Active Learning of One-Clock Timed Automata using Constraint Solving

Runqing Xu[1,2], Jie An[3], and Bohua Zhan[1,2]

[1] State Key Lab. of Computer Science, Institute of Software, CAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{xurq,bzhan}@ios.ac.cn
[3] Max Planck Institute for Software Systems, Kaiserslautern, Germany
jiean@mpi-sws.org

Abstract. Active automata learning in the framework of Angluin's L^* algorithm has been applied to learning many kinds of automata models. In applications to timed models such as timed automata, the main challenge is to determine guards on the clock value in transitions as well as which transitions reset the clock. In this paper, we introduce a new algorithm for active learning of deterministic one-clock timed automata and timed Mealy machines. The algorithm uses observation tables that do not commit to specific choices of reset, but instead rely on constraint solving to determine reset choices that satisfy readiness conditions. We evaluate our algorithm on randomly-generated examples as well as practical case studies, showing that it is applicable to larger models, and competitive with existing work for learning other forms of timed models.

Keywords: Active Learning · Timed Automata · Constraint Solving.

1 Introduction

Within Angluin's L^* framework [7], active learning is a type of model inference to learn an unknown language by making queries to a teacher. There are two kinds of queries: membership queries and equivalence queries. For a membership query, the teacher answers whether the queried word is in the target language. Usually, the learner collects query results in an *observation table*. When the observation table satisfies some readiness conditions, it can be transformed to a candidate automaton for an equivalence query. The teacher answers whether the candidate automaton recognizes the target language, and returns a counterexample if the answer is negative. In recent decades, the core algorithm has seen many technical improvements, has been extended to learn different kinds of models, and has been applied to many realistic settings. We refer to [18,20] for surveys.

For timed systems, timing constraints play a key role in the correctness of the system. In general, automata learning of timed systems require learning a timed model from either passive or active observations of the system, consisting of a collection of time-event sequences. The learned model should describe these timing behaviors correctly. Timed automata [2], extending DFAs with clock variables, is a popular formal model of timed systems. However, there are several obstacles to active learning of timed automata. Since the transitions of timed automata contain both timing constraints that test the values of clocks, and resets that update the clocks, we need to determine (1) the number of clocks, (2) the reset information, and (3) the timing constraints, none of which are directly observable from time-event sequences. Hence, existing work consider timed automata with different restrictions. Among them, An et al. introduced an active learning algorithm for deterministic one-clock timed automata (DOTAs) [4]. They first suppose that the teacher can return reset information in the queries, then the assumption is dropped by allowing the algorithm to search through possible combinations of reset information. However, this search process results in an exponential blow up, limiting the scalability of the algorithm in practical applications. Vaandrager et al. [21] considered a different class of timed models called Mealy machines with one timer, and proposed a learning algorithm with polynomial complexity.

In this paper, we present a new active learning algorithm for deterministic one-clock timed automata. The main innovation of the algorithm is to maintain all available observations in a single observation table, without committing to a particular choice of resets. The readiness conditions of the observation table, such as closedness, consistency, etc, are encoded as formulas in terms of variables for reset information and location assignments. These constraints are then solved using SMT solvers to determine feasible assignments of resets and locations that make the observation table ready, and from which a candidate automaton can be constructed. The learning algorithm is guaranteed to terminate and return a correct automata. While the theoretical worst-case complexity of the algorithm is still exponential, by leveraging the efficiency of SMT solvers, it is much more efficient than the algorithm in [4] in practice. In order to apply the algorithm to learning real-time reactive systems, we extend it to timed Mealy machines, which can be considered as an extension of Mealy machines with one clock, or extension of deterministic one-clock timed automata to include inputs and outputs.

The algorithm is implemented and evaluated on a number of randomly generated models and four models from practical applications. The experimental results show that our algorithm is scalable to much larger models compared to [4]. Additionally, our method successfully learns all four models from practical applications, with costs that is competitive against algorithms designed for other forms of models.

The organization of the paper is as follows. We give some background material in Section 2. The algorithm for learning DOTAs is described in Section 3, and its extension to timed Mealy machines in Section 4. We describe the implementation and experiments in Section 5, and finally conclude in Section 6.

Related work. Active learning of timed systems has been studied on many kinds of models with different restrictions. In [11,10], Grinchtein et al. proposed learning algorithms for event-recording automata (ERAs) [3], a kind of timed automata associating every action a with a clock x_a that records the length of time since the last occurrence of a. Henry et al. considered in [12] reset-free ERAs, where some transitions may reset no clocks. An et al. introduced a learning algorithm for deterministic one-clock timed automata (DOTAs) in [4], but due to the brute-force search over choice of resets, the algorithm is limited to timed automata with a small number of locations. For real-time automata (RTA) [8], efficient learning algorithms have been designed in both the deterministic [5] and nondeterministic case [6]. Recently, Vaandrager et al. introduced a new kind of timed models named Mealy machine with one timer (MM1T), and proposed an efficient active learning algorithm for such models [21]. It extends Mealy machine with a single timer which can be set to an integer value at transitions.

Passive learning has also been investigated for learning timed automata based on different methods [22,23,24,19,1]. Constraint solving has been used extensively in passive learning. For example, Smetsers et al. used this technique in passive learning of DFAs, Mealy machines and register automata [17], by encoding the existence of an automaton with n locations consistent with a set of observations in a logical formula. Compared to the use of constraint solving in passive learning, we remain in the active learning setting, encoding constraints for the readiness of observation tables rather than consistency with a set of observations. Moreover, we consider the case of timed automata, which requires encoding clock-reset information as well as location assignments. Our work demonstrates that constraint solving can be fruitfully applied in active learning, determining not only location assignments but also other hidden parts of the model such as clock-reset information.

2 Preliminaries

In this section, we introduce several concepts of one-clock timed automata. Let $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ be the set of non-negative reals and natural numbers, respectively. The set of boolean values is denoted as $\mathbb{B} = \{\top, \bot\}$, where $\top$ stands for *true* and $\bot$ for *false*.

Let c be the single clock variable, denote by Φ_c the set of clock constraints of the form $\phi ::= \top \mid c \bowtie m \mid \phi \wedge \phi$, where $m \in \mathbb{N}$ and $\bowtie \in \{=, <, >, \leq, \geq\}$. Since there is only one clock, a clock constraint can be represented as an integer-bounded interval whose endpoints are in $\mathbb{N} \cup \{\infty\}$. For example, $c \leq 5 \wedge c > 4$ is represented as $(4, 5]$, $c = 6$ as $[6, 6]$, and $\top$ as $[0, \infty)$. We will use inequality and interval representations interchangeably in this paper. Let the finite set of actions Σ be fixed.

Definition 1 (One-clock timed automata [4]). *A one-clock timed automaton (OTA) $\mathcal{A}$ is a 6-tuple $(\Sigma, Q, q_0, F, c, \Delta)$, where Σ is a finite set of actions, called the* alphabet*; Q is a finite set of locations; $q_0 \in Q$ is the initial location; $F \subseteq Q$ is a set of accepting locations; c is the unique clock; and $\Delta \subseteq Q \times \Sigma \times \Phi_c \times \mathbb{B} \times Q$ is a finite set of transitions.*

A transition $\delta = (q, \sigma, \phi, b, q')$ allows a jump from the *source location* q to the *target location* q' by performing the action $\sigma \in \Sigma$ if the *guard* $\phi \in \Phi_c$ is satisfied. Meanwhile, clock c is reset to zero if $b = \top$, and remains unchanged otherwise. *Clock valuation* is a function $\nu \colon c \to \mathbb{R}_{\geq 0}$ that assigns a non-negative real number to the clock. A *state* of $\mathcal{A}$ is a pair (q, ν), where $q \in Q$ and ν is a clock valuation.

Given an OTA $\mathcal{A}$, with κ being the maximum constant appearing in the guards, then the clock valuations can be divided into *regions*, where each region is of the form $[n, n]$ for $n \leq \kappa$, or $(n, n+1)$ for $n < \kappa$, or (κ, ∞). This gives a partition of $\mathbb{R}_{\geq 0}$. For clock valuation ν, we denote by $[\![\nu]\!]$ the region containing it. Regions are commonly used in algorithms for analyzing timed automata, making the state space essentially finite.

Given a *timed word* $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, where t_i represents the delay time between two actions σ_{i-1} and σ_i, there is a *run* of $\mathcal{A}$ such that

$\rho = (q_0, \nu_0) \xrightarrow{t_1,\sigma_1} (q_1, \nu_1) \xrightarrow{t_2,\sigma_2} \cdots \xrightarrow{t_n,\sigma_n} (q_n, \nu_n)$, where $\nu_0(c) = 0$, only if (1) $(q_{i-1}, \sigma_i, \phi_i, b_i, q_i) \in \Delta$, (2) $\nu_{i-1}(c)+t_i$ satisfies ϕ_i, (3) $\nu_i(c) = \nu_{i-1}(c)+t_i$ if $b_i = \bot$ and $\nu_i(c) = 0$ otherwise, for all $1 \leq i \leq n$. Let $|\omega|$ denote its *length*. When the timed automata $\mathcal{A}$ is known, each timed word ω can be extended by including the reset information b_i, indicating whether there is a reset after taking the transition for (σ_i, t_i). We denote the corresponding *reset timed word* as $\omega_r = (\sigma_1, t_1, b_1)(\sigma_2, t_2, b_2)\cdots(\sigma_n, t_n, b_n)$. If $q_n \in F$, then ω is an accepting timed word of $\mathcal{A}$. The *(recognized) timed language* $L(\mathcal{A})$ is the set of accepting timed words of $\mathcal{A}$.

