Q|$ and the number of equivalence queries; (e) the relation between m and the number of equivalence queries; (f) the relation between k and the number of equivalence queries.

Table 2 The information of the experiments in which the alphabet size $|\Sigma| = k = 4$ and the number of states $|Q| = n = 8$ and the maximal partition size $m \geqslant |\Psi^{\lambda^c}_{q,\sigma}|$ ranges from 2 to 7

Case ID	m	$\lvert\Delta\rvert_{\text{mean}}$	Membership				Equivalence				t_{mean}
			$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	
7_4_2	2	45.7	435	629.0	629	861	18	22.8	22	29	8.9
7_4_3	3	51.1	495	666.4	654	861	26	31.0	30	38	14.9
7_4_4	4	58.1	575	787.8	771	1022	36	45.4	45	66	30.1
7_4_5	5	60.6	610	864.9	837	1162	34	49.7	49	67	28.2
7_4_6	6	78.6	715	1160.6	1167	1554	58	83.0	83	106	97.5
7_4_7	7	83.2	900	1322.7	1357	1694	70	93.4	95	124	142.4

the number of membership queries and the number of equivalence queries increase linearly as a function of the maximal partition size. We find that the statistical data are similar when the maximal partition size m is 4 and 5. This is likely owing to randomness.

In Table 3, we fix the number of states of all corresponding CRTA to $|Q| = n = 8$ and the maximal partition size to $m = 4$. The size of the alphabet $|\Sigma| = k$ ranges from 2 to 7. Figure 6(c) and (f) show that the number of membership queries and the number of equivalence queries increase linearly as a function of the alphabet size.

Table 3 The information of the experiments in which the number of states $|Q| = n = 8$ and the maximal partition size $m = 4 \geqslant |\Psi_{q,\sigma}^{\lambda^c}|$ and the alphabet size $|\Sigma| = k$ ranges from 2 to 7

Case ID	k	$\|\Delta\|_{\text{mean}}$	Membership				Equivalence				t_{mean}
			$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	$N_{\min}$	N_{mean}	N_{median}	$N_{\max}$	
7_2_4	2	33.7	296	568.7	570	798	23	31.0	33	37	7.8
7_3_4	3	45.1	420	649.0	648	980	25	36.9	36	56	14.2
7_4_4	4	58.1	575	787.8	771	1022	36	45.4	45	66	30.1
7_5_4	5	73.1	695	1034.6	1060	1428	43	56.3	53	79	83.7
7_6_4	6	86.0	870	1127.5	1104	1589	48	64.1	62	89	88.4
7_7_4	7	100.8	1020	1308.7	1299	1743	48	74.0	77	99	202.2

6 Conclusion

In this paper, we present an efficient Angluin-style learning algorithm for deterministic real-time automata. We convert the problem of learning a DRTA $\mathcal{A}$ to the problem of learning a canonical real-time automaton $\mathbb{A}$ with the same recognized language, i.e., $\mathcal{L}(\mathbb{A}) = \mathcal{L}(\mathcal{A})$. With the help of the partition functions and the refinement function, we can learn a correct model in both continuous-time semantics and discrete-time semantics. We also implement a prototypical tool and utilize it to learn a number of randomly generated RTAs. The experiments provide support for the correctness of our algorithm and the complexity analysis.

Acknowledgements Jie AN and Miaomiao ZHANG have been supported partly by National Natural Science Foundation of China (Grant Nos. 61972284, 61472279). Jie AN, Lingtai WANG, Bohua ZHAN and Naijun ZHAN have been supported partly by National Natural Science Foundation of China (Grant Nos. 61625206, 61732001, 61872341). Bohua ZHAN has been partly supported by CAS Pioneer Hundred Talents Program (Grant No. Y9RC585036). The authors would like to thank the anonymous reviewers for their insightful comments and suggestions raised in the reviewing process.

