SAT-Based Synthesis of Minimal Deterministic Real-Time Automata via 3DRTA Representation

Junjie Meng[1], Jie An[2,3(✉)], Yong Li[4(✉)], Andrea Turrini[4,5(✉)], and Miaomiao Zhang[1(✉)]

[1] School of Computer Science and Technology, Tongji University, Shanghai, China
{2311452,miaomiao}@tongji.edu.cn
[2] National Key Laboratory of Space Integrated Information System, Institute of Software Chinese Academy of Sciences, Beijing, China
anjie@iscas.ac.cn
[3] University of Chinese Academy of Sciences, Beijing, China
[4] Key Laboratory of System Software (Chinese Academy of Sciences), Institute of Software Chinese Academy of Sciences, Beijing, China
{liyong,turrini}@ios.ac.cn
[5] Institute of Intelligent Software, Guangzhou, China

Abstract. Real-time automata (RTAs) can be viewed as a subclass of timed automata with only one clock that resets at each transition. In this paper, we propose a novel framework for learning deterministic RTAs (DRTAs) with minimal number of states from samples. Inspired by recent advances in learning deterministic finite automata, we introduce 3-valued Deterministic Real-Time Automata (3DRTAs) as an intermediate representation for the given sample set, thereby eliminating the redundancies present in existing approaches. Then, we solve the minimal DRTA learning problem from 3DRTAs by a reduction to a Boolean Satisfiability (SAT) problem. This then allows us to leverage state-of-the-art SAT solvers to find a minimal DRTA consistent with the given samples efficiently. More importantly, small DRTAs not only offer compact representation but also better interpretability of real-world systems. Experimental results demonstrate that our 3DRTA-based framework yields minimal DRTAs with significantly fewer states compared to those of existing methods. The proposed technique also opens new possibilities for scalable real-time automata learning in complex real-time domains.

Keywords: Real-time automata · Model learning · Automata learning · Passive learning · Grammatical inference · SAT solving

1 Introduction

Real-time systems are ubiquitous in modern computing, spanning from embedded controllers in automotive systems to network protocols and industrial automation. Understanding and modeling the temporal and real-time behavior of such systems is crucial for ensuring correctness, safety, and performance. Unlike traditional discrete systems that only consider the sequence of events, real-time

systems must also account for the precise timing of when events occur, as timing violations can lead to system failures or safety hazards.

Timed automata (TAs) [1] have emerged as a fundamental modeling formalism for real-time systems. By extending finite automata with real-valued clocks and timing constraints, TAs provide a natural and expressive framework for specifying temporal and real-time behaviors. The success of timed automata in modeling and verification [9, 10] has been demonstrated across numerous domains, from communication protocols [27] to scheduling algorithms [6] and safety-critical systems [23]. However, manually constructing timed automata models for complex real-time systems is both time-consuming and error-prone. This challenge motivates the need for automatic learning techniques that can infer timed automata models from observed system behaviors. Such learning approaches are particularly valuable when dealing with legacy systems where formal specifications are unavailable, or when validating that implementations conform to their intended temporal and real-time specifications.

The identification of timed automata from labeled samples represents a significant extension of the well-established finite automata learning problem. While automata learning has been extensively studied in the grammatical inference community [22], the addition of timing constraints introduces fundamental new challenges. Unlike discrete automata, where the primary concern is structural consistency, timed automata must satisfy both structural and timing constraints, dramatically increasing the complexity of the synthesis problem.

Recent advances in deterministic finite automata (DFAs) learning have shown the power of SAT-based synthesis approaches [21], which encode the learning problem as a Boolean satisfiability problem to find globally minimal solutions. A breakthrough in this direction is the introduction of 3-valued DFAs (3DFAs) [14], which provide a compact intermediate representation that dramatically reduces the problem size compared to the traditional Augmented Prefix Tree Acceptor (APTA) constructions. A 3DFA extends classical DFAs by allowing states to be associated with three possible outcomes: "accept", "reject", or "don't care" (undefined). This additional flexibility enables the construction of significantly smaller intermediate structures that still capture all the constraints from the labeled samples, leading to a more efficient SAT encoding with fewer variables.

The success of 3DFAs in DFA learning provides valuable insights for tackling the more complex problem of timed automata synthesis. However, as observed in [7], the introduction of clocks causes an exponential blow-up in the region graph and a sharp increase in constraint complexity, rendering synthesis and verification in the timed setting substantially harder than in the discrete case.

Among the various classes of timed automata, we focus on identifying a simple type of one-clock deterministic timed automata (1-DTAs), known as Deterministic Real-Time Automata (DRTAs) [15]. A DRTA models only the time constraints between two consecutive events, instead of arbitrary pairs of events. It can be viewed as an 1-DTA in which the clock resets at every transition. We focus on DRTAs, a restricted subclass of 1-DTAs, rather than the full class of 1-DTAs, since even the identification of DFAs is already a difficult problem:

it is NP-complete [17] and inapproximable within a polynomial [28]. Hence, it makes sense to first focus on simple extensions of DFAs. In addition, although DRTAs are a restricted subclass of 1-DTAs, their expressive power suffices for many meaningful applications like learning system models from timestamped logs [35], timing-based security or opacity analysis [37], and timed behavior verification for testing or monitoring [20].

The deterministic nature of DRTAs ensures predictable execution semantics, providing unambiguous transitions that are essential for both synthesis and verification [1]. This determinism, combined with the restriction to consecutive event timing, significantly simplifies the learning problem while maintaining sufficient expressiveness for practical real-time systems.

Several works have been conducted on learning DRTAs from samples. Existing approaches for DRTA learning [35] primarily rely on evidence-driven state-merging (EDSM) algorithms [24], which iteratively merge states in a prefix tree based on heuristics. While these methods have shown practical utility, they have their own *limitations*: (i) they often terminate at local optima without global optimality guarantees; (ii) scalability degrades as the underlying APTA grows with alphabet size, trace length, and sample count, inflating the merge search space. In addition to these EDSM-based methods, more recent work, such as the one by Tappler et al. on timed automata learning via SMT solving [32], has explored alternative paradigms. Their approach encodes the synthesis problem into an SMT instance by representing every step of each trace individually, thereby enabling the use of SMT solvers to derive candidate automata. However, this methodology exhibits two significant limitations: first, it only incorporates positive samples into the learning process, disregarding negative evidence; second, the encoding strategy introduces a large amount of redundancy, as each trajectory and each step requires an explicit representation. Consequently, with the growth in the number and length of samples, the number of encoding variables expands rapidly, which in turn exacerbates the scalability challenge and imposes substantial computational burdens on the SMT solver. We provide a detailed discussion in Section 5.2, after presenting our SAT-based encoding.

In this paper, to address these limitations, we propose a novel framework with an advanced intermediate representation that brings the power of modern constraint solving to the DRTA learning from positive and negative examples. Our method guarantees to return a DRTA with the minimal number of states that accepts all positive samples and rejects all negative samples.

Inspired by the success of 3DFAs in discrete automata learning, our approach introduces 3-valued Deterministic Real-Time Automata (3DRTAs) as an intermediate representation that compactly encodes both structural and timing constraints from labeled samples. The 3DRTA extends the 3DFA concept to the timed domain, allowing transitions to carry timing intervals alongside the traditional accept/reject/don't-care state semantics. This novel representation eliminates redundancies present in traditional prefix tree approaches while naturally handling the timing aspects of real-time systems.

By transforming the DRTA synthesis problem into a SAT encoding that leverages the 3DRTA structure, we enable the use of state-of-the-art SAT solvers to find globally optimal solutions. The key insight is that the compact 3DRTA representation leads to significantly smaller SAT formulations compared to direct approaches, making the synthesis problem tractable for practical applications while providing guarantees about the size of the synthesized automaton.

Contributions. Our main contributions in this paper are the following:

- In Section 2, after reviewing basic concepts on timed systems, we formalize the problem of learning DRTAs from labeled samples and we introduce 3DRTAs, a novel intermediate representation that extends 3DFAs to the timed domain, compactly encoding timing constraints from labeled samples while eliminating redundancies present in traditional prefix tree constructions.
- In Section 3, we show how to construct a 3DRTA consistent with a given set of samples, i.e., that accepts all positive samples and rejects all negative samples.
- In Section 4, we present a complete algorithm for synthesizing minimal DRTAs with theoretical guarantees on correctness and optimality, overcoming the limitations of heuristic-based approaches.
- In Section 5, we develop an efficient SAT encoding for DRTA synthesis that leverages the 3DRTA structure, transforming the learning problem into a constraint satisfaction problem solvable by modern SAT solvers with global optimality guarantees.
- In Section 6, we propose a region merging algorithm to reduce the excessive number of transitions that may arise in coarse DRTAs, thereby further optimizing the model size.
- In Section 7, we provide a comprehensive experimental evaluation, demonstrating significant efficiency improvements and superior model quality compared to existing methods.

1.1 Related work

The most closely related work spans timed automata learning, constraint-based synthesis, and compact automata representations.