An OTA is a *deterministic one-clock timed automaton* (DOTA) if there is at most one run for any timed word (equivalently, if the guards of transitions from any location do not intersect each other). Two DOTAs are equivalent if they recognize the same timed language. A DOTA is *complete* if for any $q \in Q$ and action $\sigma \in \Sigma$, the corresponding guards form a partition of $\mathbb{R}_{\geq 0}$. This means any given timed word has exactly one run. Any DOTA $\mathcal{A}$ can be transformed into a complete DOTA (COTA) accepting the same timed language by introducing a non-accepting "sink" location and letting all invalid or non-described behaviors go to the sink. We therefore assume that we are working with complete DOTAs. Fig. 1 shows a DOTA $\mathcal{A}$, with q_2 added as the sink location to make it complete.

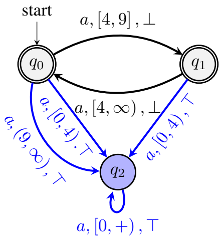


Fig. 1: An example of complete DOTA

Active learning of DOTAs is the problem of obtaining a DOTA for a system by interacting with a teacher who can answer two types of queries. In a membership query, the teacher is given a timed word, and returns whether the timed word is accepted. We use $\mathsf{MQ} : (\Sigma \times \mathbb{R}_{\geq 0})^* \to \{+, -\}$ to denote the function mapping timed words to $+$ if it is accepted, and $-$ otherwise. In an equivalence query, the teacher receives a hypothesis DOTA $\mathcal{H}$, and returns whether it is equivalent to the target automaton $\mathcal{A}$, returning a counterexample in the negative case.

An et al. [4] described an active learning algorithm for COTAs. First, a "smart" teacher is assumed, who also returns the reset information for any timed word given as membership query. With this reset information, an observation table can be maintained where each row in $S \cup R$ and each column in E is a *logical timed word*, which contains the clock valuation after each transition rather than the delay times. When the observation table satisfies readiness conditions, a candidate automaton is constructed in a manner similar to the learning of symbolic automata [9]. Next, in the "normal" teacher setting, where the teacher does not return reset information, the algorithm searches over all possible choices of resets for timed words in $S \cup R$, constructing one observation table for each. Due to the large time and memory requirements, the algorithm has limited scalability (up to only six locations for the randomly constructed examples in the paper).

3 Learning Algorithm

In this section, we describe a new algorithm for active learning of DOTAs by incorporating constraint solving using SMT solvers. The main idea of the algorithm is to maintain a single observation table that collects all results from previous membership queries, rather than one observation table for each possible choice of resets. Instead, the reset information is encoded as boolean variables with unknown values. At any time, the observation table may be ready for some choice of resets, but not for others. We encode the readiness conditions for the observation table as a formula on the variables for reset information, as well as for the location assignments of rows in the table. An SMT solver is then used to solve these constraints, returning a choice of resets that make the table ready, as well as a candidate automaton that can be sent for equivalence query.

Several difficulties must be addressed in order to realize this idea. In particular, whether two rows in the observation table are distinguishable is no longer clear-cut, but may depend on the choice of resets. This also means the partition of rows into S and R in the traditional L^* algorithm need to be extended, by adding a category S_+ of rows that are distinguishable from rows in S only for some choice of resets, that are needed for building candidate automaton.

In the remainder of this section, we first discuss comparison of timed words without being certain of clock-reset information, then extensions made to the observation table, method of encoding readiness constraints, and finally the main algorithm together with termination and complexity analysis.

3.1 Alignment and comparison of timed words

In the L^* framework, one key step is to determine if two words w_1 and w_2 belong to different equivalence classes of the target regular language, i.e., arrive at different locations of the underlying target DFA A. It is achieved by membership queries via testing words w_1 and w_2 after appending some suffix e. If $w_1 \cdot e$ is an accepting word and $w_2 \cdot e$ is not (or vice versa), then w_1 and w_2 must arrive at different locations.

In our case, suppose $\mathcal{A}$ is the target DOTA to be learned. Given two timed words ω_1 and ω_2, we wish to determine whether they arrive at different locations of $\mathcal{A}$. However, since the reset information is not observable when running timed words ω_1 and ω_2, the values of the clock at the end is unknown. If the values of the clock are not the same, the result of running $\omega_1 \cdot e$ and $\omega_2 \cdot e$ may be different even if ω_1 and ω_2 arrive at the same location. In order to effectively test using the suffix e, we need to suppose that the last time when the clock resets in ω_1 and ω_2 are known, and then align the values of the clock before executing the suffix e. We state these concepts more formally in the following definitions.

Definition 2 (Last reset of a timed word). *Given a timed word* $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$*, and DOTA* $\mathcal{A}$*. Let* $\omega_r = (\sigma_1, t_1, b_1)(\sigma_2, t_2, b_2) \cdots (\sigma_n, t_n, b_n)$ *be the reset timed word that results from running* ω *on* $\mathcal{A}$*. The* last reset $k_{\mathcal{A}}(\omega)$ *is defined to be 0 if* $b_i = \bot$ *for all* $1 \leq i \leq n$*, and* $k_{\mathcal{A}}(\omega) = i$ *if* $b_i = \top$ *and* $b_j = \bot$ *for all* $j > i$*.*

Suppose the last reset $k_{\mathcal{A}}(\omega)$ is known for a timed word ω, then we can compute the value of clock after executing ω on $\mathcal{A}$. Let $\nu_c(\omega, i)$ be the value of clock after executing ω if the last reset equals i. This is computed as the sum of t_{i+1} to t_n, where n is the length of ω (if $i = n$, for the case where the last reset occurs after (σ_n, t_n), then $\nu_c(\omega, i) = 0$). Hence, given two timed words ω_1 and ω_2 with known values of last resets i_1, i_2, it is possible to *align* the two timed words for testing a suffix e.

Definition 3 (Alignment for testing on a suffix). *Consider two timed words ω_1, ω_2, and suppose their last resets are i_1, i_2 respectively. Let $e = (\sigma_1, t_1)(\sigma_2, t_2)\cdots(\sigma_m, t_m)$ be a nonempty suffix. Let $\nu_1 = \nu_c(\omega_1, i_1)$ and $\nu_2 = \nu_c(\omega_2, i_2)$. Then form the suffixes e_1, e_2 depending on the following cases:*

- *If $\nu_1 > \nu_2$, then let $e_1 = e$ and $e_2 = (\sigma_1, t_1 + (\nu_1 - \nu_2)) \cdot (\sigma_2, t_2)\cdots(\sigma_m, t_m)$.*
- *If $\nu_1 < \nu_2$, then let $e_1 = (\sigma_1, t_1 + (\nu_2 - \nu_1)) \cdot (\sigma_2, t_2)\cdots(\sigma_m, t_m)$ and $e_2 = e$.*
- *If $\nu_1 = \nu_2$, then let $e_1 = e_2 = e$.*

Define a test $T(\omega_1, \omega_2, i_1, i_2, e)$ *between ω_1 and ω_2 with suffix e and last resets i_1, i_2 as follows. If e is nonempty, then the test compares results for two membership queries $\omega_1 \cdot e_1$ and $\omega_2 \cdot e_2$. The test succeeds, i.e. $T(\omega_1, \omega_2, i_1, i_2, e) = \top$, if $\mathsf{MQ}(\omega_1 \cdot e_1) = \mathsf{MQ}(\omega_2 \cdot e_2)$. Otherwise $T(\omega_1, \omega_2, i_1, i_2, e) = \bot$. If e is empty, the test simply compares the results for membership queries ω_1 and ω_2.*

It is clear that with the definition of e_1 and e_2, the value of clock when executing the first timed action of e_1 and e_2 during the tests must be the same. Hence, if ω_1 and ω_2 arrive at the same location, then the behavior on e_1 and e_2 must be the same as well. Hence, we obtain the following lemma.

Lemma 1 (Distinguishable timed words). *If the test T between ω_1 and ω_2 with suffix e and last resets i_1, i_2 fails, then for any DOTA $\mathcal{A}$ such that $i_1 = k_{\mathcal{A}}(\omega_1)$ and $i_2 = k_{\mathcal{A}}(\omega_2)$, the timed words ω_1 and ω_2 must arrive at different locations in $\mathcal{A}$.*

Example 1. Consider two timed words $\omega_1 = (a, 4)$ and $\omega_2 = \epsilon$. They are both accepted by $\mathcal{A}$ in Fig. 1. Consider suffix $e = (a, 5.5)$. If the last resets are $i_1 = 0$ and $i_2 = 0$ (the correct reset for $\mathcal{A}$), then $e_1 = (a, 5.5)$ and $e_2 = (a, 9.5)$, and $\mathsf{MQ}(\omega_1 \cdot e_1) = +$, $\mathsf{MQ}(\omega_2 \cdot e_2) = -$, so they can be distinguished, and indeed ω_1 and ω_2 arrive at different locations as required by Lemma 1. If the last resets are $i_1 = 1$ and $i_2 = 0$, then $e_1 = e_2 = (a, 5.5)$, and $\mathsf{MQ}(\omega_1 \cdot e_1) = \mathsf{MQ}(\omega_2 \cdot e_2) = +$, so they cannot be distinguished. Hence, when we do not know the true reset information, whether ω_1 and ω_2 are distinguishable by $e = (a, 5.5)$ depends on the choice of resets.