References

1 Angluin D. Learning regular sets from queries and counterexamples. Inf Computat, 1987, 75: 87–106
2 Aarts F, Vaandrager F W. Learning I/O automata. In: Proceedings of the 21st International Conference on Concurrency Theory, Paris, 2010. 71–85
3 Howar F, Steffen B, Jonsson B, et al. Inferring canonical register automata. In: Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation, Philadelphia, 2012. 251–266
4 Aarts F, Fiterau-Brostean P, Kuppens H, et al. Learning register automata with fresh value generation. In: Proceedings of the 12th International Colloquium on Theoretical Aspects of Computing, 2015. 165–183
5 Cassel S, Howar F, Jonsson B, et al. Active learning for extended finite state machines. Form Asp Comput, 2016, 28: 233–263
6 Bollig B, Habermehl P, Kern C, et al. Angluin-style learning of NFA. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, 2009. 1004–1009
7 Farzan A, Chen Y, Clarke E M, et al. Extending automated compositional verification to the full class of omega-regular languages. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, 2008. 2–17
8 Li Y, Chen Y, Zhang L J, et al. A novel learning algorithm for büchi automata based on family of dfas and classification trees. In: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Uppsala, 2017. 208–226
9 Bollig B, Katoen J, Kern C, et al. libalf: the automata learning framework. In: Proceedings of the 22nd International Conference on Computer Aided Verification, Edinburgh, 2010. 360–364
10 Isberner M, Howar F, Steffen B. The open-source learnlib—a framework for active automata learning. In: Proceedings of the 27th International Conference on Computer Aided Verification, San Francisco, 2015. 487–495
11 Fiterau-Brostean P, Janssen R, Vaandrager F W. Combining model learning and model checking to analyze TCP implementations. In: Proceedings of the 28th International Conference on Computer Aided Verification, Toronto, 2016. 454–471
12 Verwer S, de Weerdt M, Witteveen C. Efficiently identifying deterministic real-time automata from labeled data. Mach Learn, 2012, 86: 295–333
13 Dima C. Real-time automata. J Autom Languages Combinat, 2001, 6: 3–23
14 Verwer S, Weerdt M, Witteveen C. The efficiency of identifying timed automata and the power of clocks. Inf Comput, 2011, 209: 606–625
15 Alur R, Fix L, Henzinger T A. Event-clock automata: a determinizable class of timed automata. Theor Comput Sci, 1999, 211: 253–273
16 Grinchtein O, Jonsson B, Leucker M. Learning of event-recording automata. Theor Comput Sci, 2010, 411: 4029–4054
17 Maler O, Mens I. Learning regular languages over large alphabets. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Grenoble, 2014. 485–499
18 Drews S, D'Antoni L. Learning symbolic automata. In: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Uppsala, 2017. 173–189
19 Alur R, Dill D L. A theory of timed automata. Theor Comput Sci, 1994, 126: 183–235
20 Stigge M, Ekberg P, Guan N, et al. The digraph real-time task model. In: Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, Chicago, 2011. 71–80
21 Abdullah J, Dai G, Mohaqeqi M, et al. Schedulability analysis and software synthesis for graph-based task models with resource sharing. In: Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium, Porto, 2018. 261–270

22 Denning D E, Sacco G M. Timestamps in key distribution protocols. Commun ACM, 1981, 24: 533–536
23 Verwer S, Weerdt M D, Witteveen C. An algorithm for learning real-time automata. In: Proceedings of the 18th Annual Machine Learning Conference of Belgium and the Netherlands, Amsterdam, 2007. 57–64
24 Tappler M, Aichernig B K, Larsen K G, et al. Learning timed automata via genetic programming. 2018. ArXiv: 1808.07744
25 Schmidt J, Ghorbani A, Hapfelmeier A, et al. Learning probabilistic real-time automata from multi-attribute event logs. Intell Data Anal, 2013, 17: 93–123
26 Berg T, Grinchtein O, Jonsson B, et al. On the correspondence between conformance testing and regular inference. In: Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering, Edinburgh, 2005. 175–189
27 Hopcroft J E, Ullman J D. Introduction to Automata Theory, Languages, and Computation. Boston: Addison-Wesley Publishing Company, 1979
28 Ouaknine J, Worrell J. On the language inclusion problem for timed automata: closing a decidability gap. In: Proceedings of the 19th IEEE Symposium on Logic in Computer Science, Turku, 2004. 54–63
29 Wang L T, Zhan N J, An J. The opacity of real-time automata. IEEE Trans Comput-Aided Des Integr Circ Syst, 2018, 37: 2845–2856