In timed automata learning, we omit the introduction to the works on Angluin's L^*-style active learning [5], in which the target is to learn automata from an oracle by queries [2–4, 18, 30, 31, 33, 36]. We focus on reviewing related work on passive learning, i.e., learning automata from given samples. Verwer et al. [34] showed that one-clock deterministic timed automata are efficiently identifiable in the limit, providing theoretical learnability results for this restricted class. Building on this, Verwer et al. [35] proposed the RTI algorithm for learning DRTAs using EDSM-based approaches, demonstrating that DRTA learning can outperform sampling-based methods. Tappler et al. [32] proposed an SMT-driven methodology that encodes the problem of inferring timed automata from timestamped traces as SMT formulas and employs incremental solving along with varied search strategies. Grinchtein et al. [19] proposed passive learning

algorithms for event-recording automata (ERAs), which are a specific kind of timed automata where each action is assigned a clock to record the time length from its last occurrence to the present. However, Dima [15] pointed out that RTAs are incomparable to ERAs since RTAs may accept languages consisting of two actions separated by an interval with integer length while ERAs may not.

In constraint-based synthesis for DFAs, recent work [21,38] has demonstrated the power of SAT-based approaches for finding globally minimal solutions. This inspired us to explore a SAT-based encoding for the synthesis of DRTAs consistent with the given samples.

Another important line of work focuses on compact representations of automata. The goal is to reduce the size of the intermediate structures constructed from samples while still preserving all necessary information for synthesis. Representative examples include symbolic and interval-based extensions of DFAs, which avoid explicit state explosion by encoding sets of transitions more concisely [13, 16]. A recent work by Dell'Erba et al. [14] based on the 3DFA construction falls into this category, showing how 3-valued states provide a much smaller intermediate representation compared to traditional APTA. However, adapting these techniques to timed automata presents unique challenges due to the hybrid discrete-continuous nature of timing constraints.

Our work represents the first systematic attempt to bring both the theoretical insights of 3-valued automata and the practical power of constraint-based synthesis to DRTA learning. Unlike previous heuristic-based approaches, we provide theoretical guarantees on the minimality of synthesized models while achieving practical efficiency through our novel 3DRTA representation. In addition, we introduce a time-region merging algorithm to address the issue of overly coarse DRTAs containing an excessive number of transitions, thereby further reducing the model size. This combination of optimality and efficiency represents a significant advancement in timed automata learning.

2 Timed Systems and Languages

Throughout this paper, we fix a finite alphabet Σ and the set of non-negative real numbers $\mathbb{R}_{\geq 0}$ as the time domain; we denote by $[\mathbb{R}_{\geq 0}]$ the set of intervals in $\mathbb{R}_{\geq 0}$ with endpoints in $\mathbb{N} \cup \{\infty\}$.

Timed systems and languages. A (finite) *timed word* over $\Sigma \times \mathbb{R}_{\geq 0}$ is a finite sequence $\omega = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots (\sigma_n, \tau_n)$, where $\sigma_i \in \Sigma$ and $\tau_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq n$; $|\omega| = n$ denotes its length. The empty timed word ε satisfies $|\varepsilon| = 0$. For two timed words ω_1 and ω_2, their concatenation $\omega_1 \cdot \omega_2$ (abbreviated as $\omega_1\omega_2$) is defined by appending ω_2 to ω_1. A *timed language* $\mathcal{L}$ is a set of timed words, i.e., $\mathcal{L} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$.

A *deterministic real-time transition system* (DRTS) is defined as a triple $\mathcal{T} = (Q, q_0, \Delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\Delta \colon Q \times (\Sigma \times [\mathbb{R}_{\geq 0}]) \to Q$ is a transition relation with $|\Delta| < \infty$. The continuous progress of real time is tracked by the unique clock c, which resets at every transition, so the value of c represents the delay time between two

actions. Thus, Δ induces the transition function $\delta\colon Q \times (\Sigma \times \mathbb{R}_{\geq 0}) \to 2^Q$ such that $\delta(q,(\sigma,\tau)) = \{\, q' \in Q \mid (q,(\sigma,I),q') \in \Delta \wedge \tau \in I \,\}$, where $I \in [\mathbb{R}_{\geq 0}]$ is the region interval having $\tau \in I$ associated with the action σ. We extend δ to timed words by setting $\delta(q,\varepsilon) = \{q\}$ and $\delta(q,(\sigma,\tau)\cdot\omega) = \bigcup_{q' \in \delta(q,(\sigma,\tau))} \delta(q',\omega)$, and subsequently to sets of states $Q' \subseteq Q$ by $\delta(Q',\omega) = \bigcup_{q \in Q'} \delta(q,\omega)$.

The run of $\mathcal{T}$ on a finite word $\omega = (\sigma_1,\tau_1)\cdots(\sigma_n,\tau_n)$ is the sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \leq i < n$, the subsequent state q_{i+1} is determined by $q_{i+1} = \delta(q_i,(\sigma_{i+1},\tau_{i+1}))$.

Definition 1. *A* Deterministic Real-Time Automaton *(DRTA) is a pair* $\mathcal{A} = (\mathcal{T},A)$ *with* $\mathcal{T} = (Q,q_0,\Delta)$ *being a DRTS and* $A \subseteq Q$ *the set of accepting states.*

For every state $q \in Q$ and input symbol $a \in \Sigma$, the set of time intervals of outgoing transitions labeled by a must be *pairwise disjoint*, that is, for any two transitions $(q,(a,I_1),q_1)$, $(q,(a,I_2),q_2) \in \Delta$, we have $I_1 \cap I_2 = \emptyset$ whenever $q_1 \neq q_2$ and $I_1 = I_2$ whenever $q_1 = q_2$. This ensures that for each time value τ, at most one transition labeled a is enabled from state q.

A timed word $\omega \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ is accepted by $\mathcal{A}$ if there exists a run on ω whose last state is in A; the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. As real-time automata are determinable [15], we say that a timed language is a *real-time language* if it can be recognized by a DRTA.

Now, we can formalize the problem of learning minimal DRTAs from labeled samples as follows.

Definition 2 (Minimal DRTA synthesis problem). *Given a finite set of labeled timed words* $S = (S^+,S^-) \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$*, construct a DRTA* $\mathcal{A} = (\mathcal{T},A)$ *that satisfies the following conditions:*

(i) ***Consistency:*** *for each* $\omega^+ \in S^+$*,* $\omega^+ \in \mathcal{L}(\mathcal{A})$ *and for each* $\omega^- \in S^-$*,* $\omega^- \notin \mathcal{L}(\mathcal{A})$*;*
(ii) ***Minimality:*** *among all DRTAs consistent with* S*,* $\mathcal{A}$ *has the smallest number of states.*

To solve the above problem, we introduce 3-valued deterministic real-time automata, which extend DRTAs to process languages with "don't-care" words.

Definition 3. *A* 3-valued Deterministic Real-Time Automaton *(3DRTA) is defined as a triple* $\mathcal{A} = (\mathcal{T},A,R)$ *where* $\mathcal{T} = (Q,q_0,\Delta)$ *is a DRTS and* A *and* R*, together with* $D = Q \setminus (A \cup R)$*, form a partition of the set of states* Q*.* $A \subseteq Q$ *is the set of* accepting *states,* $R \subseteq Q$ *is the set of* rejecting *states, and the remaining states* D *are called* don't-care *states.*

A run of $\mathcal{A} = (\mathcal{T},A,R)$ is *accepting* (respectively, *rejecting*) if it ends in an accepting (resp. rejecting) state. A timed word ω is *accepted* (resp. *rejected*) by $\mathcal{A}$ if it has an accepting (resp. rejecting) run on ω. 3DRTAs map all words in $(\Sigma \times \mathbb{R}_{\geq 0})^*$ to *three* values: accepting (+), rejecting (−), and don't-care (?), where they are accepting if they have an accepting run, rejecting if they have a rejecting run, and don't-care otherwise. Note that DRTAs are a special type of

3DRTAs with only accepting and rejecting states, that is, $D = \emptyset$ and $A \cup R = Q$. As for DRTAs, we denote the language of a 3DRTA $\mathcal{A}$ by $\mathcal{L}(\mathcal{A})$, i.e., the set of all words accepted by $\mathcal{A}$.

Following An et al.'s development of a Myhill–Nerode style theorem for real-time languages [4], we know that there exists a unique minimal DRTA recognizing a given real-time language. Similarly, as suggested by Chen et al. [11] for DFAs, we can identify equivalent words that reach the same state in the 3DRTA recognizing a function $L\colon (\Sigma \times \mathbb{R}_{\geq 0})^* \to \{+, -, ?\}$. We define the equivalence relation $\sim_L \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^* \times (\Sigma \times \mathbb{R}_{\geq 0})^*$ over timed words as follows:

$$\omega_1 \sim_L \omega_2 \quad \Longleftrightarrow \quad \forall \omega' \in (\Sigma \times \mathbb{R}_{\geq 0})^* : L(\omega_1 \cdot \omega') = L(\omega_2 \cdot \omega').$$

Here, ω' denotes a possible suffix extension that can follow either ω_1 or ω_2. We denote by $|\sim_L|$ the *index* of the equivalence relation $\sim_L$, that is, the number of its equivalence classes.