Since $\mathcal{A}$ is unknown during learning, we need to iterate over all possible combinations of last resets. For two timed words ω_1 and ω_2, we define the set of valid combinations of last resets as follows.

Definition 4 (Valid combinations of last resets). *Consider two timed words ω_1, ω_2 and suppose the length of the longest common prefix of ω_1 and ω_2 is m. Then the set $\mathcal{C}(\omega_1, \omega_2)$ of valid combinations of last resets is*

$$\mathcal{C}(\omega_1, \omega_2) = \{(i_1, i_2) \mid 0 \le i_1 \le |\omega_1| \wedge 0 \le i_2 \le |\omega_2| \wedge (i_1 \le m \wedge i_2 \le m \Rightarrow i_1 = i_2)\}$$

Example 2. Suppose $\omega_1 = (a, 4)$ and $\omega_2 = (a, 4)(a, 5.5)$, then $m = 1$, and the set of valid combinations are $\mathcal{C}(\omega_1, \omega_2) = \{(0, 0), (0, 2), (1, 1), (1, 2)\}$. In particular $(0, 1)$ or $(1, 0)$ are not allowed, since they give contradicting reset choices for the transition taken by $(a, 4)$.

3.2 Timed observation table

Definition 5 (Observation table). *An observation table $\mathcal{O} = (S, S_+, R, E, f, N)$ is a 6-tuple, satisfying the following conditions:*

- *S, S_+, R are disjoint finite sets of timed words called* prefixes. *$S \cup S_+ \cup R$ is prefix-closed and $\epsilon \in S$. If $\omega \in S \cup S_+$ and $\sigma \in \Sigma$, then $\omega \cdot (\sigma, 0) \in S \cup S_+ \cup R$.*
- *E is a finite set of timed words called* suffixes, *with $\epsilon \in E$.*
- *f is a function mapping pairs $\omega_1, \omega_2 \in S \cup S_+ \cup R$ and $(i_1, i_2) \in \mathcal{C}(\omega_1, \omega_2)$ to $\mathbb{B}$, indicating whether ω_1 and ω_2 are currently distinguished under last resets i_1, i_2.*
- *N is the current limit on the number of locations in the candidate automaton.*

The value $f(\omega_1, \omega_2, i_1, i_2)$ is computed as follows. For each suffix $e \in E$, test the pair ω_1, ω_2 under last resets i_1, i_2 and suffix e as in Definition 3. If $T(\omega_1, \omega_2, i_1, i_2, e) = \top$ for all $e \in E$, then $f(\omega_1, \omega_2, i_1, i_2) = \top$, otherwise $f(\omega_1, \omega_2, i_1, i_2) = \bot$.

Definition 6 (Certainly distinct rows). *Given an observation table $\mathcal{O}$, two rows ω_1, ω_2 are* certainly distinct *if $f(\omega_1, \omega_2, i_1, i_2) = \bot$ for all $i_1, i_2 \in \mathcal{C}(\omega_1, \omega_2)$.*

We first explain the different components of $\mathcal{O}$ in an intuitive way. The set S contains timed words that are certainly distinct from each other. The set S_+ are additional rows in the observation table that are distinct from rows in S under some choice of resets, that are required to be in the interior for some candidate automata. The set R is the boundary of the observation table as usual for L^* algorithms. The condition that $\omega \cdot (\sigma, 0) \in S \cup S_+ \cup R$ is analogous to the condition that $\omega \cdot \sigma \in S \cup R$ in the DFA case. It enforces that information is available to construct the transitions in the candidate automaton for each location and action $\sigma \in \Sigma$.

$\mathcal{O}$			$S \cup S_+ \cup R$						E
			ϵ	$(a,0)$	$(a,4)\,(a,5.5)$	$(a,0)\,(a,0)$	$(a,4)$	$(a,9.5)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\neg\mathbb{b}_5$	$\bot$	$\mathbb{b}_3$	$\bot$	ϵ
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	
R	$(a,4)\,(a,5.5)$	$\mathbb{b}_5$	$\neg\mathbb{b}_5$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$(a,5.5)$
	$(a,0)\,(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	
	$(a,4)$	$\mathbb{b}_3$	$\mathbb{b}_3$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	
	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	

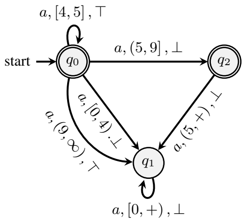


Fig. 2: **Left**: an instance of observation table $\mathcal{O}$ during learning of $\mathcal{A}$ in Fig. 1; **Right**: candidate DOTA constructed from the table after moving $(a,4)(a,5.5)$ to S_+.

Example 3. Fig. 2 shows an instance of the observation table. The rows of the table are timed words in S, S_+ and R (here S_+ is empty). The columns of the table are indexed by the same timed words. Each cell in the table summarizes when two timed words are distinguished (the function f), using formulas in terms of reset variables $\mathbb{b}$ introduced in Section 3.3. The table also shows the list of suffixes E.

The main difference between the observation table defined here and the usual ones for L^* algorithms is that we do not record a particular query result for each prefix in $S \cup S_+ \cup R$ and suffix E, as these results cannot be used effectively without knowing reset information. In contrast, we maintain which pairs of rows can be distinguished for each possible choice of reset information. This can be contrasted with the approach in [4]. Rather than maintaining a copy of the observation table for each combination of reset information, our method records all information obtained so far in a single table, with reset information determined by constraint solving.

3.3 Encoding of readiness constraints

To obtain a hypothesis DOTA $\mathcal{H}$ from observation table $\mathcal{O}$, we should provide an assignment for the location and reset information of each timed word in $S \cup S_+ \cup R$, which ensure readiness conditions for the table, such as closedness and consistency. The main idea is to encode such readiness conditions as formulas in terms of location and reset assignments, and then use SMT solvers to find feasible solutions to these constraints or prove that they are not satisfiable. The constraints are stated in terms of two variables for each row $\omega \in S \cup S_+ \cup R$: an *ending reset variable* $\mathbb{b}_\omega$ and a *location variable* $\mathbb{q}_\omega$.

Definition 7 (Ending reset variable and location variable). *Given a timed word $\omega = (\sigma_1, t_1) \ldots (\sigma_n, t_n) \in S \cup S_+ \cup R$, define the* ending reset variable $\mathbb{b}_\omega \in \{\top, \bot\}$ *to denote whether clock resets after running the final timed action (σ_n, t_n) (for the empty timed word ϵ, we declare $\mathbb{b}_\epsilon = \top$). Define the* location variable $\mathbb{q}_\omega \in \{1, \ldots, N\}$ *to represent the location of ω in the candidate automaton.*

Since the set of timed words $S \cup S_+ \cup R$ is prefix-closed, the ending reset variables $\{\mathbb{b}_\omega\}_{\omega \in S \cup S_+ \cup R}$ in fact determine whether the clock resets after each timed action for each row in $S \cup S_+ \cup R$. In particular, we can encode the last reset for ω in terms of the ending reset variables.

Definition 8 (Encoding of last reset). *Given $\omega = (\sigma_1, t_1)...(\sigma_n, t_n) \in S \cup S_+ \cup R$. Let $\omega|_i$ for $0 \leq i \leq n$ be the prefix of ω with length i. Since $S \cup S_+ \cup R$ is prefix-closed, we have each $\omega|_i \in S \cup S_+ \cup R$ as well. Let $lr(\omega, i)$, encoding the condition that the last reset of ω equals i, be defined as follows.*

$$lr(\omega, i) \triangleq \mathbb{b}_{\omega|_i} \wedge \bigwedge_{i<j\leq n} \neg \mathbb{b}_{\omega|_j}.$$

For each pair of rows $\omega_1, \omega_2 \in S \cup S_+ \cup R$ and each pair of last resets $(i, j) \in \mathcal{C}(\omega_1, \omega_2)$, the condition that the last resets of ω_1, ω_2 equal i, j respectively is encoded as follows.

$$LR(\omega_1, \omega_2, i, j) \triangleq lr(\omega_1, i) \wedge lr(\omega_2, j)$$

Based on the encoding of last reset, the readiness constraints for the observation table can be encoded in terms of the above variables as follows.