Given a finite set of labeled samples $S = (S^+, S^-)$ in $(\Sigma \times \mathbb{R}_{\geq 0})^*$, we interpret S as a classification function that induces an equivalence relation $\sim_S$. For any timed word $\omega \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, the classification of ω is given by:

$$S(\omega) = \begin{cases} + & \text{if } \omega \in S^+, \\ - & \text{if } \omega \in S^-, \\ ? & \text{otherwise.} \end{cases}$$

Finally, as S is a finite set, suppose it can be recognized by a DRTA $\mathcal{A}$ and so by a 3DRTA, we conclude with a straightforward proposition: since the number of equivalence classes $|\sim_S|$ induced by the sample set S is bounded by the number of equivalence classes $|\sim_{\mathcal{L}(\mathcal{A})}|$ of the potential target real-time language $\mathcal{L}(\mathcal{A})$, which is finite [4], it follows that $\sim_S$ has only finitely many equivalence classes.

Region words and regionization. Referring to the Myhill-Nerode theorem for real-time languages [4], we now introduce the region abstraction mapping time points to time intervals. As a DRTA can be viewed as a one-clock timed automaton where the clock resets after each action, the time domain can be partitioned into finitely many equivalence classes called *regions*. Given a constant $\kappa \in \mathbb{N}$, we define the region $[\![v]\!]_\kappa$ containing $v \in \mathbb{R}_{\geq 0}$ as follows: if $v \leq \kappa$, then $[\![v]\!]_\kappa = [v, v]$ if $v \in \mathbb{N}$, and $[\![v]\!]_\kappa = (\lfloor v \rfloor, \lfloor v \rfloor + 1)$ otherwise, where $\lfloor v \rfloor$ is the integer part of v; if $v > \kappa$, then $[\![v]\!]_\kappa = (\kappa, \infty)$. The constant κ induces a finite partition of the non-negative real line into a set of disjoint regions, each corresponding to either an integer point or an open interval between two consecutive integers. Formally, $\mathbb{R}_{\geq 0}$ is partitioned into $2\kappa + 2$ regions: $[n, n]$ for $0 \leq n \leq \kappa$, $(n, n+1)$ for $0 \leq n < \kappa$, and (κ, ∞), with $n \in \mathbb{N}$. We let $[\![\mathbb{R}_{\geq 0}]\!]_\kappa$ denote the set of all regions.

Definition 4 (Region word). *Given a timed word* $\omega = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$, *the* region word *of* ω *is defined as* $[\![\omega]\!]_\kappa = (\sigma_1, [\![\tau_1]\!]_\kappa) \cdots (\sigma_n, [\![\tau_n]\!]_\kappa)$, *where* $\kappa = \lceil \max\{\tau_i \mid 1 \leq i \leq n\} \rceil$.

To make the construction below consistent with timed automata semantics, we assume that the labeled samples are *region-consistent*: for any two timed words ω_1, ω_2, if $[\![\omega_1]\!]_\kappa = [\![\omega_2]\!]_\kappa$ then it is not the case that $\omega_1 \in S^+$ and $\omega_2 \in S^-$. Equivalently, no two samples that differ only by choosing timestamps within the *same* region may receive conflicting labels. For example, if $\kappa \geq 7$, then $(a, 6.4)$ and $(a, 6.9)$ both lie in the region $(6, 7)$, so having $(a, 6.4) \in S^+$ and $(a, 6.9) \in S^-$ is disallowed. Timed automata with integer-bounded constraints on their transitions generate region-consistent timed words.

Given a timed word ω, we denote by $\omega[i]$ the i-th timed action (σ_i, τ_i), by $\omega[i, k]$ the subword starting at the i-th action and ending at the $(k-1)$-th action if $1 \leq i < k$, and by ε otherwise. Similarly, $\omega[i \cdots]$ represents the suffix starting at the i-th action if $i < |\omega|$, and ε otherwise. A timed word ω' is a *prefix* of ω if $\omega = \omega' \cdot \omega''$ for some $\omega'' \in (\Sigma \times \mathbb{R}_{\geq 0})^*$. The set of all prefixes of ω is denoted as $\mathbf{prefixes}(\omega)$, and for a set S of timed words, $\mathbf{prefixes}(S) = \bigcup_{\omega \in S} \mathbf{prefixes}(\omega)$.

We next establish the relation between a timed word ω and its region word $[\![\omega]\!]_\kappa$, in particular how regionization preserves the prefix structure.

Lemma 1 (Prefix preservation under regionization). *For every timed word $\omega = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$ and every $1 \leq j \leq |\omega|$, the regionization operator $[\![\cdot]\!]_\kappa$ commutes with the prefix operator: $[\![\omega[1, j]]\!]_\kappa = [\![\omega]\!]_\kappa[1, j]$.*

Proof. By Definition 4, the operator $[\![\cdot]\!]_\kappa$ acts letter-wise and it is length-preserving; also, each timed action (σ, τ) is mapped to $(\sigma, [\![\tau]\!]_\kappa)$ without altering the order of actions. Hence, the image of the first j actions of ω under $[\![\cdot]\!]_\kappa$ is exactly the first j actions of $[\![\omega]\!]_\kappa$. Formally, if $\omega = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$, then $[\![\omega[1, j]]\!]_\kappa = (\sigma_1, [\![\tau_1]\!]_\kappa) \cdots (\sigma_j, [\![\tau_j]\!]_\kappa) = [\![\omega]\!]_\kappa[1, j]$. Thus, the claim follows. □

Corollary 1 (Prefixes of sets under regionization). *For any set of timed words $S \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$, we have*

$$[\![\mathbf{prefixes}(S)]\!]_\kappa \;=\; \bigcup_{\omega \in S} \mathbf{prefixes}([\![\omega]\!]_\kappa).$$

Proof. By definition, $\mathbf{prefixes}(S) = \bigcup_{\omega \in S} \mathbf{prefixes}(\omega)$. Applying Lemma 1, for each $\omega \in S$ we obtain $[\![\mathbf{prefixes}(\omega)]\!]_\kappa = \mathbf{prefixes}([\![\omega]\!]_\kappa)$. Taking the union over all $\omega \in S$ yields the result. □

This equivalence formally clarifies the relation between ω and $[\![\omega]\!]_\kappa$: reasoning about prefixes can be carried out either at the timed-word level or at the region-word level, without loss of generality.

3 Construction of 3DRTAs

Given a set of samples $S = (S^+, S^-)$, that we also write as $S = S^+ \uplus S^-$, our aim is to construct a DRTA $\mathcal{A}$ consistent with S. In the RTI algorithm proposed by Verwer et al. [35], the learning process requires the construction of a timed Augmented Prefix Tree Acceptor (APTA) prior to performing state merging.

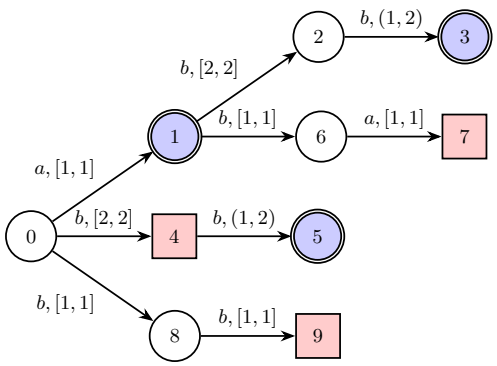

Fig. 1. The TAPTA for the sample set of timed words $S = (S^+, S^-)$, where $S^+ = (\{(a,1),(a,1)(b,2)(b,1.1),(b,2)(b,1.6)\}$ and $S^- = \{(a,1)(b,1)(a,1),(b,2),(b,1)(b,1)\})$. Blue double-circled round nodes are accepting states while red square nodes are rejecting states.

Similarly, in the approach of Tappler et al. [32], each step of every trace must be explicitly encoded into SMT formulas, which essentially correspond to a larger, unmerged timed APTA structure. Inspired by Verwer's timed APTA construction, we propose a related algorithm, termed TAPTA, outlined in Algorithm 1. The key distinction lies in how prefixes are treated: Verwer's timed APTA disregards timing information when identifying prefixes, which necessitates the subsequent use of a split operation to separate accepting and rejecting states. The split operation divides a transition $(q, (\sigma, I), q')$ into two or more sub-transitions whenever the time interval I contains both accepting and rejecting samples. A new boundary point is chosen to partition I into disjoint sub-intervals $I_1, I_2, \ldots$, and new target states are introduced so that accepting and rejecting behaviors are separated accordingly. In contrast, our TAPTA incorporates timing into the prefix representation, thereby eliminating the need for an additional split step and allowing the structure to be encoded directly.