Constraint 1 (Distinctness of rows) Given timed words $\omega_1, \omega_2 \in S \cup S_+ \cup R$, and last resets $i, j \in \mathcal{C}(\omega_1, \omega_2)$, suppose $f(\omega_1, \omega_2, i, j) = \bot$ (meaning ω_1 and ω_2 can be distinguished under last resets i, j in the observation table), then we have the following constraint, indicating ω_1 and ω_2 cannot be assigned to the same location.

$$C_1(\omega_1, \omega_2, i, j) \triangleq LR(\omega_1, \omega_2, i, j) \Rightarrow \mathbb{q}_{\omega_1} \neq \mathbb{q}_{\omega_2}$$

Define the constraint C_1 to be the conjunction of all $C_1(\omega_1, \omega_2, i, j)$, for all pairs of rows and valid last resets that can be distinguished.

$$C_1 \triangleq \bigwedge_{\substack{\omega_1,\omega_2 \in S \cup S_+ \cup R,\\ (i,j)\in\mathcal{C}(\omega_1,\omega_2),\\ f(\omega_1,\omega_2,i,j)=\bot}} C_1(\omega_1, \omega_2, i, j).$$

Constraint 2 (Consistency) Given timed word $\omega_1, \omega_2 \in S \cup S_+ \cup R$ and last resets $i, j \in \mathcal{C}(\omega_1, \omega_2)$. Suppose $\omega_1' = \omega_1 \cdot (\sigma, t_1)$ and $\omega_2' = \omega_2 \cdot (\sigma, t_2)$ also appears in $S \cup S_+ \cup R$, for some $\sigma \in \Sigma$ and $t_1, t_2 \in \mathbb{R}_{\geq 0}$. Suppose that under the last resets i, j, the value of clock after executing last timed action of ω_1', but before possible resets, is in the same region as that for ω_2', then if ω_1 and ω_2 also go to the same location, the transition to be carried out for the last timed action of ω_1' and ω_2' must be the same. Hence both ending reset and location for ω_1' and ω_2' must be the same. This is encoded as constraints as follows.

$$C_2(\omega_1, \omega_2, i, j, \sigma, t_1, t_2) \triangleq \mathbb{q}_{\omega_1} = \mathbb{q}_{\omega_2} \wedge LR(\omega_1, \omega_2, i, j) \Rightarrow \mathbb{b}_{\omega_1'} = \mathbb{b}_{\omega_2'} \wedge \mathbb{q}_{\omega_1'} = \mathbb{q}_{\omega_2'}.$$

It is added as a constraint only if $[\![\nu_c(\omega_1, i)+t_1]\!] = [\![\nu_c(\omega_2, j)+t_2]\!]$, and if $f(\omega_1, \omega_2, i, j) = \top$. We define constraint C_2 to be the conjunction of all such constraints (in the formula below, $\omega_1' = \omega_1 \cdot (\sigma, t_1)$ and $\omega_2' = \omega_2 \cdot (\sigma, t_2)$).

$$C_2 \triangleq \bigwedge_{\substack{\omega_1,\omega_2,\omega_1',\omega_2' \in S \cup S_+ \cup R,\\ (i,j)\in\mathcal{C}(\omega_1,\omega_2),\\ f(\omega_1,\omega_2,i,j)=\top,\\ [\![\nu_c(\omega_1,i)+t_1]\!]=[\![\nu_c(\omega_2,j)+t_2]\!]}} C_2(\omega_1, \omega_2, i, j, \sigma, t_1, t_2).$$

Constraint 3 (Closedness) The closedness condition for usual L^* algorithms states that each row in R must be represented by a row in S. In our case, we require that each row in R is represented by a row in $S \cup S_+$. This translates to the constraint that each location in the candidate automaton must be represented by a row in $S \cup S_+$, encoded as follows (recall N is the current limit on the number of locations).

$$C_3 \triangleq \bigwedge_{1 \leq i \leq N} \bigvee_{\omega \in S \cup S_+} \mathbb{q}_\omega = i \ \wedge \ C_3' \text{ where } C_3' \triangleq \bigwedge_{\omega \in S \cup S_+ \cup R} 1 \leq \mathbb{q}_\omega \leq N.$$

During the algorithm, we also make constraint solving queries where the closedness condition is not enforced. Then only the second part C_3' is used.

Constraint 4 (Special assignments) In order to speed-up constraint solving, we directly make assignments to the location variables of rows in S. Order the rows of S as $S = \{\omega_1, \omega_2, \ldots, \omega_{|S|}\}$, then the special assignments are encoded as follows.

$$C_4 \triangleq \bigwedge_{1 \leq i \leq |S|} \mathbb{q}_{\omega_i} = i.$$

In the main learning algorithm, we will use SMT solvers to attempt to find solutions to these constraints. The algorithm will first attempt to find a solution using C_3 together with C_1, C_2 and C_4. If a solution is found, it proceeds to hypothesis construction as described in Section 3.4. Otherwise, it attempts to find a solution using C_3' or by increasing N. The details are described in Section 3.5.

3.4 Hypothesis construction

Once the SMT solver gives a model satisfying constraints in Section 3.3, we can build a hypothesis DOTA $\mathcal{H} = (\Sigma, Q_\mathcal{H}, q_0^\mathcal{H}, F_\mathcal{H}, c, \Delta_\mathcal{H})$ from observation table $\mathcal{O} = (S, S_+, R, E, f, N)$ and assignments $\overline{\mathbb{b}_\omega}$ and $\overline{\mathbb{q}_\omega}$ to ending reset variable and location variables in the model. We define location set $Q_\mathcal{H} = \{\overline{\mathbb{q}_\omega} \mid \omega \in S \cup S_+\}$, initial location $q_0^\mathcal{H} = \overline{\mathbb{q}_\epsilon}$, and accepting locations $F_\mathcal{H} = \{\overline{\mathbb{q}_\omega} \mid \mathsf{MQ}(\omega) = + \wedge \omega \in S \cup S_+\}$. Next, we describe how to construct the transitions $\Delta_\mathcal{H}$.

Given two rows $\omega_1, \omega_2 \in S \cup S_+ \cup R$ such that $\omega_2 = \omega_1 \cdot (\sigma, t)$, we can construct an auxiliary transition $\delta' = (\overline{\mathbb{q}_{\omega_1}}, \sigma, \psi, \overline{\mathbb{b}_{\omega_2}}, \overline{\mathbb{q}_{\omega_2}})$ with $\psi = \nu(\omega_1) + t$ where $\nu(\omega_1)$ is the value of clock after executing ω_1. Since the table is prefix-closed, the reset information for every timed action in ω_1 has been determined. Therefore, we can determine $\nu(\omega_1)$. We collect all such auxiliary transitions as the set Δ'.

For any $q \in Q_\mathcal{H}$ and $\sigma \in \Sigma$, let $\Psi_{q,\sigma} = \{\psi \mid (q, \sigma, \psi, b, q') \in \Delta'\}$ be the list of clock values on auxiliary transitions from q and with action σ. We sort $\Psi_{q,\sigma}$ and apply the partition function $P(\cdot)$ to obtain m intervals, written as $g_1, \cdots, g_m$, satisfying $\psi_i \in g_i$ for any $1 \leq i \leq m$, where $m = |\Psi_{q,\sigma}|$; consequently, for every $(q, \sigma, \psi_i, b, q') \in \Delta'$, a transition $\delta_i = (q, \sigma, g_i, b, q')$ is added to $\Delta_\mathcal{H}$. This determines the transitions between locations in $\mathcal{H}$ and hence finishes the construction. The partition function $P(\cdot)$ is taken from [4], and also similar to that used for learning symbolic automata [9]. Note the condition $\omega \cdot (\sigma, 0) \in S \cup S_+ \cup R$ in Definition 5 enforces $\mu_0 = 0$ below.

Definition 9 (Partition function). *Given a list of clock valuations $\ell = \mu_0, \mu_1, \cdots, \mu_n$ with $0 = \mu_0 < \mu_1 \cdots < \mu_n$, and $\lfloor \mu_i \rfloor \neq \lfloor \mu_j \rfloor$ if $\mu_i, \mu_j \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}$ and $i \neq j$ for all $1 \leq i, j \leq n$, let $\mu_{n+1} = \infty$, then a partition function $P(\cdot)$ mapping ℓ to a set of intervals $\{g_0, g_1, \ldots, g_n\}$, which is a partition of $\mathbb{R}_{\geq 0}$, is defined as*

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

Since the table $\mathcal{O}$ with the feasible assignments satisfy the readiness constraints, the constructed hypothesis is a deterministic one-clock timed automaton, and agrees with accepting information for rows in $S \cup S_+ \cup R$. This is stated as the following theorem.

Theorem 1. *Given observation table $\mathcal{O} = (S, S_+, R, E, f, N)$ and feasible assignments to $\overline{\mathbb{b}_\omega}$ and $\overline{\mathbb{q}_\omega}$, the hypothesis $\mathcal{H} = (\Sigma, Q, q_0, F, c, \Delta)$ is deterministic. For each row $\omega \in S \cup S_+ \cup R$, $\mathcal{H}$ accepts the timed word ω iff $\mathsf{MQ}(\omega) = +$. Finally, for any two rows $\omega_1, \omega_2 \in S \cup S_+ \cup R$, if the value of f on ω_1, ω_2 and the setting of reset variables $\overline{\mathbb{b}}$ is $\perp$, then $\overline{\mathbb{q}_{\omega_1}} \neq \overline{\mathbb{q}_{\omega_2}}$, and the two rows reach distinct locations in $\mathcal{H}$.*

After the hypothesis $\mathcal{H}$ is built, it is sent for an equivalence query. If the teacher returns a counterexample ctx, the learner adds all prefixes of ctx to R during counterexample processing.

Algorithm 1: Learning DOTA using constraint solving

```
input : an observation table $\mathcal{O} = (S, S_+, R, E, f, N)$, the alphabet $\Sigma$.
output: an automata $\mathcal{H}$ recognizing the target language $L$.
1  $S \leftarrow \{\epsilon\}$, $S_+ \leftarrow \emptyset$, $R \leftarrow \{(\sigma, 0) \mid \sigma \in \Sigma\}$, $E \leftarrow \{\epsilon\}$, $N \leftarrow 1$ ;    // initialization
2  while $\top$ do
3      $\mathcal{O} \leftarrow$ move_to_S($\mathcal{O}$);
4      $flag, M \leftarrow$ SMT($C_1 \wedge C_2 \wedge C_3 \wedge C_4$) ;    // solve constraints to get model $M$
5      if $flag = \top$ then
6          $\mathcal{H} \leftarrow$ build_hypothesis($\mathcal{O}, M$) ;    // build $\mathcal{H}$ from table $\mathcal{O}$ and model $M$
7          $equivalent, ctx \leftarrow$ equivalence_query($\mathcal{H}$);
8          if $equivalent = \top$ then
9              return $\mathcal{H}$ ;    // success
10         else
11             $\mathcal{O} \leftarrow$ ctx_processing($\mathcal{O}, ctx$) ;    // counterexample processing
12     else
13         $flag, M' \leftarrow$ SMT($C_1 \wedge C_2 \wedge C_3' \wedge C_4$) ;    // solve relaxed constraints
14         if $flag = \top$ then
15             $\mathcal{O} \leftarrow$ move_to_S$_+$($\mathcal{O}, M'$) ;    // modify table $\mathcal{O}$ guided by solution $M'$
16         else
17             $N \leftarrow N + 1$ ;    // try for larger number of locations
```

3.5 Main algorithm and correctness

The overall procedure of the algorithm is given in Algorithm 1. The observation table $\mathcal{O} = (S, S_+, R, E, f, N)$ is initialized with $S = \{\epsilon\}$, $S_+ = \emptyset$, $R = \{(\sigma, 0) \mid \sigma \in \Sigma\}$, $E = \{\epsilon\}$, and $N = 1$. The function move_to_S tests each row in R to see if it is certainly distinct (according to Definition 6) from each row in S. If so the certainly distinct row is moved to S. For each row ω moved to S, $\omega \cdot (\sigma, 0)$ is added to R for every $\sigma \in \Sigma$ (Line 3). After that, the formula $C_1 \wedge C_2 \wedge C_3 \wedge C_4$ is built and sent to an SMT solver (Line 4). If a solution M is found for the ending reset and location variables, then a hypothesis $\mathcal{H}$ is constructed from the table $\mathcal{O}$ and the solution M (Line 6), and an equivalence query is performed to determine whether the hypothesis $\mathcal{H}$ is correct. If the answer is positive, the algorithm returns with automaton $\mathcal{H}$ (Line 9). Otherwise, Learner updates the table $\mathcal{O}$ by adding all prefixes of the returned counterexample ctx to R (Line 11), and begins a new iteration starting from finding certainly distinct rows (Line 3) and updating constraints. Note that new suffixes may be added to E during the computation of constraint C_2. If two timed words $\omega_1 \cdot (\sigma, t_1)$ and $\omega_2 \cdot (\sigma, t_2)$ end in the same region under some choice of resets, ω_1 and ω_2 are currently indistinguishable under this choice, but $\omega_1 \cdot (\sigma, t_1)$ and $\omega_2 \cdot (\sigma, t_2)$ can be distinguished with suffix $e \in E$, then the timed word $(\sigma, \min(t_1, t_2)) \cdot e$ is added to E. This allows us to distinguish ω_1 and ω_2 directly using $(\sigma, \min(t_1, t_2)) \cdot e \in E$. After new suffixes are added to E, the entire observation table need to be updated, with possible new distinguishable pairs and new rows added to S.

If there is no solution to $C_1 \wedge C_2 \wedge C_3 \wedge C_4$, then Learner first relaxes C_3 to C_3' (Line 13). It is now permitted that some rows in R are assigned to a location different from any row in $S \cup S_+$. If there is a solution for the relaxed condition, it indicates that some row in R may represent a new location, even though it is not certainly distinct from all rows in S. The function move_to_S$_+$ moves such rows from R to S_+, and add

$\omega \cdot (\sigma, 0)$ to R for each $\sigma \in \Sigma$ and each ω moved to S_+ (Line 15). If there is no solution even for the relaxed constraints, the learner increases N by 1 (Line 17), attempting to find a model with larger size.

Example 4. In the observation table in Fig. 2, each cell at row ω_1 and column ω_2 records at which choice of resets ω_1 and ω_2 *cannot* be distinguished, as an expression in terms of $\mathbb{b}_\omega$'s. For example, the expression $\top$ means ω_1 and ω_2 cannot be distinguished for all choice of resets, while $\bot$ means ω_1 and ω_2 are certainly distinct.

Constraint 3 requires that each row in R is represented by some row in $S \cup S_+$. Although $\omega_3 = (a, 4)$ or $\omega_5 = (a, 4)(a, 5.5)$ can be represented by ϵ by setting $\mathbb{b}_3 = \top$ or $\mathbb{b}_5 = \bot$, they are known to be certainly distinct from each other (indicated by the red $\bot$ in the table), so they cannot be both represented by ϵ. This means $C_1 \wedge C_2 \wedge C_3 \wedge C_4$ is not satisfiable, so we relax the constraint to $C_1 \wedge C_2 \wedge C_3' \wedge C_4$. This is solvable by setting $\mathbb{b}_3 = \top$, and let ω_5 represent an additional location. Then ω_5 is moved to S_+, and the candidate DOTA at the right of Fig. 2 can be constructed from the updated observation table.

Analysis of the algorithm The algorithm is sound since it returns an automaton only if it passes equivalence queries. The termination of the algorithm can be explained through a comparison with a brute-force search version of the algorithm similar to the normal teacher case in [4]. The brute-force search constructs a binary tree of observation tables, with each branching corresponds to a choice of reset for some row in $S \cup R$ (here S_+ is not needed since reset information is now certain). Our algorithm simulates a breadth-first search on the tree based on the number of locations. Rows in S_+ can be viewed as rows that are added to S in some (but not all) branches of the search tree. A simple estimate for the number of rows in $S \cup S_+$, and hence in R gives an exponential worst-case bound in terms of N. However, in practice it usually increases slowly as shown in our experiments.

Theorem 2 (Correctness and termination). *Algorithm 1 always terminates and returns a correct DOTA recognizing the underlying target timed language.*

4 Extension to Deterministic Timed Mealy Machines

For practical applications on real-time reactive systems with input/output behavior, we consider a timed version of Mealy machines. Inspired by Mealy machine with one timer (MM1T) [21], we divide the actions in Σ into *input* and *output* actions. The special *empty action* ϵ represents the invisible action or nothing happening. We assume that there is a pair of input and output actions on each transition. Hence, the model can also be viewed as a Mealy machine with one clock.

Definition 10 (Timed Mealy Machines). *A timed Mealy machine (TMM) is a 6-tuple $\mathcal{M} = (Q, I, O, q_0, c, \Delta)$, where Q is a finite set of locations; I is a finite set of inputs, containing the special empty action ϵ; O is a finite set of outputs, containing the special empty action ϵ; q_0 is the unique initial location; c is the single clock; and $\Delta \subseteq Q \times I \times O \times \Phi_c \times \mathbb{B} \times Q$ is a finite set of transitions.*

A transition $\delta = (q, i, o, \phi, b, q')$ allows a jump from q to q' and generates an output o when provided input $i \in I$ and if $\phi \in \Phi_c$ is satisfied. Meanwhile, clock c is reset to zero if $b = \top$, and remains unchanged otherwise. Given a timed word over inputs $\omega = (i_1, t_1)(i_2, t_2) \cdots (i_n, t_n) \in (I \times \mathbb{R}_{\geq 0})^*$, a *deterministic timed Mealy machine* (DTMM) $\mathcal{M}$ returns at most one output sequence $\mathcal{M}(\omega) = o_1 o_2 \cdots o_n$. Given two DTMMs $\mathcal{M}_1$ and $\mathcal{M}_2$, for any timed word ω over inputs I, if the output sequences of two DTMMs are equal, i.e., $\mathcal{M}_1(\omega) = \mathcal{M}_2(\omega)$, then the two DTMMs are equivalent, denoted as $\mathcal{M}_1 \approx \mathcal{M}_2$. We modify the learning algorithm to take into account of inputs and outputs. By the same argument as for DOTAs, we can show correctness and termination of the learning algorithm for DTMMs.

Theorem 3 (Correctness and termination for learning DTMMs). *The learning algorithm for DTMMs always terminates and returns a correct DTMM.*

5 Implementation and Experiments

To investigate the efficiency and scalability of our methods, we implemented a prototype in Python named SL for both DOTAs and DTMMs based on the tool provided in [4]. We use Z3 [14] as the constraint solving engine. We describe some detailed aspects of the implementation below. The implementation and models used for experiments are available at https://github.com/Leslieaj/DOTALearningSMT.