A TAPTA for the set of samples $S = (S^+, S^-)$ requires consideration of both identical *action prefixes* σ and *timed region prefixes* $[\![\tau]\!]_\kappa$ in the DRTA, where $\kappa = \lceil \max\{\, \tau \in \mathbb{R}_{\geq 0} \mid (\sigma, \tau) \text{ appears in } S \,\} \rceil$. We first map each word ω to its corresponding region word $[\![\omega]\!]_\kappa$. In a TAPTA, each state corresponds to a unique prefix of a region word in S. First, define a function $f\colon \mathbf{prefixes}(S) \to \mathbb{N}_S$, where $\mathbb{N}_S = \{0, 1, \ldots, |\mathbf{prefixes}(S)| - 1\}$. Intuitively, f maps each prefix $u \in \mathbf{prefixes}(S)$ to a state in the TAPTA represented by a unique number in $\mathbb{N}_S$. Formally, a TAPTA $\mathcal{P}$ for S is a 3DRTA $(\mathcal{T}, A, R)$ where the DRTS $\mathcal{T}$ consists of the state set $\mathbb{N}_S$, initial state $f(\varepsilon)$, and transition function Δ defined as $\Delta(i, (a, I)) = j$ if $f([\![\omega]\!]_\kappa) = i$ and $f([\![\omega]\!]_\kappa(a, I)) = j$, where $[\![\omega]\!]_\kappa, [\![\omega]\!]_\kappa(a, I) \in \mathbf{prefixes}(S)$, $a \in \Sigma$, and $I \in [\![\mathbb{R}_{\geq 0}]\!]_\kappa$; we define $A = \{\, i \in \mathbb{N}_S \mid f(u) = i \text{ for some } u \in S^+ \,\}$ and $R = \{\, i \in \mathbb{N}_S \mid f(u) = i \text{ for some } u \in S^- \,\}$.

For example, the TAPTA shown in Fig. 1 is generated from the sample set $S = (S^+, S^-)$, where S^+ contains the three accepted timed words $(a, 1)$,

Algorithm 1: BUILDTIMEDAPTA

Input: $S = \{S^+, S^-\}$: labeled timed traces
Output : $\mathcal{P}$: a deterministic real-time automaton (TAPTA)

$\kappa \leftarrow \lceil \max\{\, \tau \in \mathbb{R}_{\geq 0} \mid (\sigma, \tau) \text{ appears in } S \,\} \rceil$; ▷ `Determine maximum constant`
$[\![\mathbb{R}_{\geq 0}]\!]_\kappa \leftarrow \{[0,0], (0,1), [1,1], \ldots, [\kappa,\kappa], (\kappa,\infty)\}$; ▷ `Time regions`
$\mathcal{P} \leftarrow (\mathcal{T} = (Q = \{0\}, q_0 = 0, \Delta = \emptyset), A = \emptyset, R = \emptyset)$; ▷ `Initialize TAPTA`
foreach *timed trace* $\omega = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n) \in S$ **do**
  $q \leftarrow 0$; ▷ `Start from initial state`
  for $i \leftarrow 1$ **to** n **do**
    $\tau \leftarrow \tau_i$, $\sigma \leftarrow \sigma_i$;
    if *there is no transition* $(q, \sigma, [\![\tau]\!]_\kappa, q')$ *in* Δ **then**
      $q_{\text{new}} = |Q|$; ▷ `Create a new, fresh state`
      $Q \leftarrow Q \cup \{q_{\text{new}}\}$; ▷ `Add it to the TAPTA`
      $\Delta \leftarrow \Delta \cup \{q \xrightarrow{\sigma, [\![\tau]\!]_\kappa} q_{\text{new}}\}$; ▷ `Add the corresponding transition`
    $q \leftarrow \Delta(q, (\sigma, [\![\tau]\!]_\kappa))$; ▷ `Move to next state`
  if $\omega \in S^+$ **then**
    $A \leftarrow A \cup \{q\}$; ▷ `Final state is accepting`
  else
    $R \leftarrow R \cup \{q\}$; ▷ `Final state is rejecting`
return $\mathcal{P}$; ▷ `Return the TAPTA`

$(a,1)(b,2)(b,1.1)$, and $(b,2)(b,1.6)$; the three rejected timed words in S^- are $(a,1)(b,1)(a,1)$, $(b,2)$, and $(b,1)(b,1)$; and we have $f = \{\varepsilon \mapsto 0, (a,[1,1]) \mapsto 1, (a,[1,1])(b,[2,2]) \mapsto 2, (a,[1,1])(b,[2,2])(b,(1,2)) \mapsto 3, \ldots\}$. The set of accepting states is $A = \{1,3,5\}$ and the set of rejecting states is $R = \{4,7,9\}$.

3DRTA construction. For simplicity of presentation, we assume that the full TAPTA $\mathcal{P}$ from S is given. Note that our 3DRTA can also be constructed on-the-fly from S using the same techniques presented in [12,14]. Our reduction process works in a backward manner as follows: as data structure, it uses a hash map called `Register` mapping equivalent states to the same representative state.

Initially, we map all accepting states without outgoing transitions into one single representative state and store them in the `Register`. Since each rejecting state is not equivalent to other states, each rejecting state is stored as its own representative in the `Register`. Then our reduction procedure iteratively traverses the TAPTA states in a backward manner from the leaves towards the root and processes the 3DRTA as follows. In each iteration, we first collect the states whose all successors are representative states in `Register`, and then identify equivalent states. Based on the definition of $\sim_S$, we define that two states $p, q \in Q$ are equivalent, denoted $p \equiv q$ if, and only if:

- both states share identical acceptance properties (i.e., both are accepting, rejecting, or don't-care states);
- for every input symbol $\sigma \in \Sigma$ and time region $I \in [\![\mathbb{R}_{\geq 0}]\!]_\kappa$, they either both lack transitions or share the same successor state in the `Register`.

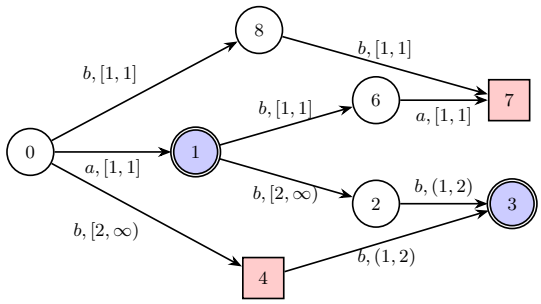

Fig. 2. The 3DRTA $\mathcal{A} = (\mathcal{T}, A, R)$ constructed from the TAPTA in Fig. 1, where $A = \{1, 3\}$ and $R = \{4, 7\}$.

A representative state is then created for each equivalence class and stored in the `Register`. All equivalent states in the DRTA are replaced by their representative. This reduction iterates until all states, including the initial state q_0, are processed, yielding a minimal DRTA consistent with the labeled timed sample set S. The temporal consistency enforced by time regions ensures that merged states preserve both symbolic and real-time behaviors of the original automaton.

Theorem 1. *Given a set of samples $S = (S^+, S^-)$, the 3DRTA construction produces a 3DRTA consistent with S.*

Proof. From the samples $S = (S^+, S^-)$, first the TAPTA for it is constructed. By a simple inspection of the construction and by Lemma 1, it is easy to observe that the TAPTA is consistent with S, as any (regionized) sample word s in S leads to a unique state that is accepting (rejecting, resp.) if and only if $s \in S^+$ ($s \in S^-$, resp.); inner states reached by prefixes not in S are "don't-care" states.

Similarly, in the 3DRTA construction, only leaf states of the same type are merged, and inner states are merged only if there is no outgoing transition that can distinguish them (by e.g. having a different input symbol σ, or a different time region I, or a different target state). This, together with Lemma 1 and Corollary 1, ensures that consistency with S is preserved. □

Note that our 3DRTA can also be constructed *on the fly* from S in the same manner as in [14] provided the sample words are taken out from S in the standard lexicographical order.

As an example, this construction process can merge the states 3 and 5 of the TAPTA in Fig. 1 reached by positive samples into the representative state 3, and merge the states 7 and 9 reached by negative samples into the representative state 7, as shown in Fig. 2. As clarified in Definition 4, every timestamp in a timed word is first mapped to its corresponding region form. Consequently, precise integer points such as 1 remain as single-point regions $[1, 1]$. In contrast, if the maximum integer delay κ actually occurs in the sample set, the corresponding timestamps $\tau = \kappa$ still belongs to the singleton region $[\kappa, \kappa]$; the tail region (κ, ∞) is reserved exclusively for delays strictly greater than κ. Accordingly,

Algorithm 2: SAT-based DRTA synthesis

Input: $S = (S^+, S^-)$: positive and negative timed trace samples
Output : $\mathcal{A}$: a deterministic real-time automaton (DRTA)

```
1  A ← BuildTimed3DRTA(S⁺, S⁻);            ▷ Construct 3DRTA structure
2  n ← 2;                                  ▷ Initialize minimum state count
3  while true do
4      Φ_n ← Encode(A, n);                 ▷ Generate SAT constraints
5      SAT, D ← Solve(Φ_n);                ▷ Check Φ_n satisfiability
6      if SAT then
7          D' ← OptimizeTransitions(D);    ▷ Apply region merging
             optimization
8          return D';                      ▷ Return synthesized DRTA
9      else
10         n ← n + 1;                      ▷ Increment state count and retry
```

the transition in Fig. 2 from state 1 to state 2 labeled with $(b, [2, 2])$ should be interpreted as covering all delays $\tau \geq 2$ by merging $[2, 2]$ and $(2, \infty)$ in the final simplification step presented in Section 6.