Incremental solving Our implementation takes advantage of incremental SMT solving functionality in Z3. This allows Z3 to reuse information from previous calls to accelerate the solving process. For each query, we push a backtracking point after adding all new constraints in $C_1 \wedge C_2 \wedge C_4$, then insert the constraints C_3 or C_3' depending on the stage of the algorithm. After the query finished, we pop to the previous backtracking point, hence removing C_3 or C_3' before the next query.

Sink location We use a sink location to denote timed behaviors that are invalid or nondescribed, and it is sometimes possible for membership queries to return whether a timed word reached a sink location. Our implementation takes advantage of this information when it is available. The same technique has been used in previous works such as [4].

Equivalence query Equivalence between timed automata with one clock is decidable. We implemented an equivalence query oracle based on [15], but simplified for the deterministic case. In actual applications when the target automaton is unknown, this can be usually replaced by techniques based on conformance testing.

We evaluated our prototype tool on two benchmarks that are previously used in [4] and [21]. They respectively contain hundreds of randomly generated DOTAs and several models from practical applications which are in the form of DOTAs and MM1Ts. The models from practical applications consist of the abstract automata of an Authentication and Key Management service of the WiFi (AKM), the functional specification of TCP protocol, a car alarm system (CAS), and a particle counter (PC). All experiments have been carried out on an Intel Core i7-9750H @ 2.6GHz processor with 16GB RAM running Ubuntu 20.04 Linux system.

Table 1: Experimental results on DOTAs.

Group	$\|\Delta\|$	Method	#Membership			#Equivalence			$\|Q_{\mathcal{H}}\|$	#Learnt	$t(s)$
			N_{min}	N_{mean}	N_{max}	N_{min}	N_{mean}	N_{max}			
6_2_10	11.9	DOTAL	73	348.3	708	10	16.7	30	5.6	7/10	39.88
		SL	104	1894.8	3929	11	20.8	35	5.6	10/10	0.78
4_4_20	16.3	DOTAL	231	317.0	564	27	30.8	40	4.0	6/10	100.223
		SL	1740	3497.7	5329	24	32.8	42	4.0	10/10	1.42
7_4_20	26.0	DOTAL				–				0/10	TO
		SL	6092	9393.3	15216	44	51.5	69	7.0	10/10	2.90
10_4_20	39.1	DOTAL				–				0/10	TO
		SL	8579	16322.3	23726	59	76.5	93	10.0	10/10	5.89
12_4_20	47.6	DOTAL				–				0/10	TO
		SL	13780	20345.5	29011	70	88.0	102	12.0	10/10	10.052
14_4_20	58.4	DOTAL				–				0/10	TO
		SL	18915	28569.0	40693	92	110.6	126	14.0	10/10	14.692
AKM (17_12_5)	40	DOTAL				–				0/1	TO
		SL	3453	3453.0	3453	49	49.0	49	12	1/1	7.19
TCP (22_13_2)	22	DOTAL				–				0/1	TO
		SL	4713	4713.0	4713	32	32.0	32	20	1/1	19.04
CAS (14_10_27)	23	DOTAL				–				0/1	TO
		SL	4769	4769.0	4769	18	18.0	18	14	1/1	126.30
PC (26_17_10)	42	DOTAL				–				0/1	TO
		SL	10854	10854.0	10854	28	28.0	28	25	1/1	109.01

Group: 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 a DOTA in the corresponding group. **Method**: DOTAL and SL represent the method in [4] and our method 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. **#Learnt**: the number of the learnt DOTAs (learnt/total). t: average wall-clock time in seconds. TO: Timeout of 5 mins

5.1 Experiments on DOTAs

We first compared the performance of our learning algorithm SL with the algorithm DOTAL of [4] in the normal teacher setting. In [4], the generated random DOTAs are up to 14 locations, but the algorithm only managed to learn automata with up to 6 locations in the normal teacher setting. The examples are divided into different groups depending on the number of locations, number of actions, and maximum clock value in guards. Each group contains ten automata. Moreover, we tested translations of practical models to DOTA provided in [21]. The experimental results are shown in Table 1.

From the results, we can see that the algorithm DOTAL fails in all of the larger examples due to time and memory limits. In the two groups of smaller examples 6_2_10 and 4_4_20, the algorithm DOTAL can learn some of the cases. In the comparison between number of membership and equivalence queries, we see that SL takes about the same number of equivalence queries, and several times more membership queries. This is likely due to the fact that we exhaustively test all pairs of rows in the table under all reset conditions. However, the algorithm SL is scalable to much larger examples than DOTAL. SL also successfully learns the DOTA models of four practical applications

Table 2: Experimental results on DTMMs and MM1Ts.

Case	**DTMM**			**SL**			**MM1T**			**MM1T-L_M^*** [21]		
	$\lvert Q\rvert$	$\lvert I\rvert$	$\lvert \Delta\rvert$	**#M**	**#E**	$t(s)$	$\lvert Q\rvert$	$\lvert I\rvert$	$\lvert \Delta\rvert$	**#R**	**#I**	$t(s)$
AKM	5	5	28	691	34	2.6	4	5	24	5361	29693	5070.4
TCP	11	8	19	751	10	1.9	11	8	19	401	1868	65.7
CAS	8	4	17	1654	21	17.1	8	4	17	494	2528	79.5
AKM	8	8	24	1194	27	6.8	8	8	24	392	1864	85.1

which are all bigger than the randomly generated DOTAs, and far above the ability of the DOTAL algorithm. This shows the potential of the new algorithm in real applications.

5.2 Experiments on TMMs

We also evaluated our learning algorithm for timed Mealy machines. We first transformed the four MM1T models to DTMMs. As shown in Table 2, for each practical application, its DTMM model is more succinct than the corresponding DOTA model in Table 1. The size of the DTMM model is also comparable to the size of the MM1T model (the two are equal except the AKM case).

We then run our learning algorithm SL on these models. Compared to learning the corresponding DOTA, learning DTMM takes fewer membership and equivalence queries, except for taking more equivalence queries in the case CAS. Hence, we find DTMMs to be more suitable for learning timed reactive systems than DOTAs. We also run the experiment on MM1T using the algorithm **MM1T-**L_M^*. As reported in [21], the performance is evaluated according to the total number of resets to the system under learning (SUL) **#R** and the total number of the performed input actions **#I**. As these are not directly comparable to number of membership and equivalence queries, we list the results side-by-side in the table. Note also that their implementation is based on LearnLib [13] and uses a *random word* equivalence oracle with 1000 tests, while we conduct an exact equivalence checking. We find the cost of learning to be comparable between SL and methods for learning MM1Ts. We also note that [21] showed experimentally that the algorithm **MM1T-**L_M^* compares favorably against heuristic learning methods based on genetic programming [19,1].

6 Conclusion

In this paper, we proposed a new algorithm for active learning of deterministic one-clock timed automata and timed Mealy machines, using constraint solving based on SMT solving to determine resets and location assignments. This takes advantage of the ability of SMT to solve large constraint systems efficiently, allowing the algorithm to scale up to much larger timed automata models.

In future work, we wish to consider extension of the algorithm to learning timed automata with multiple clocks as well as the non-determinstic case. We wish to also consider incorporating ideas from algorithms such as TTT in order to improve efficiency, in particular reducing the number of membership queries.

References

1. Aichernig, B.K., Pferscher, A., Tappler, M.: From passive to active: Learning timed automata efficiently. In: NFM 2020. LNCS, vol. 12229, pp. 1–19. Springer (2020)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. Theor. Comput. Sci. **211**(1-2), 253–273 (1999)
4. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: TACAS 2020. LNCS, vol. 12078, pp. 444–462. Springer (2020)
5. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. Sci. China Inf. Sci. **64**(9) (2021)
6. An, J., Zhan, B., Zhan, N., Zhang, M.: Learning nondeterministic real-time automata. ACM Trans. Embed. Comput. Syst. **20**(5s), 99:1–99:26 (2021)
7. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
8. Dima, C.: Real-time automata. Journal of Automata, Languages and Combinatorics **6**(1), 3–23 (2001)
9. Drews, S., D'Antoni, L.: Learning symbolic automata. In: TACAS 2017. LNCS, vol. 10205, pp. 173–189. Springer (2017)
10. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theoretical Computer Science **411**(47), 4029–4054 (2010)
11. Grinchtein, O., Jonsson, B., Pettersson, P.: Inference of event-recording automata using timed decision trees. In: CONCUR 2006. LNCS, vol. 4137, pp. 435–449. Springer (2006)
12. Henry, L., Jéron, T., Markey, N.: Active learning of timed automata with unobservable resets. In: FORMATS 2020. LNCS, vol. 12288, pp. 144–160 (2020)
13. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib - A framework for active automata learning. In: CAV 2015. LNCS, vol. 9206, pp. 487–495 (2015)
14. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
15. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: LICS 2004. pp. 54–63 (2004)
16. Shahbaz, M., Groz, R.: Inferring Mealy machines. In: FM 2009. LNCS, vol. 5850, pp. 207–222. Springer (2009)
17. Smetsers, R., Fiterau-Brostean, P., Vaandrager, F.W.: Model learning as a satisfiability modulo theories problem. In: LATA 2018. LNCS, vol. 10792, pp. 182–194. Springer (2018)
18. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: SFM 2011. LNCS, vol. 6659, pp. 256–296 (2011)
19. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: FORMATS 2019. LNCS, vol. 11750, pp. 216–235. Springer (2019)
20. Vaandrager, F.W.: Model learning. Commun. ACM **60**(2), 86–95 (2017)
21. Vaandrager, F.W., Bloem, R., Ebrahimi, M.: Learning Mealy machines with one timer. In: LATA 2021. LNCS, vol. 12638, pp. 157–170. Springer (2021)
22. Verwer, S., de Weerdt, M., Witteveen, C.: One-clock deterministic timed automata are efficiently identifiable in the limit. In: LATA 2009. LNCS, vol. 5457, pp. 740–751. Springer (2009)
23. Verwer, S., de Weerdt, M., Witteveen, C.: The efficiency of identifying timed automata and the power of clocks. Inf. Comput. **209**(3), 606–625 (2011)
24. Verwer, S., de Weerdt, M., Witteveen, C.: Efficiently identifying deterministic real-time automata from labeled data. Machine Learning **86**(3), 295–333 (2012)

Appendix A Proofs of Theorem 1 and Theorem 2

Proof (of Theorem 1). Since $\overline{\mathbb{b}_\omega}$ and $\overline{\mathbb{q}_\omega}$ are feasible solutions to the constraints, all constraints in $C_1 \wedge C_2 \wedge C_3 \wedge C_4$ are satisfied. The construction of the hypothesis guarantees that for each transition of the timed word $\omega \in S \cup S_+ \cup R$, whether reset occurs after the transition agrees with the corresponding assignment of $\mathbb{b}_\omega$. Constraint 1 shows that rows in $S \cup S_+ \cup R$ assigned to the same location must be either all accepting or all non-accepting. Constraint 2 shows that the resulting candidate automaton is deterministic, and during the construction of transitions using the partition function (Definition 9), no two of the time values μ_i are in the same region. Constraint 3 shows each row in $S \cup S_+ \cup R$ corresponds to some location. Constraint 4 is for efficiency only, the condition that rows in S are assigned to different rows are already enforced by other constraints.

We prove by induction that for each row $\omega \in S \cup S_+ \cup R$, the location reached by ω in $\mathcal{H}$ agrees with its assignment $\overline{\mathbb{q}_\omega}$. This is clear for the empty timed word ϵ. Now suppose ω is of the form $\omega_1 \cdot (\sigma_n, t_n)$. Since $S \cup S_+ \cup R$ is prefix-closed, we have $\omega_1 \in S \cup S_+ \cup R$ and the location reached by ω_1 agrees with $\overline{\mathbb{q}_{\omega_1}}$. Then, according to the construction, the auxiliary transition $(\overline{\mathbb{q}_{\omega_1}}, \sigma, \psi, \overline{\mathbb{b}_\omega}, \overline{\mathbb{q}_\omega})$ with ψ equal to the clock valuation after executing ω. Then, according to the partition function, a transition will be formed with guard containing ψ from $\overline{\mathbb{q}_{\omega_1}}$ to $\overline{\mathbb{q}_\omega}$ and with action σ. This shows the location reached by ω agrees with $\overline{\mathbb{q}_\omega}$.

Next, the fact that each row $\omega \in S \cup S_+ \cup R$ is accepted by the hypothesis $\mathcal{H}$ iff $\mathsf{MQ}(\omega) = +$ follows from the above statement, together with the definition of $Q_\mathcal{H}$ and $F_\mathcal{H}$. Finally, for any two rows $\omega_1, \omega_2 \in S \cup S_+ \cup R$, if the value of f on ω_1, ω_2 and the setting of reset variables $\overline{\mathbb{b}}$ is $\perp$, then $\overline{\mathbb{q}_{\omega_1}} \neq \overline{\mathbb{q}_{\omega_2}}$ follows from Constraint 1, and by the statement proved above, they also reach different locations in $\mathcal{H}$.

Proof (of Theorem 2). Termination of the algorithm is proved by a simulation argument, comparing against a brute-force search version of the algorithm similar to the normal teacher case in [4]. The brute-force search constructs a binary tree of observation tables, with each branching corresponds to a choice of reset for some row in $S \cup R$ (since the reset information is now certain, the observation tables are the usual ones for L^* algorithm, and the category S_+ is not needed). The search is performed in the breadth-first manner, with respect to the number of rows in S. Along the branch where all choice of resets agree with the target automaton, one correct automaton will be eventually learned. This is because different rows in S must arrive at different symbolic states (a pair of location and region) in the automaton, so the size of S is bounded by the number of symbolic states (number of locations multiplied by the number of regions). In practice, the use of partition function means the correct automaton is found when S is much smaller than the number of symbolic states. Along other branches, a correct automaton may or may not be obtained. In the latter case the number of rows in S increase indefinitely. Hence, the breadth-first search must terminate with learning a correct automaton.

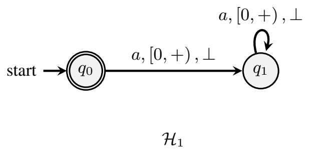
Our algorithm simulates the breadth-first search, with the value of N in the observation table corresponding to the current size of S. Rows in S in our algorithm corresponds to rows that appear in S along all branches of the search tree, and rows in S_+

corresponds to rows that appear in S along some, but not all branches of the search tree. Each model generated by constraint solving corresponds to considering some branch of the tree, but with counterexamples added to all branches at the same time. This shows our algorithm terminates with learning a correct automaton as well.

Appendix B Detailed Running Example

The following Fig. 3 illustrates the detailed learning steps for the DOTA $\mathcal{A}$ in Fig. 1. In Step 1, the observation table $\mathcal{O}_1 = (S, S_+, R, E, f, N)$ is initialized with $S = \{\epsilon\}, S_+ = E = \emptyset, R = \{(a,0)\}, N = 1$, then move_to_S operation will move $(a,0)$ to S and add $(a,0)(a,0)$ to R since $(a,0)$ is distinct from ϵ under any reset. Since $(a,0)(a,0)$ cannot be distinguished from $(a,0) \in S$, the first observation table $\mathcal{O}_1$ is prepared to encode constraints $Cons_1 = C_1 \wedge C_2 \wedge C_3 \wedge C_4$. By solving it with the SMT solver, we obtain a model M of the reset variables and location variables. After the hypothesis construction, the DOTA $\mathcal{H}_1$ is built from $\mathcal{O}_1$ and M. By performing an equivalence query for $\mathcal{H}_1$, the teacher returns a counterexample $ctx_1 = (a,4)$. The counterexample is added to R since it is not distinct from ϵ. O_2 is prepared now. After the same process, we get the hypothesis $\mathcal{H}_2$ shown in the Step 2. The teacher returns a counterexample $(a,4)(a,5.5)$ and we add it in R after Step 3. However, the inconsistency is find that $(a,4)(a,5.5)$ and $(a,9.5)$ end in the same region under $\mathbb{b}_0 = \mathbb{b}_3 = \mathbb{b}_5 = \bot$, but the timed words $(a,4)$ and ϵ can not be distinguished with suffix $e = \epsilon$. Then the timed word $(a, \mathsf{min}(5.5, 9.5)) \cdot e$, i.e., $(a,5.5)$ is added to E. The situation in Example 4 occurs at Step 4, and as introduced in the example we can obtain the table $\mathcal{O}_5$ in Step 5 by moving $(a,4)(a,5.5)$ to S_+. After equivalence query, teacher returns counterexample $(a,4)(a,0)$. Two timed words $(a,0)$ and $(a,0)(a,5.5)$ are added to E during constructing the consistency constraint. Now timed word $(a,4) \in R$ is certainly distinct from all the rows in S, which violates Constraint 3, an thus the corresponding constraint $Cons_6$ built from $\mathcal{O}_6$ is unsatisfiable. Therefore, $(a,4)$ is moved to S and a correct automaton $\mathcal{H}_7$ is learned in the last step.