As can be seen from the result, the obtained 3DRTA can be significantly smaller than the original TAPTA. Therefore, by using 3DRTAs instead of the TAPTA in our encoding, we can substantially reduce the number of required variables and constraints, resulting in a more manageable SAT problem and overall faster solving performance, as also confirmed by the experiments in Section 7.

4 SAT-based DRTA Synthesis Framework

In this section we develop a comprehensive SAT-based synthesis framework for DRTAs that integrates minimal automaton construction, constraint encoding, iterative state space exploration, and verification processes. This framework addresses the fundamental challenge of learning temporal automata from timed traces while ensuring both minimality and correctness properties. The overall algorithm is given as Algorithm 2.

The synthesis process begins with the construction of a minimal 3DRTA structure from positive and negative timed trace samples $S = (S^+, S^-)$, where each sample s_i represents a sequence of timed events $(\sigma_1, \tau_1) \cdots (\sigma_l, \tau_l) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$. We assume that S is not trivial, i.e., $S^+ \neq \emptyset$ and $S^- \neq \emptyset$; for trivial cases, DRTAs with universal and empty language can be directly produced.

The synthesis framework employs the BuildTimed3DRTA procedure to generate an initial 3DRTA structure $\mathcal{A} = (\mathcal{T}, A, R)$. We formalize the state space minimization search strategy as an optimization problem $\min_n n$, where n represents the minimum number of states needed by a DRTA consistent with S. The algorithm begins with $n = 2$ (minimum number of states for non-trivial DRTAs) and systematically increments this bound until a satisfiable solution is discovered, ensuring the synthesis of a minimal deterministic automaton.

For each candidate state set cardinality n, the algorithm calls the ENCODE procedure (detailed in Section 5) to transform the temporal learning problem into a Boolean satisfiability instance. The encoding process generates a comprehensive constraint system Φ_n that is then submitted to a SAT solver to determine whether Φ_n encodes a valid DRTA with n states consistent with S. If the formula Φ_n is satisfiable, then the algorithm calls the OPTIMIZETRANSITIONS procedure (given in Section 6) providing the region merging optimization: it applies heuristic methods to minimize the number of temporal regions while preserving determinism and completeness. Otherwise, the number of states is increased to $n + 1$ since no DRTA with n states is consistent with S.

5 3DRTA to SAT Encoding for DRTA Synthesis

We now present the SAT encoding Φ_n returned by the ENCODE$(\mathcal{A}, n)$ procedure for solving the DRTA synthesis problem. Given the initial 3DRTA structure $\mathcal{A} = (\mathcal{T} = (Q, q_0, \Delta), A, R)$ consistent with the sample set $S = (S^+, S^-)$, we are looking for a DRTA $\mathcal{A}_s$ with n states that accepts all positive samples in S^+ and rejects all negative samples in S^-. States, transitions, and accepting states of $\mathcal{A}_s$ are obtained by the model of Φ_n, provided it is satisfiable.

5.1 SAT encoding

To encode the DRTA synthesis problem, we use the following four types of variables, where $[n]$ denotes the set $[n] = \{0, 1, \ldots, n-1\}$; to simplify the presentation, we refer to states and transitions of $\mathcal{A}$ as nodes and edges, respectively, and we use the terms states and transitions for $\mathcal{A}_s$:

- **Node state variables** $x_{v,i}$, where $v \in Q$ and $i \in [n]$. $x_{v,i} \equiv 1$ if and only if the node v of $\mathcal{A}$ is mapped to the state i of $\mathcal{A}_s$.
- **Transition relation variables** $e_{i,\sigma,I,j}$, where $\sigma \in \Sigma$, $I \in [\![\mathbb{R}_{\geq 0}]\!]_\kappa$, and $i, j \in [n]$. $e_{i,\sigma,I,j} \equiv 1$ if and only if $\mathcal{A}_s$ has a transition from state i to state j over symbol σ within time region I.
- **Acceptance variables** z_i, where $i \in [n]$. $z_i \equiv 1$ if and only if state i of $\mathcal{A}_s$ is an accepting state.

We now give the list of constraints for the SAT encoding. First of all, we require that the synthesized $\mathcal{A}_s$ must be a valid DRTA. That is, it must be deterministic, complete, and satisfy the temporal constraints for all sample traces. For this, the following constraints must be enforced:

1. **Positive sample acceptance**: every positive timed trace in S^+ must be accepted by $\mathcal{A}_s$:

$$\bigwedge_{v \in A} \bigwedge_{i \in [n]} x_{v,i} \implies z_i.$$

2. **Negative sample rejection**: every negative timed trace in S^- must be rejected by $\mathcal{A}_s$:
$$\bigwedge_{v\in R}\bigwedge_{i\in[n]} x_{v,i} \implies \neg z_i.$$
3. **State-region mapping**: each node v of $\mathcal{A}$ must be represented in $\mathcal{A}_s$:
$$\bigwedge_{v\in Q}\bigvee_{i\in[n]} x_{v,i}$$
4. **Unique state–region mapping**: let $M = \{\, v \in Q \mid \exists u, u' \in \textbf{prefixes}(S) : u \neq u' \wedge v = \Delta(r,u) = \Delta(r,u') \,\}$ be the set of nodes reached by multiple different prefixes; only states with unique prefixes must have a single representative:
$$\bigwedge_{v\in Q\setminus M}\ \bigwedge_{i,j\in[n],\, i<j} \neg(x_{v,i} \wedge x_{v,j}).$$
5. **Transition relation**: for each $v \in Q$, $\sigma \in \Sigma(v)$, $I \in [\![\mathbb{R}_{\geq 0}]\!]_\kappa$, and $i, j \in [n]$, if state i represents node v and there exists a transition from i to j over (a, I), then j represents node $\Delta(v,(a,I))$:
$$\bigwedge_{v\in Q}\bigwedge_{a\in\Sigma(v)}\bigwedge_{I\in[\![\mathbb{R}_{\geq 0}]\!]_\kappa}\bigwedge_{i,j\in[n]} (x_{v,i} \wedge e_{i,a,I,j}) \implies x_{\delta(v,(a,I)),j}$$
6. **Deterministic transitions**: for the same source state i, when receiving the same symbol σ and two time regions intersect, at most only one marked state can be the successor state:
$$\bigwedge_{\sigma\in\Sigma}\ \bigwedge_{\substack{I_1,I_2\in[\![\mathbb{R}_{\geq 0}]\!]_\kappa \\ I_1\cap I_2\neq\emptyset}}\ \bigwedge_{\substack{i,j,t\in[n] \\ j\neq t}} \neg(e_{i,\sigma,I_1,j} \wedge e_{i,\sigma,I_2,t})$$
7. **Initial node constraint**: the initial node q_0 of $\mathcal{A}$ is mapped to the initial state 0 of $\mathcal{A}_s$:
$$x_{q_0,0}$$
8. **Original edge covering constraint**: each edge of $\mathcal{A}$ must have a corresponding transition in $\mathcal{A}_s$:
$$\bigwedge_{(v,\sigma,I,u)\in\Delta}\ \bigvee_{i,j\in[n]} x_{v,i} \wedge e_{i,\sigma,I,j} \wedge x_{u,j}$$

Encoding size. Regarding the number of clauses in the SAT encoding, the dominant factors are: (i) $\mathcal{O}((|A| + |R|) \cdot n)$ from positive/negative sample consistency (constraints 1 and 2), (ii) $\mathcal{O}(|Q| \cdot n)$ from mapping (constraint 3), (iii) $\mathcal{O}(|Q \setminus M| \cdot n^2)$ from uniqueness (constraint 4), (iv) $\mathcal{O}(|Q| \cdot |\Sigma| \cdot |[\![\mathbb{R}_{\geq 0}]\!]_\kappa| \cdot n^2)$ from transition consistency (constraint 5), (v) $\mathcal{O}(|\Sigma| \cdot |[\![\mathbb{R}_{\geq 0}]\!]_\kappa|^2 \cdot n^2)$ from determinism (constraint 6), and (vi) $\mathcal{O}(|\Delta| \cdot n^2)$ from edge covering (constraint 8). Since $|[\![\mathbb{R}_{\geq 0}]\!]_\kappa| = 2\kappa + 2$, the overall number of constraints is
$$\mathsf{size}(\Phi_n) = \mathcal{O}\big(n^2(|Q||\Sigma|\kappa + |\Sigma|\kappa^2 + |\Delta|) + n(|A| + |R|)\big)$$

that is thus polynomial in the sizes $|Q|$, $|\Sigma|$, κ, $|\Delta|$ of the 3DRTA $\mathcal{A}$, and the number of states n we allow the DRTA synthesis procedure to use.

Correctness of the encoding and minimality. We formalize the connection between satisfiability of the encoding and the existence of a consistent DRTA, and then derive minimality from the size-increasing search over n.

Theorem 2. *Let $S = (S^+, S^-)$ be a finite, internally consistent sample set. For any $n \in \mathbb{N}_{\geq 1}$, the formula $\Phi_n(S)$ produced by* Encode *is satisfiable if and only if there exists a DRTA with n states that is consistent with S.*

Proof. ($\Rightarrow$) Suppose $\Phi_n(S)$ is satisfiable with a model M. Decoding M yields a DRTA $\mathcal{A}_n$ with n states: (i) the determinism and completeness clauses of Φ_n induce a well-formed transition structure over time regions; (ii) structural clauses fix initial/accepting status; and (iii) trace clauses ensure that every $\omega \in S^+$ is accepted and every $\omega \in S^-$ is rejected. Hence $\mathcal{A}_n$ is a DRTA consistent with S.

($\Leftarrow$) Conversely, let $\mathcal{A}_n$ be a DRTA with n states consistent with S. Assign the variables of $\Phi_n(S)$ to match the transitions, time regions, and accepting states of $\mathcal{A}_n$. By construction, determinism/completeness/structure constraints are satisfied, and each labeled trace in S is evaluated in accordance with $\mathcal{A}_n$, so $\Phi_n(S)$ is satisfiable. □

Corollary 2. *Consider the procedure that checks $\Phi_n(S)$ in increasing order of n and returns the first n^* for which $\Phi_{n^*}(S)$ is satisfiable. Then n^* is the minimal number of states among all DRTAs consistent with S.*

Proof. By Theorem 2, $\Phi_n(S)$ is satisfiable exactly when an n-state consistent DRTA exists. Hence the first n^* with $\Phi_{n^*}(S)$ satisfiable is the smallest such n. Starting from $n = 1$ covers trivial cases (empty or universal languages) using a single-state DRTA. □

Hence, the algorithm ensures both theoretical soundness and practical efficiency when synthesizing minimal timed automata from complex timed behavioral specifications, automatically determining the optimal state space cardinality while maintaining correctness properties.

Existing passive DRTA learning, such as the RTI algorithm [35], relies on heuristic state merging and provides no global minimality guarantees. To the best of our knowledge, no prior work is able to construct, from finite samples, a provably minimal DRTA with a complete decision procedure. Our framework fills this gap: the existence and uniqueness of minimal DRTAs is guaranteed by the Myhill–Nerode theorem for real-time languages [4], and we are the first to leverage this result to actually synthesize state-minimal DRTAs from finite labeled samples.

5.2 Comparison with Tappler's SMT-based approach

We now compare our method, based on the 3DRTA representation, with the SMT-based approach for timed automata synthesis proposed by Tappler et

Table 1. Constraint count comparison between Tappler's SMT encoding and our 3DRTA-based encoding across sample sizes.

Samples	10	15	20	25	50	75	100	150	300	500	1000
Tappler	225	394	460	539	1180	1834	2265	3920	6755	11635	22620
3DRTA	125	259	305	352	767	1164	1396	2274	3523	5673	9177

al. [32]. While both approaches aim at synthesizing timed models, they differ fundamentally in modeling assumptions, data usage, and constraint formulation.

Introduction to basic situation. Following Tappler et al. [32], the action alphabet is partitioned into *inputs* and *outputs*, $\Sigma = \Sigma_I \cup \Sigma_O$. By convention, input labels carry a "?" (e.g., $a?$) and output labels a "!" (e.g., $b!$). Determinism is enforced on outputs: for any state ℓ, the guards of outgoing *output* edges from ℓ are pairwise disjoint, so at most one output can be enabled at any time. Moreover, an upper bound on the delay before producing an output is required (the *k-urgency* assumption). In contrast, our DRTA framework does not separate Σ into inputs/outputs; traces are classified as accepting/rejecting via positive/negative samples. Hence our raw traces (without "?/!" annotations) cannot be used directly in Tappler's implementation unless converted to pure-input sequences.

Data usage. Tappler's method relies exclusively on positive samples, motivated by the fact that real system testing can only observe the executions produced by the system, without access to negative traces. To avoid overgeneralization in such a setting, their method introduces an additional *k-urgency* assumption, i.e., the requirement that every output edge must be associated with an upper bound on its timing interval. Our framework, by contrast, directly exploits both positive and negative samples, enabling us to eliminate spurious behaviors without additional assumptions. This fundamental difference implies that our negative-sample data cannot be directly input into their algorithm.

Constraint formulation. Tappler's method explicitly encodes every step of each trace into SMT formulas. While this enables the use of powerful SMT solvers, it also introduces substantial redundancy, causing the number of encoding variables to increase rapidly with the number and length of the samples. In contrast, our 3DRTA-based encoding eliminates prefix redundancies through region-based reasoning and achieves the same expressive power with significantly fewer constraints, while still preserving determinism and completeness.

Practical implications. Given these differences, our experimental samples cannot be directly applied to Tappler's method: the lack of negative-sample handling, the input/output separation, and the *k-urgency* requirement create fundamental mismatches in problem settings. Considering these issues, our experiments in Section 7 do not compare our approach with Tappler's method. Therefore, instead, we compare the number of constraints in the encoded formula. For a fair comparison of the constraint sizes, we reformulate our data into

Algorithm 3: Region Merging for DRTA

Input: Δ: The set of all transitions (q, σ, I, q')
Output : Opt: optimized mapping $(q, \sigma) \mapsto \{q' \mapsto \mathcal{I}\}$

```
Opt ← ∅;
G ← GroupBy(Δ, keys = (q, σ));
foreach (q, σ) ∈ keys(G) do
    B ← GroupBy(G[(q, σ)], keys = q′);
    if |B| = 1 then                                   ▷ Only one target
        Opt[(q, σ)] ← {q′ ↦ {[0, ∞)}};
    else
        𝔓 ← ExtractProtectionPoints(B);     ▷ Get protection points
        B ← Optimization(B, 𝔓);
        H ← FindGaps(B);            ▷ Identify uncovered time intervals
        B ← FillGaps(B, H, 𝔓);                  ▷ Fill missing regions
        Opt[(q, σ)] ← B;
return Opt
```

the input format supported by their algorithm and align the number of states. As shown in Table 1, our 3DRTA-based encoding consistently yields a substantially smaller number of constraints compared to Tappler's SMT encoding across all sample sizes.

With increasing sample scale, the gap between the two approaches becomes more pronounced, highlighting the advantage of our method in producing more compact encoding and enabling more efficient solving in practice.

6 Region Merging for DRTAs

We address the optimization of a DRTA $\mathcal{A}$ by merging time regions on transitions while preserving language semantics. Let $\Delta \subseteq Q \times (\Sigma \times [\mathbb{R}_{\geq 0}]) \times Q$ be the transition relation of $\mathcal{A}$, with $|\Delta| < \infty$. Given $q, q' \in Q$ and $\sigma \in \Sigma$, we define $\Delta(q, \sigma, q') = \{\, I \mid (q, \sigma, I, q') \in \Delta \,\}$ and $\Delta(q, \sigma) = \{\, (I, q') \mid (q, \sigma, I, q') \in \Delta \,\}$. Our goal is to minimize the number of regions used by Δ while ensuring (i) determinism, (ii) completeness, and (iii) protection, introduced below.

Constraints. Let $q \in Q$ and $\sigma \in \Sigma$; then we enforce the following constraints. (i) *Determinism*: the sets of time regions assigned to distinct targets are pairwise disjoint: for all $q'_1 \neq q'_2$, $I_1 \in \Delta(q, \sigma, q'_1)$, and $I_2 \in \Delta(q, \sigma, q'_2)$, we require $I_1 \cap I_2 = \emptyset$. (ii) *Completeness*: $\bigcup_{(I,q') \in \Delta(q,\sigma)} I = \mathbb{R}_{\geq 0}$. (iii) *Protection*: let $\mathfrak{P}(q, \sigma) = \{\, \tau \in \mathbb{N} \mid \exists q' \in Q : (q, \sigma, [\tau, \tau], q') \in \Delta \,\}$ be the set of *protection points*, i.e., timestamps τ such that $(q, \sigma, [\tau, \tau], q') \in \Delta$ for some state q'. These protection points are integer timestamps that must remain mapped to their original successor state during region merging, i.e., the exact transition $(q, \sigma, [\tau, \tau], q')$ cannot be reassigned to a different $q'' \neq q'$.

Workflow. The overall framework of the region merging algorithm is shown in Algorithm 3. For each pair (q, σ) we aggregate $\Delta(q, \sigma)$ and work locally. If

$\Delta(q,\sigma)$ has a single target q', we can replace all its regions by the maximal region $[0,\infty)$ assigned to q' (this is safe by determinism and preserves completeness). Otherwise, we (1) compute the protection map $\pi\colon \mathfrak{P}(q,\sigma) \to Q$; each protection point is mapped to its unique target state q', and it must not be included in any transition leading to a state other than q'. (2) We run a protection-aware merging routine within each target q' to coalesce adjacent or overlapping regions whenever doing so does not capture a foreign protection point. We *split* any region I that straddles a protected point $\tau \in \mathfrak{P}(q,\sigma)$ into subregions whose interiors exclude τ, so that no subsequent merge can violate protection. Finally, (3) we *fill gaps* to restore completeness by assigning uncovered intervals to adjacent targets according to a priority heuristic.