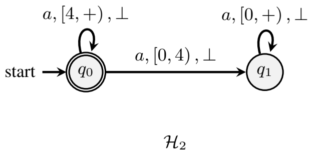
	$\mathcal{O}_1$		$S \cup S_+ \cup R$			E
			ϵ	$(a,0)$	$(a,0)\,(a,0)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\bot$	ϵ
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\top$	
R	$(a,0)(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\top$	



(a) Step 1

counterexample: $(a,4)$

	$\mathcal{O}_2$		$S \cup S_+ \cup R$				E
			ϵ	$(a,0)$	$(a,0)\,(a,0)$	$(a,4)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\bot$	$\top$	ϵ
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\top$	$\bot$	
R	$(a,0)(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\top$	$\bot$	
	$(a,4)$	$\mathbb{b}_3$	$\top$	$\bot$	$\bot$	$\top$	

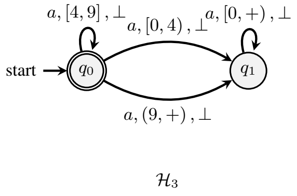


(b) Step 2

Fig. 3: The detailed learning steps for the example DOTA $\mathcal{A}$ in Fig. 1

counterexample: $(a, 9.5)$

$\mathcal{O}_3$			$S \cup S_+ \cup R$					E
			ϵ	$(a,0)$	$(a,0)\,(a,0)$	$(a,4)$	$(a,9.5)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	ϵ
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\top$	$\bot$	$\top$	
	$(a,0)(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\top$	$\bot$	$\top$	
R	$(a,4)$	$\mathbb{b}_3$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	
	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\top$	$\bot$	$\top$	



(c) Step 3

counterexample: $(a, 4)(a, 5.5)$

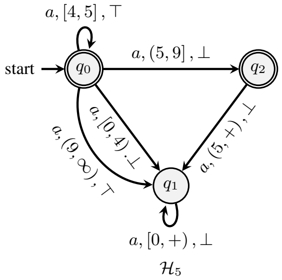
$\mathcal{O}_4$			$S \cup S_+ \cup R$						E
			ϵ	$(a,0)$	$(a,4)\,(a,5.5)$	$(a,0)\,(a,0)$	$(a,4)$	$(a,9.5)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\neg\mathbb{b}_5$	$\bot$	$\mathbb{b}_3$	$\bot$	
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	ϵ
	$(a,4)\,(a,5.5)$	$\mathbb{b}_5$	$\neg\mathbb{b}_5$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$(a, 5.5)$
R	$(a,0)\,(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	
	$(a,4)$	$\mathbb{b}_3$	$\mathbb{b}_3$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	
	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	

UNSAT

(d) Step 4

Move $(a, 4)(a, 5.5)$ to S_+

$\mathcal{O}_5$			$S \cup S_+ \cup R$						E
			ϵ	$(a,0)$	$(a,4)\,(a,5.5)$	$(a,0)\,(a,0)$	$(a,4)$	$(a,9.5)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\neg\mathbb{b}_5$	$\bot$	$\mathbb{b}_3$	$\bot$	
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	ϵ
S_+	$(a,4)\,(a,5.5)$	$\mathbb{b}_5$	$\neg\mathbb{b}_5$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$(a, 5.5)$
	$(a,0)\,(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	
R	$(a,4)$	$\mathbb{b}_3$	$\mathbb{b}_3$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	
	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	



(e) Step 5

Fig. 3: The detailed learning steps for the example DOTA $\mathcal{A}$ in Fig. 1 (Continued).

Counterexample: $(a,4)(a,0)$

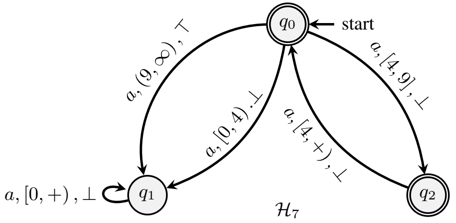
$\mathcal{O}_6$			$S \cup S_+ \cup R$								E
			ϵ	$(a,0)$	$(a,4)\,(a,5.5)$	$(a,0)\,(a,0)$	$(a,4)$	$(a,9.5)$	$(a,4)\,(a,5.5)\,(a,0)$	$(a,4)\,(a,0)$	
S	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\neg\mathbb{b}_5 \wedge \neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_5)$	$\bot$	$\bot$	$\bot$	$\bot$	$\neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_7) \wedge \neg\mathbb{b}_7$	ϵ
	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	
S_+	$(a,4)\,(a,5.5)$	$\mathbb{b}_5$	$\neg\mathbb{b}_5 \wedge \neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_5)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\neg\mathbb{b}_5$	$(a,5.5)$
	$(a,0)\,(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	
	$(a,4)$	$\mathbb{b}_3$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$(a,0)$
R	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	
	$(a,4)\,(a,5.5)\,(a,0)$	$\mathbb{b}_6$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$(a,0)\,(a,5.5)$
	$(a,4)\,(a,0)$	$\mathbb{b}_7$	$\neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_7) \wedge \neg\mathbb{b}_7$	$\top$	$\neg\mathbb{b}_5$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	

UNSAT

(f) Step 6

Move $(a,4)$ to S

$\mathcal{O}_7$			$S \cup S_+ \cup R$								E
			ϵ	$(a,0)$	$(a,4)$	$(a,4)\,(a,5.5)$	$(a,0)\,(a,0)$	$(a,9.5)$	$(a,4)\,(a,5.5)\,(a,0)$	$(a,4)\,(a,0)$	
	ϵ	$\mathbb{b}_0$	$\top$	$\bot$	$\bot$	$\neg\mathbb{b}_5 \wedge \neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_5)$	$\bot$	$\bot$	$\bot$	$\neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_7) \wedge \neg\mathbb{b}_7$	ϵ
S	$(a,0)$	$\mathbb{b}_1$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	
	$(a,4)$	$\mathbb{b}_3$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$(a,5.5)$
S_+	$(a,4)\,(a,5.5)$	$\mathbb{b}_5$	$\neg\mathbb{b}_5 \wedge \neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_5)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\neg\mathbb{b}_5$	
	$(a,0)\,(a,0)$	$\mathbb{b}_2$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$(a,0)$
	$(a,9.5)$	$\mathbb{b}_4$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	
R	$(a,4)\,(a,5.5)\,(a,0)$	$\mathbb{b}_6$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$(a,0)\,(a,5.5)$
	$(a,4)\,(a,0)$	$\mathbb{b}_7$	$\neg(\mathbb{b}_3 \wedge \neg\mathbb{b}_7) \wedge \neg\mathbb{b}_7$	$\top$	$\bot$	$\neg\mathbb{b}_5$	$\top$	$\bot$	$\bot$	$\top$	



(g) Step 7

Fig. 3: The detailed learning steps for the example DOTA $\mathcal{A}$ in Fig. 1, (Continued).

Appendix C Timed Mealy machines

Here, we present an example of timed Mealy machine and some modification of our learning algorithm on TMMs.

C.1 An example of TMM

Fig. 4 illustrates TMM model and MM1T model of alternating-bit protocol sender described in [21].

Compared to MM1T where the timer can be set to integer values on transitions and affect the behavior of the system through timeouts only, a timed Mealy machine is equipped with a clock making it more convenient to set timing constraints which have both upper and lower bounds. Every Mealy machine with one timer can be transformed into a timed Mealy machine. One technical issue is that we need to copy a location if it is set different timer values on different incoming transitions.

Compared to OTAs, TMMs are more convenient for modeling reactive systems with input/output behavior. Since TMMs allow an input-output pair on each transition, we do not need to represent a pair of input/output as two transitions. Moreover, the outputs are provided by the teacher when learning TMMs, instead of having to be learned in the OTA case. This corresponds more naturally to the learning scenario for systems with input/output behavior. All these are motivations to extend our learning algorithm to DTMMs.

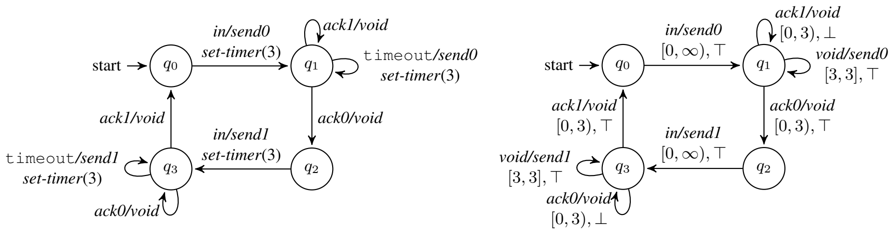


Fig. 4: Left: MM1T model of alternating-bit protocol sender; Right: the corresponding TMM model. In MM1T, at any time, if it receives a input action *in* and moves from q_0 to q_1, the timer is set to 3 and starts countdown in q_1. If receiving no acknowledgement in three time units, `timeout` is triggered and the timer is reset to 3 for waiting for *ack0*. In TMM, `timeout` is replaced by a new special input action *void* since a clock replaces the timer. We do not need to copy q_1 (also q_3) since the timer values set by different transitions are the same.

C.2 Modifications of learning algorithm

Given a timed word ω, a DTMM produces an output sequence rather than whether ω is accepted. This provides richer information for distinguishing between timed words.

More precisely, the membership query is modified so that on given a timed word over input actions ω, the teacher returns the output sequence corresponding to ω. For the equivalence query, the teacher determines whether $\mathcal{M}_{\mathcal{H}} \approx \mathcal{M}$ for the current hypothesis DTMM $\mathcal{M}_{\mathcal{H}}$.

The learning algorithm is modified according to the changes in the membership query. In particular, the procedure for distinguishing two rows under last resets (the test $T(\omega_1, \omega_2, i_1, i_2, e)$ in Section 3.1) is modified. Instead of comparing whether the two timed words are accepted, we compare the output of the two queries *in the portion of suffix* e. This is because the output on ω_1 and ω_2 may be different even if they reach the same location. Hypothesis construction (Section 3.4) is modified to take into account additional output information from queries. The other parts of the algorithm are unchanged. The modification from DOTA to DTMM case is similar to the modification from learning DFA to Mealy machines, for example described in [16].