Protection-aware merging. Given $\mathfrak{P}(q,\sigma)$, fix a target $s \in Q$ and consider two candidate regions $R_1, R_2 \in \Delta(q,\sigma,s)$ with $R_1 \cap R_2 \neq \emptyset$ or sharing endpoints. Define $R_{\mathrm{m}} = \mathrm{hull}(R_1 \cup R_2)$, where hull denotes the interval hull, i.e., the smallest interval covering the two regions. We allow the merge if and only if no protection point belonging to a different target would be newly engulfed:

$$\mathrm{CanMerge}(R_1,R_2) := \nexists p \in \mathfrak{P}(q,\sigma) : p \in R_{\mathrm{m}} \wedge p \notin (R_1 \cup R_2) \wedge \pi(p) \neq s.$$

If CanMerge holds, we replace R_1 and R_2 by R_{m}; otherwise, we remove this protection point from the merged region R_{m}.

Gap filling. To satisfy completeness, let $\mathsf{Gaps}(q,\sigma)$ denote the (finite) set of *uncovered intervals*, i.e., maximal contiguous sub-intervals of $\mathbb{R}_{\geq 0}$ that are not contained in any interval I with $(I,q') \in \Delta(q,\sigma)$. For a gap g and a candidate target s we define their priority as

$$\mathrm{Priority}(g,s) = \begin{cases} 2 & \text{if } g \text{ is lower-adjacent to some } I \in \Delta(q,\sigma,s), \\ 1 & \text{if } g \text{ is upper-adjacent to some } I \in \Delta(q,\sigma,s), \\ \frac{1}{\mathrm{dist}(g,s)+\varepsilon} & \text{otherwise,} \end{cases}$$

where $\varepsilon > 0$ is a fixed small constant (e.g., 10^{-6}), used only to avoid division by zero when $\mathrm{dist}(g,s) = 0$. Its precise value does not affect the relative ordering of priorities, as long as it is sufficiently small. The distance dist is defined as

$$\mathrm{dist}(g,s) = \min\{\, |e - e'| \mid e \in \mathrm{E}(g) \wedge e' \in \mathrm{E}(\Delta(q,\sigma,s)) \,\}$$

where $\mathrm{E}(g)$ denotes the set of the two endpoints of the gap g and $\mathrm{E}(\Delta(q,\sigma,s))$ denotes the set of all endpoints of intervals $I \in \Delta(q,\sigma,s)$. Thus, $\mathrm{dist}(g,s)$ is the minimal absolute distance between any endpoint of g and any endpoint of an interval associated with target state s. We assign each g to the target s having maximum priority. In case of ties, we apply the *minimum-color-first rule*, meaning that among candidate targets we select the one that currently has the smallest number of assigned intervals (i.e., the "least colored" state); if a tie still occurs, we resolve it by taking arbitrarily one the targets in the tie. This ensures a balanced allocation of gaps across targets and avoids unnecessary region proliferation. Assignments never include a protected point with $\pi(p) \neq s$.

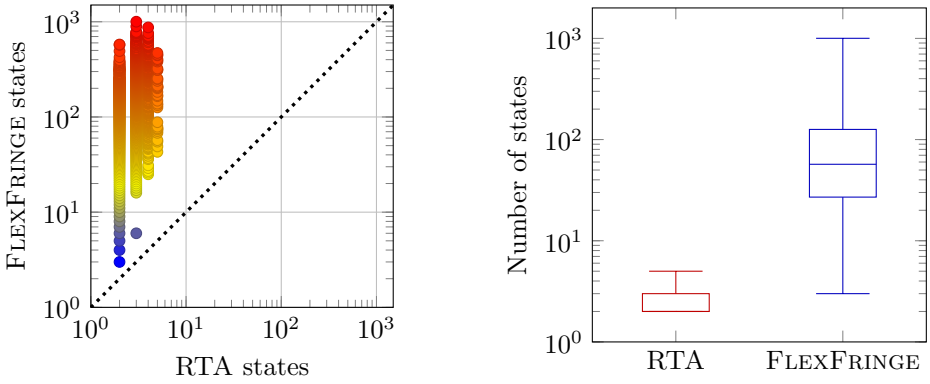

Fig. 3. State comparison between RTA and FlexFringe

For example, suppose $\Delta(q, \sigma)$ relates $[1, 3]$, $[2, 4)$, $[3, 5]$ to q_1 and the point $[4, 4]$ to q_2. The procedure protects 4, splits q_1's regions around 4, merges the remainder into $[0, 4) \cup (4, \infty)$ for q_1, and keeps $[4, 4]$ for q_2, thereby preserving determinism and completeness with a minimal number of regions.

Complexity. Let n denote the number of initial regions for a fixed (q, σ). With sorting and linear scans inside each target, the overall running time is $O(n^2)$ in the worst case (dominated by guarded merge checks) and $O(n)$ space. Empirically, this yields a substantial reduction in the number of regions while maintaining the original transition semantics.

7 Experimental Evaluation

We have implemented our approach in a Python-based prototype[1] and generated randomly timed automata of different sizes that we used to sample random traces then used as input benchmarks. The automata have between 3 and 30 states and between 2 and 8 letters; the produced benchmarks have between 5 and 1000 samples, for a total of 6972 instances. Let FlexFringe and RTA denote the tools implementing the original RTI [35] and our RTA approaches, respectively. We ran the tools on a desktop machine with an i5-12400F CPU and 16 GB of memory running Ubuntu 24.04.3; we used BenchExec [8] to get reliable benchmark outcomes.

In general, FlexFringe is faster than RTA, but it synthesizes DRTAs that are much larger. Regarding the running time, both tools were able to generate DRTAs for the benchmarks well within the 30 minutes time limit we imposed in BenchExec; when the generated 3DRTA contains a large number of states, the SAT encoding from Section 5.1 contains a huge number of variables and constraints, which would take Z3 [25], the SAT solver we used, a large amount

[1] See https://github.com/Sakura-0407/VMCAI_Synthesis-of-Minimal-DRTA

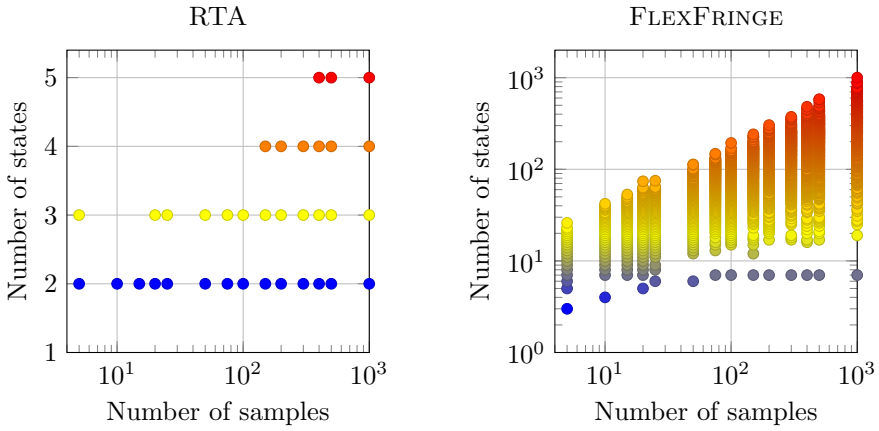

Fig. 4. State distribution vs. number of samples by RTA and FlexFringe

of time to check its satisfiability. Still, all benchmarks have been completed in less than 15 minutes each.

Regarding the size of the synthesized DRTAs, the plots in Fig. 3 show the comparison between the number of states produced by our RTA and the original FlexFringe, as well their state distributions; we use logarithmic axes in both plots. As we can see from the scatter plot on the left part of Fig. 3, RTA consistently produces fewer states than FlexFringe, since there is not a single mark on or below the dashed diagonal line. Moreover, RTA is able to generate DRTAs with much fewer states than FlexFringe: for instance, on one benchmark RTA needed just 2 states while FlexFringe used 577 states while on another benchmark the states were 3 and 1002, respectively. This is confirmed by the box plots on the right part of Fig. 3: all solved cases by RTA needed at most 5 states, while on the same benchmarks FlexFringe reached 1002 states, with the central 50% of the synthesized DRTAs having between 27 and 126 states.

Fig. 4 shows the distribution of the number of states with respect to the number of samples. As one would expect, and as confirmed by the plots, when we have more samples, usually we need more states to make the DRTA consistent with them. However, RTA is able to achieve this with much fewer states than FlexFringe, even with very few samples like 5 or 10.

These experiments confirm that the synthesis procedure we proposed in Sections 3–6 yields DRTAs that are much smaller than the ones produced by FlexFringe based on heuristics.

8 Conclusion and Future Work

In this paper, we presented a systematic framework for synthesizing minimal deterministic real-time automata (DRTAs) from positive and negative timed traces. Our approach is centered around three main contributions. First, we introduced the 3DRTA as a novel intermediate representation that extends the idea of 3-valued automata to the timed setting, allowing timing constraints to be encoded compactly while avoiding the redundancies inherent in prefix-tree constructions. Second, we developed a SAT-based synthesis procedure that incrementally explores the state space and guarantees minimality of the resulting DRTA. Third, we proposed a region merging algorithm that reduces the number of temporal regions in transitions without violating determinism, completeness, or exact-point semantics. Through extensive experiments, we demonstrated that our 3DRTA-based encoding generates substantially fewer constraints than Tappler's SMT-based method, thereby yielding a more compact encoding and significantly improving solving efficiency. In addition, compared to the RTI algorithm, which relies on heuristic evidence-driven state merging, our approach is able to synthesize DRTAs with much smaller state spaces. These results highlight the advantage of combining the 3DRTA representation with SAT-based synthesis, offering both theoretical minimality guarantees and practical scalability.

Although our work provides a solid theoretical and empirical foundation, several directions remain open for future research. One promising avenue is to improve robustness against noisy or incomplete data, where the strict separation of positive and negative samples may not hold in practice. Another direction is to extend our framework to richer classes of timed automata, such as event-recording automata or multi-clock models, and to investigate how the principles of 3DRTA encoding can be generalized. Further work may also focus on tighter integration with modern SAT/SMT solvers, for example by adopting incremental encodings or solver-guided heuristics to accelerate synthesis. Adapting the existing divide-and-conquer strategies for logic learning [26,29] to accelerate automata learning is also an interesting direction. Finally, large-scale case studies in real-world verification and system identification would provide valuable insights into the applicability and performance of our method in industrial contexts.

Acknowledgments. We thank the anonymous reviewers for their useful remarks that helped us improve the quality of the paper.
Work supported in part by the National Key R&D Program of China (Grant No. 2022YFA1005101), by the National Natural Science Foundation of China (NSFC) (Grants Nos. W2511064, 62472316, 62192732, 62032024, and 62032019), by the ISCAS Basic Research (Grant Nos. ISCAS-JCZD-202406 and ISCAS-JCZD-202302), by the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-040), and by the ISCAS New Cultivation Project ISCAS-PYFX-202201.
This project is part of the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant no. 101008233.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2020. LNCS, vol. 12078, pp. 444–462. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_25
3. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. Sci. China Inf. Sci. **64**(9) (2021). https://doi.org/10.1007/s11432-019-2767-4
4. An, J., Zhan, B., Zhan, N., Zhang, M.: Learning nondeterministic real-time automata. ACM Trans. Embed. Comput. Syst. **20**(5s), 99:1–99:26 (2021). https://doi.org/10.1145/3477030
5. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
6. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Priced timed automata: Algorithms and applications. In: 3rd International Symposium on Formal Methods for Components and Objects, FMCO 2004. LNCS, vol. 3657, pp. 162–182. Springer (2004). https://doi.org/10.1007/11561163_8
7. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets, LNCS, vol. 3098, pp. 87–124. Springer (2004). https://doi.org/10.1007/978-3-540-27755-2_3
8. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. Int. J. Softw. Tools Technol. Transf. **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y
9. Bouyer, P., Laroussinie, F., Reynier, P.: Diagonal constraints in timed automata: Forward analysis of timed systems. In: 3rd International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2005. LNCS, vol. 3829, pp. 59–73. Springer (2005). https://doi.org/10.1007/11603009_10
10. Cassez, F., Jensen, P.G., Larsen, K.G.: Verification and parameter synthesis for real-time programs using refinement of trace abstraction. Fundamenta Informaticae **178**(1-2), 31–57 (2021). https://doi.org/10.3233/FI-2021-1997
11. Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verification. In: 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2009. LNCS, vol. 5505, pp. 31–45. Springer (2009). https://doi.org/10.1007/978-3-642-00768-2_3
12. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite state automata. Comput. Linguistics **26**(1), 3–16 (2000). https://doi.org/10.1162/089120100561601
13. D'Antoni, L., Veanes, M.: Minimization of symbolic automata. In: Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2014. pp. 541–553. ACM (2014). https://doi.org/10.1145/2535838.2535849
14. Dell'Erba, D., Li, Y., Schewe, S.: DFAMiner: Mining minimal separating DFAs from labelled samples. In: 26th International Symposium on Formal Methods, FM 2024 (2). LNCS, vol. 14934, pp. 48–66. Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_4
15. Dima, C.: Real-time automata. Journal of Automata, Languages and Combinatorics **6**(1), 3–23 (2001). https://doi.org/10.25596/jalc-2001-003

16. Fisman, D., Frenkel, H., Zilles, S.: Inferring symbolic automata. Logical Methods in Computer Science **19**(2), 5:1–5:37 (2023). https://doi.org/10.46298/LMCS-19(2:5)2023
17. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3), 302–320 (1978). https://doi.org/10.1016/S0019-9958(78)90562-4
18. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theoretical Computer Science **411**(40–42), 4029–4054 (2010). https://doi.org/10.1016/J.TCS.2010.07.008
19. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: 3rd International Joint Conference on Automated Reasoning, IJCAR 2006. LNCS, vol. 4130, pp. 483–497. Springer (2006). https://doi.org/10.1007/11814771_40
20. Hessel, A., Larsen, K.G., Nielsen, B., Pettersson, P., Skou, A.: Time-optimal real-time test case generation using UPPAAL. In: 3rd International Workshop on Formal Approaches to Testing of Software, FATES 2003. LNCS, vol. 2931, pp. 136–151. Springer (2003). https://doi.org/10.1007/978-3-540-24617-6_9
21. Heule, M., Verwer, S.: Exact DFA identification using SAT solvers. In: 10th International Colloquium on Grammatical Inference, ICGI 2010. LNCS, vol. 6339, pp. 66–79. Springer (2010). https://doi.org/10.1007/978-3-642-15488-1_7
22. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognit. **38**(9), 1332–1348 (2005). https://doi.org/10.1016/J.PATCOG.2005.01.003
23. Jiang, Z., Pajic, M., Sokolsky, O., Lee, I.: Modeling and verification of a dual chamber implantable pacemaker. In: 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2012. LNCS, vol. 7214, pp. 188–203. Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_14
24. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: 4th International Colloquium on Grammatical Inference, ICGI 1998. LNCS, vol. 1433, pp. 1–12. Springer (1998). https://doi.org/10.1007/BFB0054059
25. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
26. Neider, D., Gavran, I.: Learning linear temporal properties. In: 2018 Formal Methods in Computer Aided Design, FMCAD 2018. pp. 1–10. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603016
27. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: 20th IEEE Symposium on Logic in Computer Science, LICS 2005. pp. 188–197. IEEE Computer Society (2005). https://doi.org/10.1109/LICS.2005.33
28. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. In: 21st Annual ACM Symposium on Theory of Computing, STOC 1989. pp. 421–432. ACM (1989). https://doi.org/10.1145/73007.73048
29. Riener, H.: Exact synthesis of LTL properties from traces. In: 2019 Forum for Specification and Design Languages, FDL 2019. pp. 1–6. IEEE (2019). https://doi.org/10.1109/FDL.2019.8876900
30. Shen, W., An, J., Zhan, B., Zhang, M., Xue, B., Zhan, N.: PAC learning of deterministic one-clock timed automata. In: 22nd International Conference on For-

mal Engineering Methods, ICFEM 2020. LNCS, vol. 12531, pp. 129–146. Springer (2020). https://doi.org/10.1007/978-3-030-63406-3_8
31. Tang, X., Shen, W., Zhang, M., An, J., Zhan, B., Zhan, N.: Learning deterministic one-clock timed automata via mutation testing. In: 20th International Symposium on Automated Technology for Verification and Analysis, ATVA 2022. LNCS, vol. 13505, pp. 233–248. Springer (2022). https://doi.org/10.1007/978-3-031-19992-9_15
32. Tappler, M., Aichernig, B.K., Lorber, F.: Timed automata learning via SMT solving. In: 14th International Symposium on NASA Formal Methods, NFM 2022). LNCS, vol. 13260, pp. 489–507. Springer (2022). https://doi.org/10.1007/978-3-031-06773-0_26
33. Teng, Y., Zhang, M., An, J.: Learning deterministic multi-clock timed automata. In: Ábrahám, E., Jr., M.M. (eds.) 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024. pp. 6:1–6:11. ACM (2024). https://doi.org/10.1145/3641513.3650124
34. 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). https://doi.org/10.1016/J.IC.2010.11.023
35. Verwer, S., de Weerdt, M., Witteveen, C.: Efficiently identifying deterministic real-time automata from labeled data. Machine Learning **86**(3), 295–333 (2012). https://doi.org/10.1007/S10994-011-5265-4
36. Waga, M.: Active learning of deterministic timed automata with Myhill-Nerode style characterization. In: Enea, C., Lal, A. (eds.) CAV 2023. Lecture Notes in Computer Science, vol. 13964, pp. 3–26. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_1
37. Wang, L., Zhan, N., An, J.: The opacity of real-time automata. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11), 2845–2856 (2018). https://doi.org/10.1109/TCAD.2018.2857363
38. Zakirzyanov, I., Shalyto, A., Ulyantsev, V.: Finding all minimum-size DFA consistent with given examples: SAT-based approach. In: Software Engineering and Formal Methods - SEFM 2017 Collocated Workshops: DataMod. LNCS, vol. 10729, pp. 117–131. Springer (2017). https://doi.org/10.1007/978-3-319-74781-1_9